A Configurable Reference Modelling Language[1]

M. Rosemann[a], W.M.P. van der Aalst[b,a]

[a] *Centre for Information Technology Innovation, Faculty of Information Technology, Queensland University of Technology, 126 Margaret Street, Brisbane Qld 4000, Australia, phone: +61 7 3864 9473, fax +61 7 3864 9390, m.rosemann@qut.edu.au*

[b] *Faculty of Technology and Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, w.m.p.v.d.aalst@tm.tue.nl*

Abstract

Enterprise Systems (ES) are comprehensive off-the-shelf packages that have to be configured to suit the requirements of an organization. Most ES solutions provide reference models that describe the functionality and structure of the system. However, these models do not capture the potential configuration alternatives. This paper discusses the shortcomings of current reference modelling languages using Event-Driven Process Chains (EPCs) as an example. We propose Configurable Event-Driven Process Chains as an extended reference modelling language which allows capturing the core configuration patterns. A formalization of this language as well as examples for typical configurations are provided. A program of further research including the identification of a comprehensive list of configuration patterns, deriving possible notations for reference model configurations and testing the quality of these proposed extensions in experiments and focus groups is presented.

Keywords

Reference Model, Enterprise Systems, Configuration, Event-Driven Process Chains

---

[1] This research project is financially supported by SAP Research.

1 Introduction

Enterprise Systems (ES) offer business solutions for typical functional areas such as procurement, materials management, production, sales and distribution, financial accounting and human resource management [24, 32]. These functions are typically individualised for countries and industries, *e.g.* automotive, retailing, high-tech. Such off-the-shelf-solutions require configuration before they can be used in the individual context of an organization.

As an approach to improve the understandability of these systems and to stress the process-oriented nature of their solutions, ES vendors have developed application reference models which describe the processes and structure of the system. Enterprise Systems reference models exist in the form of function, data, system organization, object and business process models, although the latter are by far the most popular type. Current reference models, however, are based on conventional modelling languages that have been developed for the design of enterprise-individual models. Thus, they are not able to adequately depict possible system configurations. Even further, they don't provide decision support regarding the selection of relevant variants. Current application reference models "just" depict the sum of all possible system capabilities and cannot sufficiently deal with the requirement of optionality.

This paper contributes to this area by extending an existing process modelling language used in the by far most successful Enterprise System (SAP) with configurable elements. Thus, it becomes possible to clearly highlight the required decisions which have to be made at build-time, *i.e.* during the individualisation of the generic models.

While our selection of the process modelling language has been driven by its popularity in the relevant Enterprise Systems modelling practice, the proposed extensions can be easily adapted to other modelling techniques (*e.g.* UML or Petri-Nets). Furthermore, we contribute to the area of Enterprise Systems modelling by providing a comprehensive list of criteria, which have to be satisfied by configurable process modelling languages.
This paper is structured as follows. The next section provides an overview about the characteristics of application reference models and gives an example for such a model (the so-called EPC model) and its current shortcomings. The third section outlines the research problem and the research methodology. Section four lists the requirements for a configurable reference modelling technique. Section 5 first formalizes the notion of Event-Driven Process Chains (EPCs), followed by a presentation of Configurable EPCs (C-EPCs). This paper ends with a section on related work, a brief summary and a discussion of the future work.

2 Reference Models

Reference models are generic conceptual models that formalize recommended practices for a certain domain [16,18]. Often labelled with the term 'best practice' reference models claim to capture reusable state-of-the-art practices [39,40]. The depicted domains can be very different and range from selected functional areas such as financial accounting or Customer Relationship Management to the scope of an entire industry sector, *e.g.* higher education.

The main objective of reference models is to streamline the design of enterprise-individual (particular) models by providing a generic solution. The application of reference models is motivated by the ‘Design by Reuse’ paradigm. Reference models

accelerate the modelling process by providing a repository of potentially relevant business processes and structures. These ideally 'plug and play' models are also called Partial Enterprise Models in the terminology of the Generalised Enterprise Reference Architecture and Methodology (GERAM) [7].

Reference models can be differentiated along the following main criteria

- scope of the model (*e.g.*, functional areas covered)
- granularity of the model (*e.g.*, number of levels of decomposition detail)
- views (*e.g.*, process, data, objects, organization) that are depicted in the model
- degree of integration between the views
- purposes supported
- user groups addressed
- internal or external (commercial) use
- availability of the model (*e.g.*, paper, tool-based, Web-based)
- availability of further textual explanation of the model
- explicit inclusion of alternative business scenarios
- existence of guidelines on how to use these models
- availability of relevant quantitative benchmarking data

A further and more comprehensive differentiation based upon the *domain* that underlies the reference model can be found in [6 and 34].

The term reference model is also used for models describing the structure and functionality of business applications including Enterprise Systems [11]. In these cases, a reference model can be interpreted as a structured semi-formal description of a particular application. Application reference models correspond to an existing off-the-shelf-solution that supports the functionality and structure described in the model. They can be used for a better understanding and evaluation of the appropriateness of the software.

Furthermore, they aim to facilitate the implementation of the software and can be used for related end user training [6,20,33].

One of the most comprehensive models 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. Most of the other market leading ES vendors have an approach towards such reference models. An overview of the Baan reference model, for example, is provided in [43]. See also [17] for reference models in Intentia.

Foundational conceptual work for the SAP reference model had been conducted by SAP AG and the IDS Scheer AG in a collaborative research project in the years 1990-1992 [23]. The outcome of this project was the process modelling language *Event-Driven Process Chains* (EPCs) [23, 38], which has been used for the design of the reference process models in SAP. EPCs also became the core modelling language in the Architecture of Integrated Information Systems (ARIS) [38]. It is now one of the most popular reference modelling languages and has also been used for the design of many SAP-independent reference models (*e.g.*, ARIS-based reference model for Siebel CRM or industry models for banking, retail, insurance, telecommunication, etc.).

EPCs are directed graphs, which visualise the control flow and consist of events, functions and connectors. Each EPC starts with at least one event and ends with at least one event [1, 3, 11, 23, 29, 36, 38]. An event triggers a function, which leads to a new event. Three types of connectors (AND, Exclusive OR, OR) can be used to model splits and joins. Figure 1 shows an example for an EPC as it potentially can be found as part of the SAP reference model. This model shows a part of the invoice verification process.

The EPC contains 10 events, 4 functions, and 4 connectors. The events can be seen as pre- and/or post-conditions of functions. For example, function *Release invoice* can be executed if event *Invoice posted* is received and the completion of this function will trigger the event *Payment must be effected*. There are three functions triggering event *Invoice posted*. The exclusive OR connector in the centre of the diagram shows that there is no need to synchronize these three functions, *e.g.*, the completion of *Evaluated Receipt Settlement* directly triggers event *Invoice posted*. The three other connectors are AND connectors. Hence, both input events need to be triggered in order to enable function *Evaluated Receipt Settlement*.

Figure 1 shows a classical EPC which does not contain any configuration information. Therefore, valuable information is lacking. For example, it is not shown that *Evaluated Receipt Settlement*, *i.e.* the scenario in which payment for goods is triggered through goods issue, is only of interest for a subset of all SAP customers. This implies that the exclusive OR connector may be a choice made at design time for the whole process rather than at run-time for an individual process instance. It also does not show that *Invoicing Plan Settlement* is only relevant, if *Evaluated Receipt Settlement* is conducted. Note that because of the exclusive OR connector only one of them is executed but it does not make any sense to allow for *Invoicing Plan Settlement* if *Evaluated Receipt Settlement* is not allowed. Moreover, the model does not give any insights into the necessity or criticality of the possible configurations. Thus, the model expressive power is limited and does not address the possible configurations of the corresponding Enterprise System.

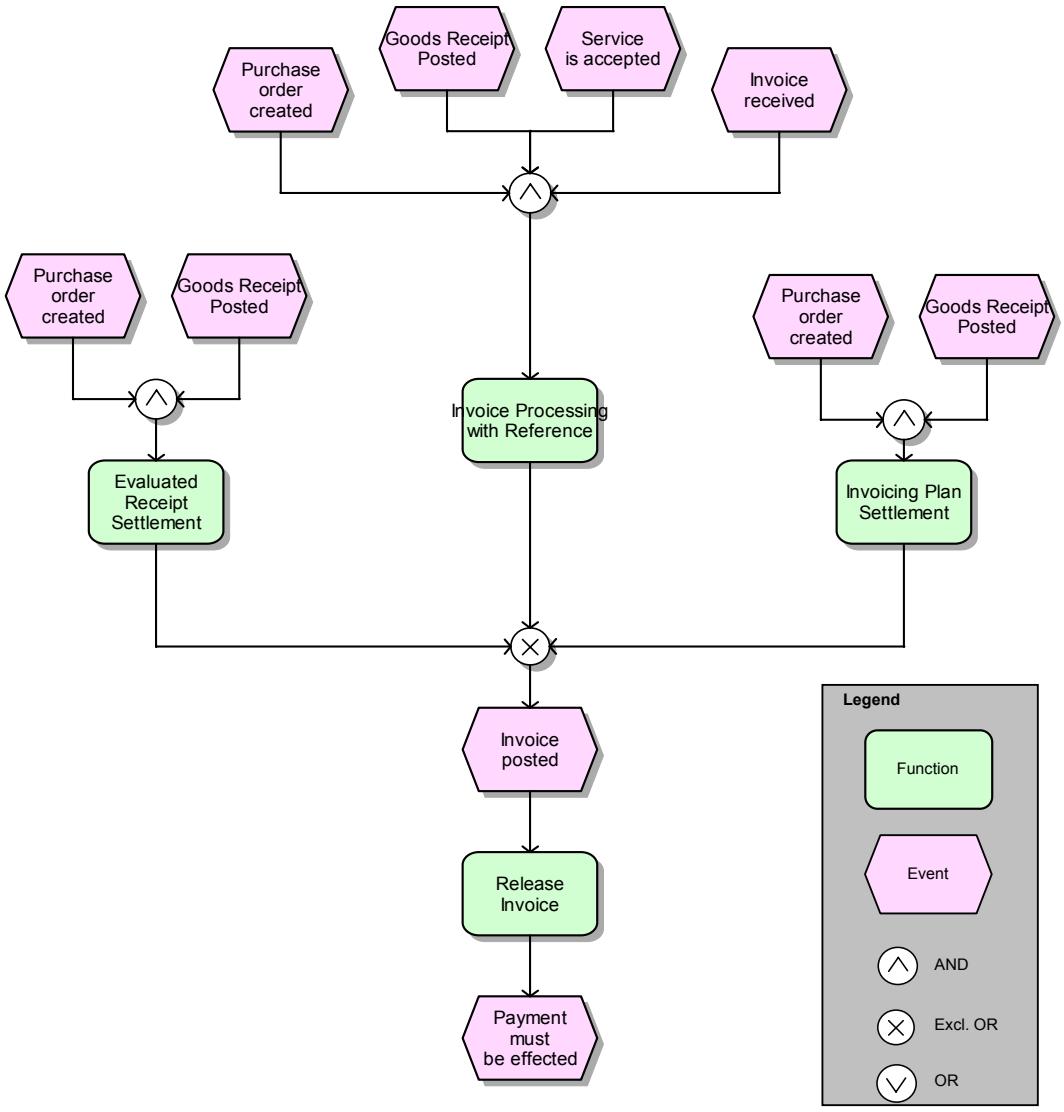

Fig. 1: An example for a reference model in EPC-notation

3 Research Problem and Research Methodology

The existence of reference models highlights a difference from the traditional software development process. Instead of starting from scratch and continuously adding functionality, ES solutions require a continuous narrowing down of the scope of the system. This process starts with the "big picture", which is then reduced to the relevant part. Reference models can be used as a description of this big picture. It is necessary to

select the necessary functions and to decide during the configuration process between alternatives (*e.g.*, reporting in financial accounting or controlling).

The reference model lifecycle is initiated by the reference model designers, *i.e.* the Enterprise Systems vendor. During the design phase available individual conceptual models are evaluated, selected and consolidated.[2] Such a reference model will typically not only include one proposed alternative, but a range of often mutually exclusive alternatives. This might be because the depicted scenarios cover different industries or different countries. At this stage, for example, SAP maintains 27 alternative industry solutions. However, the current use of traditional modelling languages does not support a consolidation of these models. Figure 2 demonstrates this problem in a simple example. It shows the consolidation of corresponding reference models from two different industries. The XOR split (second from the left) in this case represents a decision point that is of relevance during the so-called configuration time. A model in this phase cannot necessarily be executed. It rather captures different alternatives for a domain and has to be configured before it can serve as the actual build time model for individual process instances.

[2] An organization might also declare the internal best practice in one subsidiary etc. as the internal benchmark. Thus, an existing conceptual model can have the status of a reference model. This practice can, for example, be observed in global organizations that roll-out the business blueprint of one location to all their subsidiaries worldwide. These models are also called prototypical models [7]. They do *not* require configurations and are not within the scope of this paper.

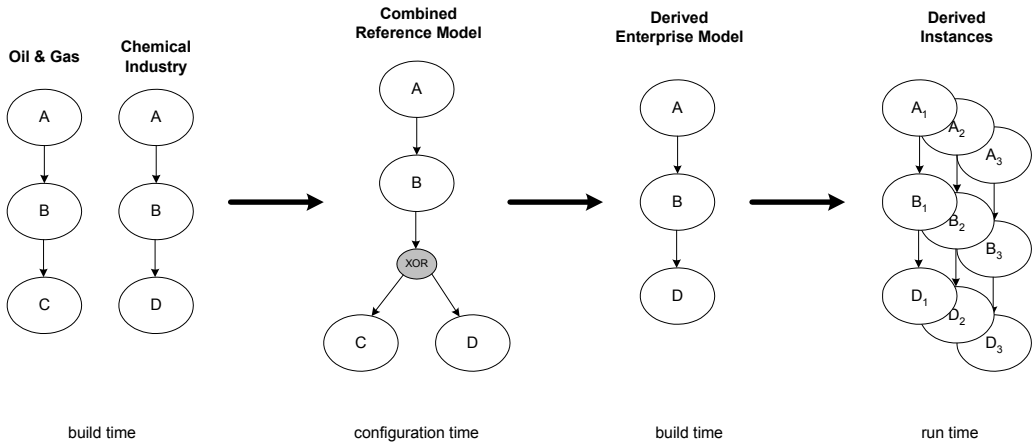

Fig. 2: Configuration time, build time and run time

The lack of the required expressiveness of current reference modelling languages for configuration time is for two reasons a serious issue for model users. First, it does not become obvious what configuration alternatives exist during the system implementation phase. Second, the models do not provide any decision support in the actual selection of an alternative. Current reference models show what processes are supported in general, but not what might be a recommended alternative. They represent the entire functionality from the viewpoint that the complete system is used and look like an ordinary build time model. However, only a subset is typically used within an individual organization. Figure 3 provides a related example for the customer master data entry process in SAP R/3. It becomes obvious that the actually selected and implemented business process is less than 50 % of the provided reference model. The connectors in the process model on the left, however, do not differentiate between decisions which configure the model and decisions at run time. As a result all connectors seem to refer to run time decisions. Consequently, a model user has problems in identifying the configurable parts in this process.

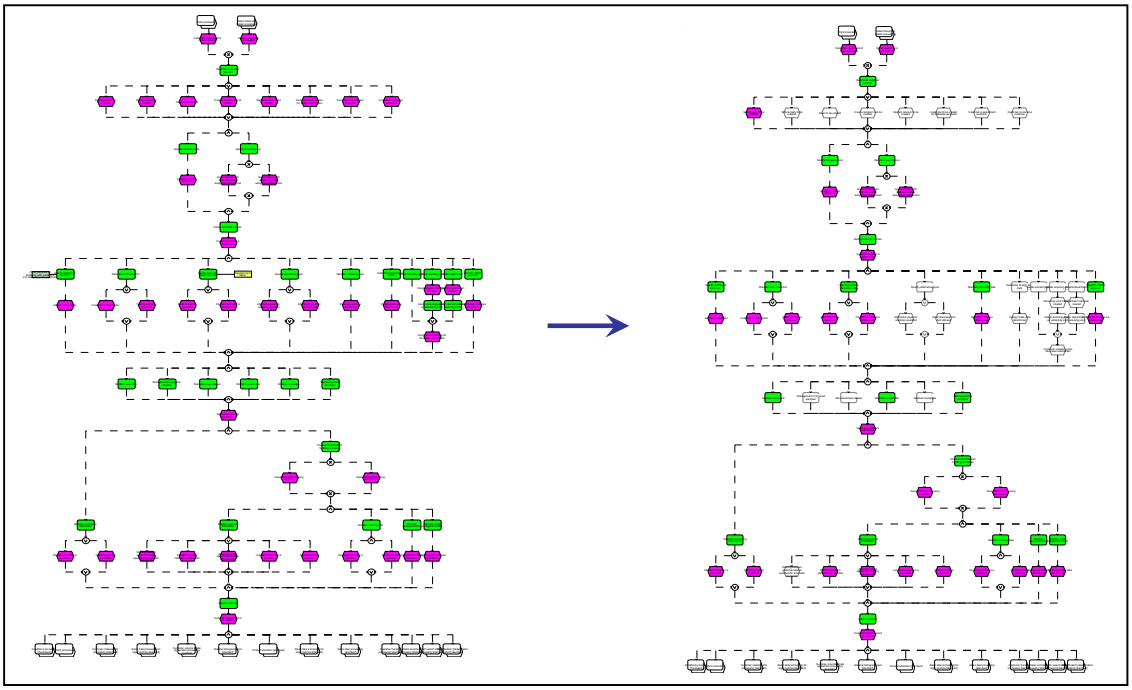

Fig. 3: Individualisation of a SAP reference model

The main objective of this paper is to propose a new configurable reference modelling language by extending an existing popular modelling technique. This research is embedded in a more comprehensive SAP-funded research project with the following three phases.

1) The first task was the identification and classification of *configuration patterns*. A configuration pattern describes a distinguishable configuration case. Based on the work that has been conducted on workflow patterns [4], we derived a set of configuration patterns that classify alternative configuration scenarios. As far as possible, examples from the SAP reference model have been assigned to each configuration pattern. The SAP reference model has been used because of its maturity, its worldwide use and its availability to the researchers.

2) The next step has been the *development and formalization of a new and dedicated reference modelling language*, which supports the specification of these configuration patterns. This task has been constrained by the desire to rather extend current reference modelling languages than to develop an entire new language. This has been motivated by the significant development efforts that have been invested in reference models already. We selected EPCs as the starting point for our research due to the popularity of this language for the design of reference models.

3) The proposed configurable reference modelling language and the corresponding notation *will be rigorously tested* in two ways. First, experiments with experienced business process analysts will be conducted. The selected group of analysts will be familiar with SAP, process modelling and reference modelling. These studies will examine the quality and time of comprehension using a configurable EPC in comparison with current process models. Second, focus groups with SAP application consultants who are using the SAP reference model in their consulting practice will be conducted in order to further explore issues related to the acceptance of the proposed new modelling techniques.

This paper reports on the second phase, *i.e.* the proposed configurable reference modelling language. This language is only focused on the so called essential configurations, *i.e.* the system variability as it is visible and relevant to the project team, and not the technical configurations, which subsume aspects related to the technical realisation [21].

4 Requirements for a configurable reference modelling technique

Reference modelling languages have to be configurable. A configurable modelling language is characterised by its capability to support decisions at build time, *i.e.* the model user can individualise the model by selecting from alternative options before instances will be derived from it. This means that they should not only capture decisions on an instance level, but also on a type level. Unlike decisions on an instance level, *i.e.* at runtime, decisions on a type level, *i.e.* at build time, have an impact on the model and its actual structure. Such configuration decisions have to be clearly differentiated from runtime decisions and can be highlighted as *variation points* in a model [21]. A variation point captures a decision point together with the related possible choices. Furthermore, a configurable reference modelling language has to consider the following requirements.

a) The language has to support configurations regarding *entire processes, functions, control flow and data*.

b) It should be possible to differentiate configuration decisions into *mandatory and optional* decisions. Mandatory decisions have to be made before the very first instance can be derived from this model. The decision could be not to use a certain variant. Optional decisions can initially be neglected. It should be possible to maintain *defaults* for optional configuration decisions. This allows the instantiation of the model even without explicitly making all possible decisions.

c) Configuration should be differentiated into *global and local* decisions. Global decisions are based on the general context and can be made without studying the individual process model. Such context information includes industry, country, size etc. The relevant context factors have to be maintained for every variation

point. As soon as information regarding the relevant context has been provided, a first (hidden or background) configuration of the reference model can take place. Local configurations require an explicit study of the relevant process model. In these cases the decision maker has to consider the available individual choices and make a trade-off decision.

d) Configuration decisions should also be differentiated into *critical and non-critical* decisions. Critical decisions have significant impact on the use of the system, can often not be re-done and should be made by the project team. Non-critical decisions are of minor importance, can be changed over time and can be made by individual team members.

e) Configuration decisions can have *interrelationships*. Any pre-requisites for a configuration decision should be clearly highlighted. This can include other decisions, which have to be made before. Moreover, any impact of one decision on other decisions has to be depicted. This means, a logical order between configuration decisions has to be considered. This includes interrelationships within one model, between two process models but also interrelationships between a reference process model and a related reference data model [35].

f) Configuration decisions can be made on *different levels*. For example, a first configuration of the SAP reference model might be an individualization for an entire global organization. The next level of configuration can be made for a certain country or business unit.

g) Variation points should *refer to further related information within the Enterprise System*. This can include the system online help and the system configuration module, *i.e.* in SAP the Implementation Guide. Such information can provide valuable support for the decision maker.

h) The entire configuration process should also be *guided by recommendations or configuration guidelines*. Such information could come as benchmarking data from the outside of the system if a critical mass of system users is willing to provide the required data. It can include information such as the processing time of a given process path, the number of times a decision has been made in the same industry or the required investments and implementation time for a certain configuration.

i) Enterprise System reference models are already very comprehensive. Any further extension of these modelling languages has to carefully consider the *impact on the perceived model complexity*.

The following section introduces configurable EPCs as an approach to capture variation points in a reference process model. At the end of the next section we will reflect on the requirements identified.

5 Configurable Event-Driven Process Chains (C-EPCs)

Before introducing Configurable EPCs (C-EPCs), we first formalize the notion of the classical EPC. Then C-EPCs are introduced and formalized followed by a definition of their semantics and a discussion on partially configured C-EPCs. The section is concluded by some reflections on the requirements stated in the previous section.

5.1 Formalization of EPCs

In this section, we give a formal definition of an EPC. This definition is based on the restrictions described in [23] and imposed by tools such as ARIS and SAP R/3 and allows us to specify the requirements an EPC should satisfy. Note that we need to provide a formal definition of EPCs to be able to precisely define the notion of configuration we are after.

Definition 1 *[EPC (1)] An Event-Driven Process Chain is a five-tuple (E,F,C,l,A):*

- E is a finite (non-empty) set of events,
- F is a finite (non-empty) set of functions,
- C is a finite set of logical connectors,
- $l \in C \rightarrow \{\wedge, XOR, \vee\}$ is a function which maps each connector onto a connector type,
- $A \subseteq (E \times F) \cup (F \times E) \cup (E \times C) \cup (C \times E) \cup (F \times C) \cup (C \times F) \cup (C \times C)$ is a set of arcs.

An EPC is composed of three types of nodes: events (*E*), functions (*F*) and connectors (*C*). Figure 1 shows an EPC containing 10 events, 4 functions and 4 connectors. The type of each connector is given by the function *l*: *l(c)* is the type ($\wedge$, *XOR*, or $\vee$) of a connector $c \in C$. Relation *A* specifies the set of arcs connecting functions, events and connectors. Definition 1 shows that it is not allowed to have an arc connecting two functions or two events. There are many more requirements an EPC should satisfy, *e.g.*, only connectors

are allowed to branch, there is at least one start event, there is at least one final event, and there are several limitations with respect to the use of connectors. To formalize these requirements we need to define some additional concepts and introduce some notations.

Definition 2 *[Directed path, elementary path] Let EPC be an Event-Driven Process Chain. A directed path p from a node n_1 to a node n_k is a sequence $\langle n_1, n_2, \ldots, n_k \rangle$ such that $\langle n_i, n_{i+1} \rangle \in A$ for $1 \leq i \leq k-1$.*

The definition of directed path will be used to limit the set of routing constructs that may be used. It also allows for the definition of C_{EF} (the set of connectors on a path from an event to a function) and C_{FE} (the set of connectors on a path from a function to an event). C_{EF} and C_{FE} partition the set of connectors C. Based on the function l we also partition C into $C_\wedge$, $C_\vee$, and C_{XOR}. The sets C_J and C_S are used to classify connectors into join connectors and split connectors.

Definition 3 *[$N, C_\wedge, C_\vee, C_{XOR}, \bullet, C_J, C_S, C_{EF}, C_{FE}$] Let EPC=(E, F, C , l, A) be an Event-Driven Process Chain.*

- *$N = E \cup F \cup C$ is the set of nodes of EPC.*
- *$C_\wedge = \{ c \in C \mid l(c) = \wedge \}$*
- *$C_\vee = \{ c \in C \mid l(c) = \vee \}$*
- *$C_{XOR} = \{ c \in C \mid l(c) = XOR \}$*
- *For $n \in N$:*
  - *$\bullet n = \{ m \mid (m,n) \in A \}$ is the set of input nodes, and*
  - *$n \bullet = \{ m \mid (n,m) \in A \}$ is the set of output nodes.*
- *$C_J = \{ c \in C \mid |\bullet c| \geq 2 \}$ is the set of join connectors.*
- *$C_S = \{ c \in C \mid |c \bullet| \geq 2 \}$ is the set of split connectors.*
- *$C_{EF} \subseteq C$ such that $c \in C_{EF}$ if and only if there is a path $p = \langle n_1, n_2, \ldots, n_{k-1}, n_k \rangle$ such that $n_1 \in E$, $n_2, \ldots, n_{k-1} \in C$, $n_k \in F$, and $c \in \{ n_2, \ldots, n_{k-1} \}$.*
- *$C_{FE} \subseteq C$ such that $c \in C_{FE}$ if and only if there is a path $p = \langle n_1, n_2, \ldots, n_{k-1}, n_k \rangle$ such that $n_1 \in F$, $n_2, \ldots, n_{k-1} \in C$, $n_k \in E$, and $c \in \{ n_2, \ldots, n_{k-1} \}$.*
- *$C_{EE} \subseteq C$ such that $c \in C_{EE}$ if and only if there is a path $p = \langle n_1, n_2, \ldots, n_{k-1}, n_k \rangle$ such that $n_1 \in E$, $n_2, \ldots, n_{k-1} \in C$, $n_k \in E$, and $c \in \{ n_2, \ldots, n_{k-1} \}$.*
- *$C_{FF} \subseteq C$ such that $c \in C_{FF}$ if and only if there is a path $p = \langle n_1, n_2, \ldots, n_{k-1}, n_k \rangle$ such that $n_1 \in F$, $n_2, \ldots, n_{k-1} \in C$, $n_k \in F$, and $c \in \{ n_2, \ldots, n_{k-1} \}$.*

These notations allow for the definition of syntactical correctness for EPCs.

Definition 4 *[EPC (2)] An Event-Driven Process Chain EPC = (E,F,C,l,A) is syntactically correct if and only if the following requirements are satisfied:*

- *The sets E, F, and C are pairwise disjoint, i.e., $E \cap F = \emptyset$, $E \cap C = \emptyset$, and $F \cap C = \emptyset$.*
- *For each $e \in E$: $|\bullet e| \leq 1$ and $|e \bullet| \leq 1$.*
- *There is at least one event $e \in E$ such that $|\bullet e| = 0$ (i.e. a start event).*
- *There is at least one event $e \in E$ such that $|e \bullet| = 0$ (i.e. a final event).*
- *For each $f \in F$: $|\bullet f| = 1$ and $|f \bullet| = 1$.*
- *For each $c \in C$: $|\bullet c| \geq 1$ and $|c \bullet| \geq 1$.*
- *C_J and C_S partition C, i.e., $C_J \cap C_S = \emptyset$ and $C_J \cup C_S = C$.*
- *C_{EE} and C_{FF} are empty, i.e., $C_{EE} = \emptyset$ are $C_{FF} = \emptyset$.*
- *C_{EF} and C_{FE} partition C, i.e., $C_{EF} \cap C_{FE} = \emptyset$ and $C_{EF} \cup C_{FE} = C$.*

The first requirement states that each component has a unique identifier (name). Note that connector names are omitted in the diagram of an EPC. The other requirements correspond to restrictions on the relation *A*. Events cannot have multiple input arcs and there is at least one start event and one final event. Each function has exactly one input arc and one output arc. A connector *c* is either a join connector ($|c \bullet| = 1$ and $|\bullet c| \geq 2$) or a split connector ($|\bullet c| = 1$ and $|c \bullet| \geq 2$). The last requirement states that a connector *c* is either on a path from an event to a function or on a path from a function to an event. Clearly, the EPC shown in Figure 1 satisfies the requirements and is therefore syntactically correct. In the remainder of this paper we assume all EPCs to be syntactically correct.

Note that $\{C_J, C_S\}$, $\{C_{EF}, C_{FE}\}$, and $\{C_\wedge, C_{XOR}, C_\vee\}$ partition *C*, *i.e.*, C_J and C_S are disjoint and $C = C_J \cup C_S$, C_{EF} and C_{FE} are disjoint and $C = C_{EF} \cup C_{FE}$, and $C_\wedge$, C_{XOR} and $C_\vee$ are pair-wise disjoint and $C = C_\wedge \cup C_{XOR} \cup C_\vee$. In principle there are $2 \times 2 \times 3 = 12$ kinds of connectors! In the original definition of EPCs [23] two of these 12 constructs are not

allowed: a split connector of type C_{EF} cannot be of type *XOR* or $\vee$, *i.e.*, $C_S \cap C_{EF} \cap C_{XOR} = \varnothing$ and $C_S \cap C_{EF} \cap C_\vee = \varnothing$. As a result of this restriction, there are no choices between functions sharing the same input event. A choice is resolved *after* the execution of a function, not *before*. In the formalization of EPCs, we will not impose this restriction and consider $C_S \cap C_{EF} \cap C_{XOR} = \varnothing$ and $C_S \cap C_{EF} \cap C_\vee = \varnothing$ as a guideline rather than a requirement.

5.2 On the semantics of EPCs

Definition 4 only provides syntactical requirements and does not make any statements about the behaviour of the corresponding EPC. For example, one could easily construct an EPC which contains a deadlock, e.g., in a model an XOR-split connector could split two flows which are joined using and AND-join connector thus resulting in a deadlock at runtime. In this paper, we do not consider issues related to the verification of the dynamic behaviour of EPCs. This is outside the scope of the paper and there is no general consensus on the semantics of EPCs. When we turn to configurable EPCs we will also only consider their syntactical correctness and not issues related to their dynamic behaviour. Nevertheless, we briefly discuss some related work. Since their original definition in 1992 [23], the semantics of EPCs have often been debated in literature [1,3,14,25,29,36]. In the original article [23] but also in later work of the authors only an informal definition of the semantics was given. This triggered many questions. For example, EPCs allow for OR-joins. Although, the intent of an OR-join connector can often be derived from the context, its semantics is not clear. Note that an OR-join may synchronize or not, *i.e.*, it may continue after the first input or wait for more to come. In fact, in [3] it is shown that no suitable semantics exists for the OR join. These problems

triggered two types of approaches: (1) approaches that provide formal semantics by mapping EPCs onto a language with formal semantics (*e.g.*, Petri nets) [1,25,29,36] and (2) approaches that provide a way to refine a given EPC into a model with formal semantics (e.g., by a designer selecting the desirable behaviour) [14]. In this paper, we assume only an informal semantics. In fact, we express the semantics of configurable EPCs in terms of ordinary EPCs. This makes our approach independent of the particular semantics chosen for EPCs. In other words, any of the formalization approaches mentioned [1,3,14,25,29,36] can be used as a semantical foundation.

5.3 Configurable EPCs

This section introduces the notion of a *Configurable EPC* (C-EPC). Let us return to the initial example shown in Figure 1. As was argued, the exclusive OR connector was not necessarily a decision at run-time but could also be configuration decision at build-time. For example, it could be decided at build-time that only the middle branch (invoice processing with reference) would be enabled. This would result in an EPC where the two other branches are blocked, as shown in Figure 4. The main goal of a C-EPC is to be able to specify that a concrete EPC is an acceptable configuration or not.

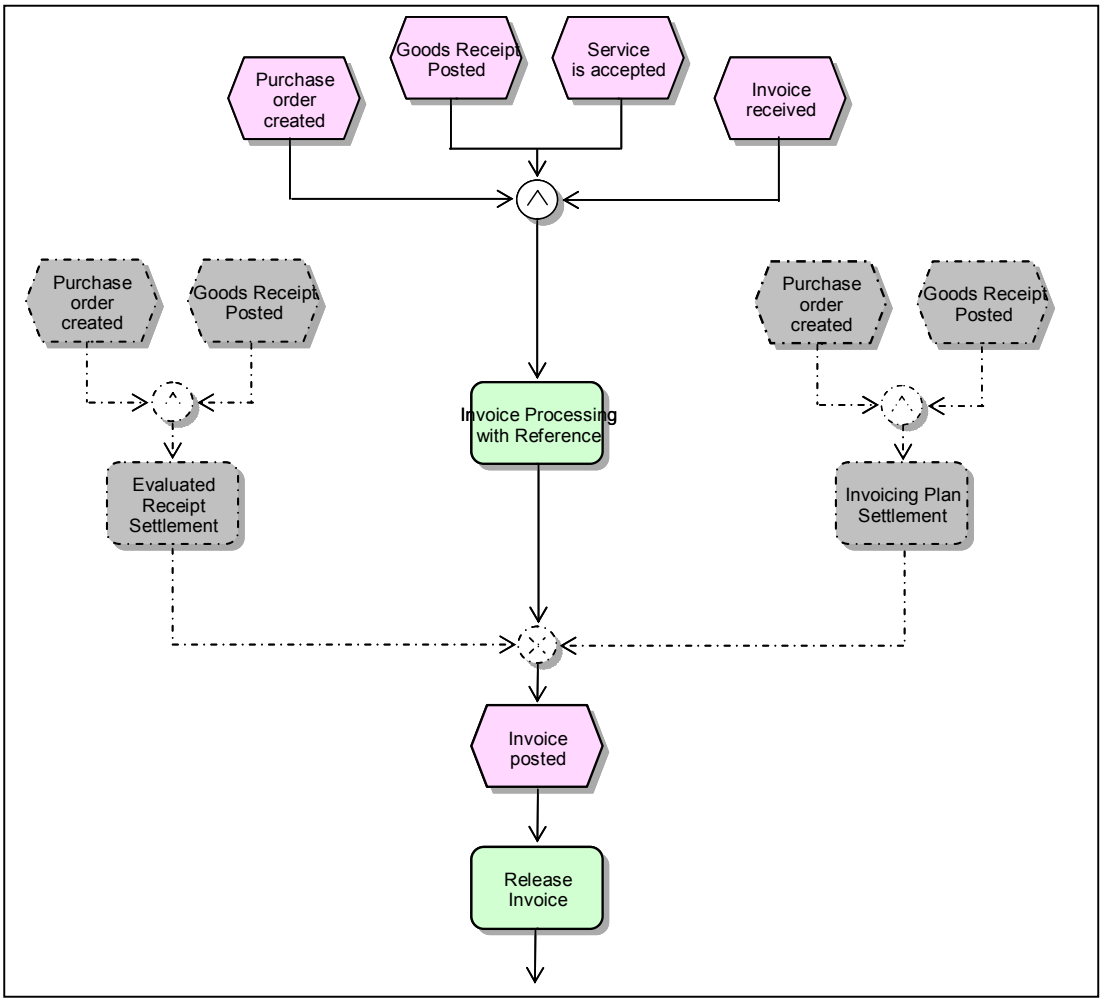

Fig. 4: Example for a model derived from the initial example (Figure 1)

In a C-EPC functions and connectors can be configurable. Configurable functions may be included (*ON*), skipped (*OFF*) or conditionally skipped (*OPT*). Configurable connectors may be restricted at build-time time, *e.g.*, a configurable connector of type $\vee$ may be mapped onto an $\wedge$ connector. Local configuration choices like skipping a function may be limited by configuration requirements. For example, if one configurable connector c of type $\vee$ is mapped onto an $\wedge$ connector, then another configurable function f needs to be included. This configuration requirement may be denoted by the logical expression $c{=}\wedge \Rightarrow f{=}ON$. To guide the configuration process there is also a partial order suggesting

the order of configuration. Moreover, besides the configuration requirements there may also be configuration guidelines. One can think of configuration requirements as hard constraints and interpret configuration guidelines as soft constraints.

Definition 5 *[Configurable EPC] A Configurable EPC (C-EPC) is a ten-tuple* $(E,F,C,l,A,F^C,C^C,O^C,R^C,G^C)$*:*

> *-* E*,* F*,* C*,* l*, and* A *are as specified in Definition 1 satisfying the constraints mentioned in Definition 4,*
> *-* $F^C \subseteq F$ *is the set of configurable functions,*
> *-* $C^C \subseteq C$ *is the set of configurable connectors,*
> *-* $O^C \subseteq (F^C \cup C^C) \times (F^C \cup C^C)$ *is a partial order over the configurable nodes suggesting the order of configuration,*
> *-* R^C *is a set of configuration requirements, and*
> *-* G^C *is a set of configuration guidelines.*

Both R^C *and* G^C *are sets of logical expressions where the atomic statements bind the configurable nodes to concrete values, e.g., "c=XOR" and "f=ON" where c is a configurable connector and f is a configurable function.*

Configurable nodes are denoted by thick circles (for configurable connectors) or thick rectangles (for configurable functions). Configuration requirements are denoted by dotted lines connecting the configurable nodes the logical expression refers to and configuration guidelines are denoted by dashed lines connecting the configurable nodes the logical expression refers to (see Figure 5). The partial order of configurable nodes O^C is not shown in the example of Figure 5.

A configurable function may be configured as included (*ON*), skipped (*OFF*) or conditionally skipped (*OPT*). Configurable connectors are mapped onto a concrete choice for the split or join considered. Clearly, a configurable connector of type $\wedge$ may not be mapped onto a concrete connector of type $\vee$. The concrete connector should always represent a behaviour allowed by the configurable connector, *i.e.*, the configuration process only restricts the possible execution sequences. In case of a configurable

connector of type *XOR* or $\vee$, also only one of the options may be selected, *e.g.*, if a split connector *c* has an output function *f*, then $c=SEQ_f$ denotes that function *f* is always selected.

In Figure 5 there are three configurable functions: *A*, *E*, and *F*. Each of these three functions can be configured as included (*ON*), skipped (*OFF*) or conditionally skipped (*OPT*). The other three functions cannot be configured, *i.e.*, are always "*ON*". There are four connectors and only the XOR connector is configurable. The configurable XOR connector can be set to XOR (*i.e.*, a choice at runtime), or select one of the two paths (*i.e.*, at configuration time the left-hand side or right-hand side is selected). Figure 5 also shows a requirement and a guideline. The requirement states that if *A* is configured as *OFF*, the path starting with event 3 should no be selected. The guideline states that if E is configured as *ON*, then F should also be configured as *ON* (and visa versa).

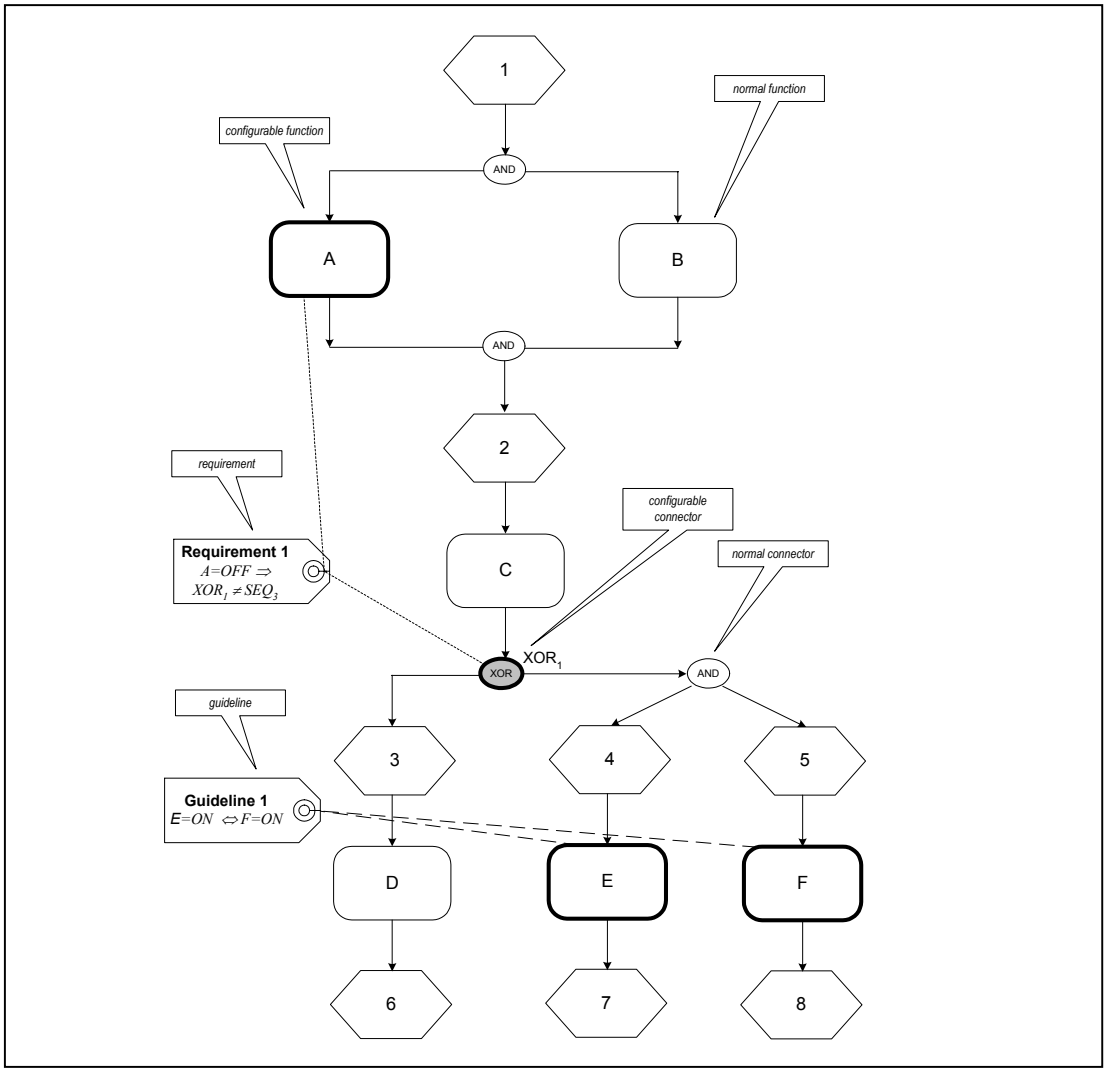

Fig. 5: Example for a C-EPC

Configurable connectors can only be configured to a connector type that restricts its behaviour. For example, a configurable OR connector may be mapped onto an AND connector, but it is not possible to map an XOR connector onto an AND connector. Table 1 illustrates the configuration rules for connectors. This table only describes the overall constraints. Each row corresponds to a configurable connector type (OR^C, XOR^C,

AND^C), *e.g.*, an OR^C may be mapped onto an OR ($\vee$), XOR, AND ($\wedge$), or SEQ (SEQ_n for some node n).

	OR	**XOR**	**AND**	**SEQ**
OR^C	X	X	X	X
XOR^C		X		X
AND^C			X	

Table 1: Constraints for the configuration of connectors

To formalize the constraints shown in Table 1 we defined a partial order $\leq^C$. This partial order is used to specify which concrete connector type may be used for a given connector type, *i.e.*, $x \leq^C y$ if and only if a connector of type y may be configured to x (*e.g.*, $\wedge \leq^C \vee$ but not $\vee \leq^C \wedge$).

Definition 6 *$[\leq^C, CT, CTS] \leq^C$ defines a partial order on $CT = \{\wedge, XOR, \vee\} \cup CTS$ where $CTS=\{ SEQ_n \mid n \in E \cup F \cup C\}$. $\leq^C = \{ (\wedge,\wedge), (XOR,XOR), (\vee,\vee), (XOR,\vee), (\wedge,\vee) \} \cup \{ (n,XOR) \mid n \in CTS\} \cup \{ (n,\vee) \mid n \in CTS\} \cup \{ (n,n) \mid n \in CTS\}$.*

Note that *$\leq^C = \{ (n,n) \mid n \in CT\} \cup (XOR,\vee) \cup (\wedge,\vee) \cup \{ (n_1,n_2) \mid n_1 \in CTS \wedge n_2 \in \{XOR,\vee\}\}$.*

Recall that this partial order is motivated by the fact that the configurable connector has to subsume the behaviour of the concrete connector.A configuration maps all configurable nodes onto concrete values like *ON*, *OFF*, and *OPT* for functions and $\wedge$, *XOR*, $\vee$, and SEQ_n for connectors.

Definition 7 *[Configuration] Let $CEPC=(E,F,C,l,A,F^C,C^C,O^C,R^C,G^C)$ be a C-EPC. $l^C \in (F^C \rightarrow \{ ON, OFF, OPT \}) \cup (C^C \rightarrow CT)$ is a configuration of CEPC if for each $c \in C^C$:*

- *$l^C(c) \leq^C l(c)$*
- *if $l^C(c) \in CTS$ and $c \in C_J$, then there exists an $n \in \bullet c$ such that $l^C(c) = SEQ_n$,*
- *if $l^C(c) \in CTS$ and $c \in C_S$, then there exists an $n \in c \bullet$ such that $l^C(c) = SEQ_n$,*

Function l^C maps configurable functions onto values like *ON*, *OFF*, and *OPT*, *i.e.*, $l^C(f) \in \{ON, OFF, OPT\}$ for $f \in F^C$. Configurable connectors are mapped onto the set CT, *i.e.*,

$l^C(c) \in CT$ for $c \in C^C$. Clearly this mapping should be consistent with Table 1 and the partial order $\preceq^C$. Moreover, if $l^C(c) = SEQ_n$, then n should be in the preset (for a join connector) or postset (for a split connector) of c.

Figure 6 shows two EPCs resulting from a configuration. Consider the EPC shown in Figure 6(a), *i.e.*, the EPC in the left hand side. If we use the configuration $\{(A,OFF),(XOR_1,SEQ_{AND3}),(E,ON),(F,ON)\}$, we obtain this EPC. Note that because function A is not needed, the AND-split and AND-join also were removed. Functions E and F are both ON thus satisfying the guideline. The requirement shown in Figure 5 is also satisfied. Since A is skipped, the configurable XOR-split XOR_1 could not be set to SEQ_3 without violating this requirement. Figure 6(b), *i.e.*, the EPC in the right hand side, results from the configuration $\{(A,ON),(XOR_1,SEQ_3),(E,OFF),(F,OFF)\}$. This configuration specifies that function *A* is always used and the configurable XOR-split is set to take only the left path involving function *D*. The setting of the two remaining configurable functions (*E* and *F*) is not relevant since they are not reachable because of the configuration of the XOR-split.

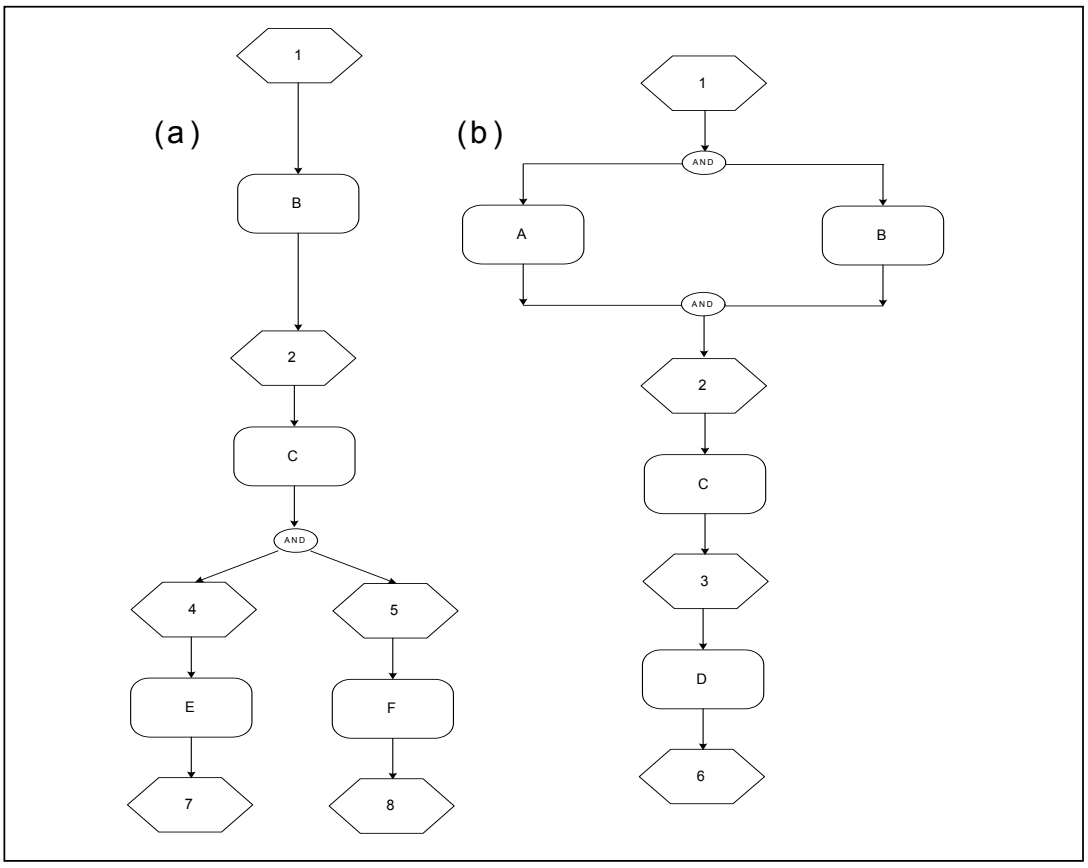

Fig. 6: Two configurations of the C-EPC shown in Figure 5

The example in Figure 7 shows that optional functions might lead to problems. The left-hand side of this figure shows a C-EPC with a configurable function *A*. The right-hand side shows possible configurations. In the left-most variant $l^C(A)=ON$ (Variant 1) and *A* is simply included. For Variant 2 $l^C(A)=OFF$ and the function is skipped and the two events are merged. In case $l^C(A)=OPT$ two variants are possible. The first one (left) simply inserts an OR-split and an OR-join connector to bypass *A*. This solution however violates the guideline/rule that an event should not be followed by an OR-split, cf. Section 5.1 and [23]. One way to solve this is to add an additional function Z and an additional events (1a) as shown in the right-most variant in Figure 7. The complication of this last construct is that configurations like $l^C(A)=OPT$ should be augmented with an

additional decision function Z. We will not enforce this but envision some post processing where fragments involving an event followed by an OR-split are refined as shown in Figure 7. We will not add this refinement to the formalizations given in this section.

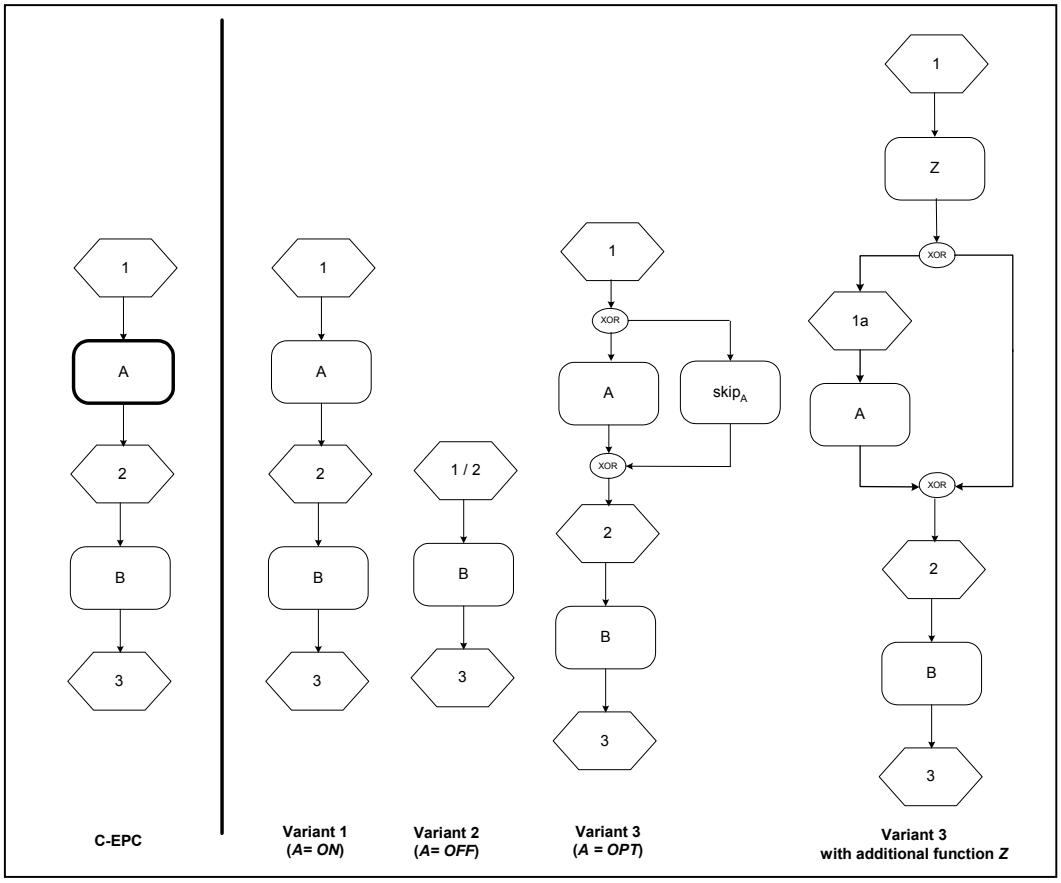

Fig. 7: Example for a configuration with additional elements

As indicated before, R^C and G^C are *sets* of logical expressions where the atomic statements bind the configurable nodes to concrete values. Configurable functions are mapped onto the set $\{ON, OFF, OPT\}$ and configurable connectors are mapped onto $\{\wedge, XOR, \vee\} \cup \{ SEQ_n \mid n \in E \cup F \cup C\}$. Examples illustrating the syntax of these atomic statements are "$c=XOR$" and "$f=ON$". These statements correspond to respectively

$l^C(c)=XOR$ and $l^C(f)=ON$ for some configurable connector c and some configurable function f. Suppose that $c_1,c_2 \in C^C$ and $f_1,f_2 \in F^C$. Examples of hard/soft constraints (*i.e.*, requirements in R^C or guidelines in G^C) are: (1) $c_1=\wedge \Leftrightarrow f_1=ON \wedge f_2=ON$, (2) $f_1=ON \vee f_2=ON$, and (3) $c_1=\wedge \Rightarrow c_2=\wedge$. Note that in Figure 5 already a requirement ($A=OFF \Rightarrow XOR_1 \neq SEQ_1$) and a guideline ($E=ON \Leftrightarrow F=ON$) have been given.

Configurations may have guidelines and/or requirements that are conflicting, *e.g.*, in Figure 5 we can add the following two requirements $A=OFF \Leftrightarrow E=ON$ and $A=OFF \Leftrightarrow F=OFF$. Clearly there requirements are conflicting with the original guideline. If there are no conflicting requirements the model is valid. If, in addition, the guidelines are not conflicting, the configuration is suitable.

Definition 8 *[Valid/suitable configuration] Let CEPC=(E,F,C,l,A,F^C,C^C, O^C,R^C,G^C) be a C-EPC and l^C a configuration of CEPC. l^C is a valid configuration if it satisfies all configuration requirements, i.e., it satisfies all logical expressions in R^C. l^C is a suitable configuration if it is valid and it satisfies all configuration guidelines, i.e., it satisfies all logical expressions in R^C and G^C.*

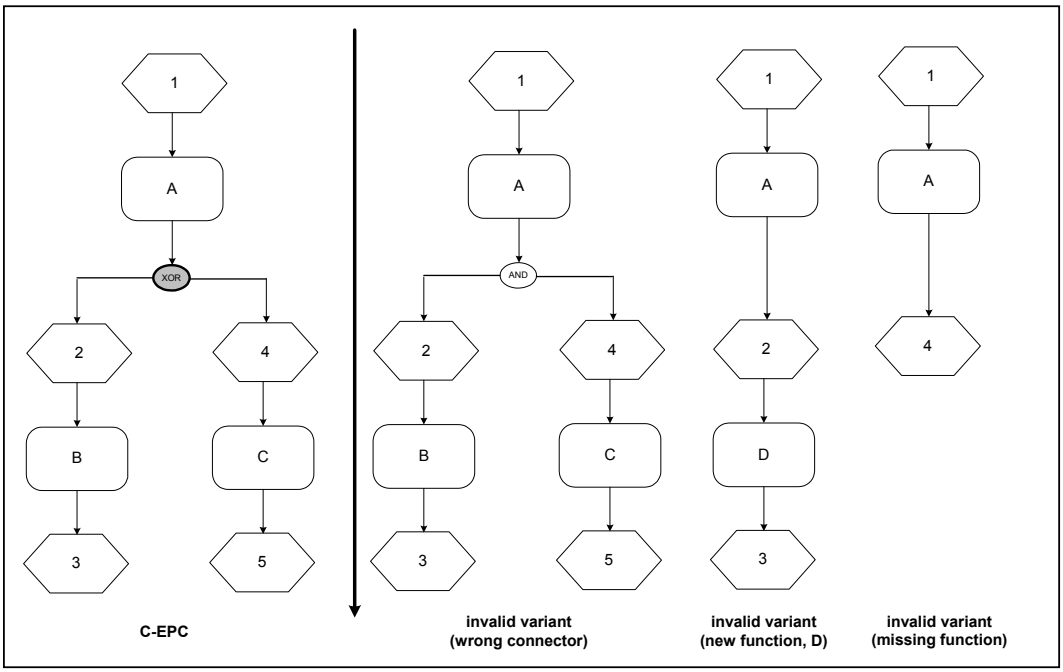

Fig. 8: Examples for invalid configurations

A configuration is valid if it satisfies all requirements. Figure 8 shows some examples for invalid configurations. Since we want to avoid C-EPCs that have no valid configurations, we introduce the notion of a "satisfiable" C-EPC.

Definition 9 *[Satisfiable] Let CEPC=(E,F,C,l,A,F^C,C^C,O^C,R^C,G^C) be a C-EPC. CEPC is satisfiable if and only if there is valid configuration.*

Give the fact that all requirements and guidelines are logical expressions it is fairly easy to provide tool support to guide the designer towards a valid configuration.

5.4 Semantics of configurations

In examples we already showed that a configuration corresponds to a concrete EPC. Now we provide an algorithm to construct an EPC based on a C-EPC and a configuration. Note that a C-EPC defines a space of concrete EPCs. Each valid configuration maps a C-EPC onto a concrete EPC. The function β maps a C-EPC and its configuration onto a concrete EPC β(*CEPC*,l^C).

Definition 10 *[β] Let CEPC=(E,F,C,l,A,F^C,C^C,O^C,R^C,G^C) be a C-EPC and l^C a configuration of CEPC. The corresponding EPC β(CEPC,l^C) is constructed as follows:*

1. *EPC_1=(E,F,C,l_1,A_1) with l_1 = {(c,l(c)) / c ∈ C\C^C} ∪{(c,l^C(c)) / c ∈ C^C} and A_1 = A \({(c,n) ∈ C_S × c• / ∃$_{n' ∈ c•}$ l^C(c)=$SEQ_{n'}$ ∧n ≠n'} ∪{(n,c) ∈ •c ×C_J / ∃$_{n' ∈ •c}$ l^C(c)=$SEQ_{n'}$ ∧n ≠n'}) is the EPC obtained by mapping the configurable connectors onto their concrete type and removing arcs not involving the selected sequence.*[3]

2. *For each f ∈ F^C such that l^C(f) = OFF, rename the function to $skip_f$ to reflect that the corresponding function is not executed. If •f ∪f• ⊆ E, then merge input and output event into one, i.e., EPC_2=(E_2,F_2,C,l_1,A_2) with E_2 = (E ∪{e}) \(•f ∪f•), F_2 = F\{f}, and A_2 = { (n_1,n_2) ∈ A / {n_1,n_2}∩(•f ∪f•)=∅} ∪{ (n_1,e) / (e_1 ∈ •f) ∧(n_1,e_1) ∈ A} ∪{ (e,n_2) / (e_2 ∈f•) ∧(e_2,n_2) ∈ A} where e is the new connector (no name clashes, i.e., e ∉ N) merging the old input and output connector. Repeat this for each f of this type and let EPC_2 be the resulting EPC.*[4]

3. *For each f ∈ F^C such that l^C(f) = OPT, add function $skip_f$, a split connector $split_f$, and a join connector $join_f$ making f optional, i.e., EPC_3=(E_2,F_3,C_3,l_3,A_3) with F_3 =F_2∪{$skip_f$}, C_3 = C∪{$split_f$,$join_f$}, l_3=l_1∪{ ($split_f$,XOR),($join_f$,XOR) }, A_3 = { (n_1,n_2) ∈ A_2 / f ∉ {n_1,n_2}} ∪{($split_f$,f),($split_f$,$skip_f$),($skip_f$,$join_f$),(f,$join_f$)} ∪{ (n,$split_f$) / (n,f) ∈ A_2} ∪{ ($join_f$,n) / (f,n) ∈ A_2}. Repeat this for each f of this type and let EPC_3 be the resulting EPC.*

---

[3] Note that such an EPC may not satisfy all the requirements stated in Definition 4.

[4] Note that it is not always possible to remove functions that are connected to a connector since connectors are either on a path from an event to a function or vice versa.

4. *Remove all connectors with just one input and one output node, i.e., $EPC_4=(E_2,F_3,C_4,l_4,A_4)$ with $C_4 = \{c \in C_3 \mid |c\bullet| > 1 \vee |\bullet c| > 1\}$, $l_4 = \{(c,x) \in l_3 \mid c \in C_4\}$, and $A_4 = \{ (n_1,n_2) \in A_3 \mid \{n_1,n_2\} \cap (C_3 \setminus C_4) = \emptyset\} \cup \{(n_1,n_2) \mid \exists_{c \in C3 \setminus C4} \{(n_1,c),(c,n_2)\} \in A_3\}$.*

5. *Remove all isolated nodes, i.e., nodes without input and output arcs.*

6. *Re-apply Step 2 of the algorithm, i.e., try to remove the remaining functions labelled "$skip_f$".*

7. *Remove all nodes not on some path from a start event to a final event. Consider only start and final events also present in original EPC, i.e., not the new start/final events that may have been introduced in e.g. Step 1.*

8. *Re-apply Step 4 of the algorithm, i.e., remove connectors with just one input and one output node that may have been introduced in Step 7. The resulting EPC is $\beta(CEPC,l^C)$.*

It is easy to verify that the examples given thus far are indeed consistent with the algorithm. Although Definition 10 suggests that $\beta(CEPC,l^C)$ is indeed an EPC satisfying the requirements mentioned before, this remains to be proven.

Theorem 1 *[$\beta(CEPC,l^C)$ is an EPC] Let $CEPC=(E,F,C,l,A,F^C,C^C,O^C,R^C,G^C)$ be a C-EPC and l^C a configuration of CEPC. $\beta(CEPC,l^C)$ is an EPC satisfying all requirements stated in Definition 4.*

Proof.

$EPC_0=(E,F,C,l,A)$ satisfies all requirements by definition. Next we check how the requirements are affected by the seven steps.

- The sets E, F, and C are pair-wise disjoint. Although not always stated explicitly we assume no name clashes.
- For each $e \in E$: $|\bullet e| \leq 1$ and $|e \bullet| \leq 1$. Cardinality of number of input and output nodes for events is not changed. Step 2 may merge two events but does not jeopardize this requirement. All other steps can only reduce the number of inputs/outputs.
- There is at least one event $e \in E$ such that $|\bullet e| = 0$ (*i.e.* a start event). Start events are not removed.
- There is at least one event $e \in E$ such that $|e \bullet| = 0$ (*i.e.* a final event). Final events remain final events.
- For each $f \in F$: $|\bullet f| = 1$ and $|f \bullet| = 1$. Functions may be removed but the cardinality of number of input and output nodes for functions is not changed.
- For each $c \in C$: $|\bullet c| \geq 1$ and $|c \bullet| \geq 1$. Existing connectors and newly added connectors ($split_f$, $join_f$) satisfy this requirement.
- C_J and C_S partition C. This guaranteed by Step 4.
- C_{EF} and C_{FE} partition C. The nature of connectors is never changed.

We will now use the initial example shown in Figure 1 to summarize the recommendations for a configurable reference modelling language. The example is a condensed version of the reference model for invoice verification as it can be found in the Enterprise System SAP R/3 Ver. 4.6c. Figure 1 showed the current non-configurable reference model, Figure 4 provided an example for one model, which can be derived from this reference model.

The model depicted in Figure 1 can be perceived as a 'Max-EPC' as it includes all possible ways of invoice verification supported by the SAP system. A more detailed analysis, however, shows that this model includes many optional elements. The core of this process is the classical invoice processing with reference to a purchase order, a delivery note or service entry sheet and the actual invoice. This process is mandatory and

all elements have to be configured. Evaluated receipt settlement (ERS) is an option that allows bypassing the entire classical invoice verification process. Based on long term contracts and a clear specification of the goods, invoices are posted and released based on the arrival of goods which conform in quantity and quality to the specifications of the purchase order or contract. Thus, ERS is typically only a relevant option, if the company is of significant size and the business relationship is based on a highly repetitive purchasing process based on a long-term contract with a clear specification of the payment details. In a similar way, invoice plan settlement is an optional function. In this case, invoices are consolidated in an invoice plan and scheduled over a series of future dates independently of individual procurement transactions and the actual receipt of goods and services. This is relevant for regularly recurring procurement transactions (*e.g.* car leasing, subscriptions) (so called periodic invoicing plan) and transactions that are subject to stage payments (*e.g.* a building project) (so called partial invoicing plan). Invoicing plan settlement facilitates the automatic creation and payment of invoices and uses functionality of the evaluated receipt settlement solution [37]. Figure 9 shows the reference model in C-EPC notation that can be derived from this description.

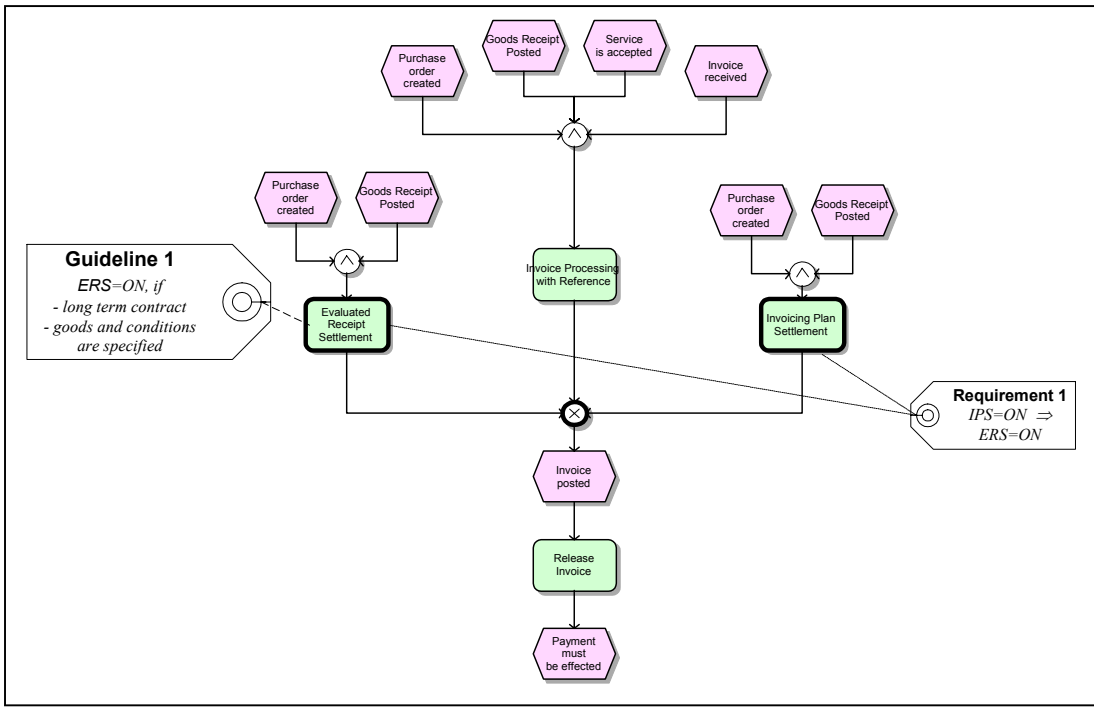

Fig. 9: The example from Figure 1 in C-EPC notation

5.5 Partially configured C-EPCs

Definition 7 assumes a complete configuration, *i.e.*, l^C is a complete function mapping each configurable node onto a concrete value. However, the configuration process may go through several stages and therefore we also add the notion of a partial configuration.

Definition 11 *[Partial configuration] Let* $CEPC=(E,F,C,l,A,F^C,C^C,O^C,R^C,G^C)$ *be a C-EPC.* $l^C \in (F^C \nrightarrow \{ON, OFF, OPT\}) \cup (C^C \nrightarrow CT)$[5] *is a partial configuration of CEPC if for each* $c \in C^C \cap dom(l^C)$*:*

- $l^C(c) \leq^C l(c)$
- *if* $l^C(c) \in CTS$ *and* $c \in C_J$*, then there exists an* $n \in \bullet c$ *such that* $l^C(c) = SEQ_n$,
- *if* $l^C(c) \in CTS$ *and* $c \in C_S$*, then there exists an* $n \in c \bullet$ *such that* $l^C(c) = SEQ_n$,

[5] Note that partial functions are denoted by $A \nrightarrow B$, *i.e.*, a $f \in A \nrightarrow$ B is a function with a domain that is a subset of A.

One can think of a C-EPC with a partial configuration as another C-EPC. Using an algorithm similar to the one described in Definition 10, one can transform C-EPC with a partial configuration into a new C-EPC. We omit details, but it is straightforward to realize this using Definition 10. Simply consider the configurable nodes that are not configured as unconfigurable nodes when applying the algorithm. Let β′ be the modified algorithm which transforms a C-EPC with a partial configuration into a new C-EPC.

Without proof we give the following theorem.

Theorem 2 *[$\beta(CEPC,l^C)$ is an EPC] Let $CEPC_1$ be a C-EPC and l^C a partial configuration of CEPC. $CEPC_2=\beta'(CEPC_1,l^C)$ is the corresponding C-EPC.*

- *If $CEPC_2$ is satisfiable, then $CEPC_1$ is also satisfiable.*
- *If $l^C{}_2$ is a valid (suitable) configuration of $CEPC_2$, then $l^C{}_2$ is also a valid (suitable) configuration of $CEPC_1$*

The above allows us to indicate whether a partial configuration of a C-EPC is satisfiable.

The concept of partial configured C-EPC opens up interesting possibilities. Consider for example a configurable Enterprise System like SAP. There could be a top-level C-EPC which indicates all possible configurations of SAP with respect to a given process. This C-EPC could be partially configured per industry. (Recall that SAP has 27 alternative industry solutions, as indicated in Section 3.) In other words, for each industry there are partial configured C-EPCs. Such partial configured C-EPCs can be used as a starting point within a given organization. For large organizations there may be different versions of the same process, *e.g.*, per country or per region. However, at the same time the organization may want to enforce some unification. Therefore, the industry specific C-EPC may be partially configured into an organization-specific C-EPC. The latter C-EPC may be configured within specific parts of the organization (*e.g.*, per region). This

example shows that it may be worthwhile to have (partially configured) C-EPCs at different levels where at each level the lower level is a (partial) configuration of the upper level. For example, there may be a C-EPC at the level of SAP (What can the system do?), at the level of an industry (What configurations of SAP make sense for the automotive industry?), and at the level of one organization (What configurations do we allow within our organization?). Only the C-EPC at the organizational level is configured completely to support a concrete process within some part of the organization (How do we do this process within the Eindhoven branch of our organization?).

Apart from configuration at various levels there can always be the need for customization (*i.e.*, support processes that do not fit into the C-EPC). The latter should be avoided (if possible) since it is risky and costly. If customization is unavoidable, it may be interesting to use the notions of inheritance described in [2,5]. These notions of inheritance can easily be applied to EPCs and C-EPCs. The topic of customization is however out of the scope of this paper.

5.5 Extensions

To conclude this section we reflect on the requirements given in Section 4 in the context of the C-EPC language just defined.

a) The C-EPC language defined in this section mainly focuses on the process and control-flow aspects. The data aspect and function aspect have not been addressed explicitly. Note that functions can be configured but this only refers to their presence rather than the functionality of these functions.

b) C-EPCs do not distinguish between mandatory and optional decisions. However, it is fairly easy to add this functionality. It could be defined as an extension of the

partial order O^C. It is also possible to extend the language with defaults for optional configuration decisions.

c) C-EPCs do not differentiate between global and local decisions. Again it is fairly easy to add this as an attribute to all configurable nodes. However, the real challenge is to get this information.

d) Similarly remarks hold for the difference between critical and non-critical decisions.

e) Configuration decisions can have interrelationships. This is partly covered by the requirements (R^C) and guidelines (G^C) in a C-EPC. However, these are restricted to interrelationships within one model and not for *e.g.* interrelationships between two process models and interrelationships between a reference process model and a related reference data model.

f) Configuration decisions can be made on different levels. This can be supported by the partially configurable C-EPCs as discussed in the previous subsection.

g) In a C-EPC variation points do not refer to further related information within the Enterprise System. However, this can be added easily.

h) The entire configuration process should also be guided by recommendations or configuration guidelines. This is supported by the guidelines (G^C) and the partial order O^C.

i) The last requirement refers to the impact of configuration extensions on the perceived model complexity. The C-EPC is a natural extension of the standard EPC and should not cause any problems for the typical user of a reference model. The most complex parts are the interrelationships defined in the requirements (R^C)

and guidelines (G^C) in a C-EPC since these are expressed in logical expressions. It may be worthwhile to think of more graphical notations for modelling typical requirements like for example dependency constraints.

As indicated the C-EPC language defined in this paper covers many of the requirements but not all. The language reported in this paper focuses on the core functionality of a configurable reference modelling language based on EPCs.

6 Related Work

This area of research can be divided into requirements engineering for the *development* of Enterprise Systems [10,12] and requirements engineering for the *configuration of Enterprise Systems*. The latter one is the focus of this paper. Academic contributions in this field are still the exception.

Soffer et al.'s [41] suggestions on ERP modelling can be regarded as the closest to our proposed ideas. The goal of their paper is to determine what language is most appropriate for representing ERP system capabilities. Following the concept of scenario-based requirements engineering, Soffer et al. evaluate the Object-Process Modelling Methodology. The so-called argumentation facet, related to the ability of a modelling language to express optionality-related information, is just one of many of their criteria. The paper does not comprehensively analyse requirements related to modelling Enterprise Systems configurability and evaluates an existing technique rather than developing a new and more appropriate technique.

A number of papers have contributed to the area of goal modelling and the interrelationships of goal models to other model types and requirement specifications. As an example, Rolland and Prakesh [31] suggest a map including ERP goals and objectives for the identification and evaluation of user needs. In [30], Rolland links goals and strategies in so called maps. She applies this idea to processes from SAP's Material Management (MM) solution. This interesting work stays on the rather high level of goals and strategies, does not utilise techniques and contents of existing reference models and is very brief regarding configurability of goals and related processes.

Further work on goal modelling has been conducted by Soffer and Wand [42], who propose a generic theory-based process modelling framework based on Bunge's ontology for a goal-driven analysis of process model validity. Giorgini et al. [19] present a formal framework for reasoning with goal models. All these papers do not consider the configurability of models and do not study modelling techniques which are widely utilised in practice.

Gulla and Brasethvik [20] introduce three process modelling tiers to manage the complexity of process modelling in comprehensive ERP Systems projects. Their functional tier dimension deals with the functionality of the Enterprise System. However, they do not study how reference models fit into in this tier. Brehm et al. [9] discuss alternative ways of configuring Enterprise Systems. Their taxonomy for ERP configuration and customising is widely cited. However, they do not demonstrate how this work can be linked to reference models of Enterprise Systems.

Related work has also been conducted in the area of variability management in software families. Halmans and Pohl [21] discuss issues related to the communication of the variability of a software-product family. They propose an extension to use case diagrams based on cardinalities in order to explicitly depict variation points. They do not support dependencies between variation points. Moreover, use case diagrams have not widely been used for reference models. Halmans and Pohl [21] have been influenced by previous work on representing variability in use case diagrams by Bertolino et al. [8], von der Massen and Lichter [27], and John and Mutig [22]. Software product families have also been investigated in from an architectural viewpoint. In fact, there have been several workshops on software architectures for product families, cf. [26]. As an example consider the work of Dolan et al. [15] on the role of the various stakeholders when it comes to software product families. All these publications do not relate to large Enterprise Systems solutions and process modelling techniques which are used in this context.

7 Conclusion and Outlook

Reference models have been defined in this paper as reusable conceptual models that depict recommended structures and processes. One main class of reference models are application reference models that document the functionality of off-the-shelf-solutions. Reference modelling languages face specific requirements regarding the configuration of these models. However, current models such as the SAP reference models (and other Enterprise Systems reference models) are designed using modelling languages that do not cater for the needs of configuration. Thus, only limited opportunities exist to specify valid configurations. This paper proposed a new dedicated reference modelling language that

allows exactly this explicit specification of configurations in reference process models. This language has been called Configurable EPCs and has been derived from the popular EPCs.

The current focus of our research is on developing a list of configuration patterns and exploring alternative ways of modelling these patterns. The quality of our proposed reference modelling language as well as its notations will be tested in experiments and focus groups. This project is funded by SAP Corporate Research and it is the explicit aim to develop an applicable language. As part of this research project, a related SAP-funded empirical study on the actual modelling practice in Australia is currently conducted. This study will give important insights into the problems with the existing reference models. Furthermore, it is planned to extend this work to configurable collaborative business scenario diagrams.

Another interesting question is: "Given a C-EPC and a partial configuration, is the partial configuration satisfiable?". Related questions are: "If not satisfiable, why not?" or "If satisfiable, which configurations are still possible?". Since the number of configurations is finite, it is easy to provide automated support for addressing these questions. Moreover, it would be interesting to link these questions to the dynamics of the resulting EPCs. It may be the case that a partial configuration satisfiable in terms of the configuration requirements but that the resulting EPCs will always deadlock.

A further area of research will be the inclusion of evidence-based research. This could include access to relevant benchmarking information or typical configuration decisions made in one industry sector. This could be visualised in the reference models using the proposed configuration guidelines and would provide valuable guidance for the required

decisions. Clearly this is also linked to the topic of process mining (cf. www.processmining.org), *i.e.*, extracting knowledge from event logs. By analysing the logs of SAP and other Enterprise Systems one can link certain performance metrics to configuration decisions.

8 References

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

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

[3] W.M.P. van der Aalst, J. Desel, 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, Trier, Germany, November 2002. Gesellschaft für Informatik, Bonn, 71-80.

[4] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Workflow Patterns, Distributed and Parallel Databases 14(3) (2003) 5-51.

[5] T. Basten, W.M.P. van der Aalst, Inheritance of Behavior. Journal of Logic and Algebraic Programming, 47(2) (2001) 47-145.

[6] J. Becker, M. Kugeler, M. Rosemann, eds., Process Management, Berlin et al., 2003.

[7] P. Bernus, GERAM: Generalised Enterprise Reference Architecture and Methodology, version 1.6.3, March 1999.

[8] A. Bertolino, A. Mantechi, S. Gnesi, G. Lamir, A. Maccari, Use Case Description of Requirements for Product Lines. Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 - REPL '02. Technical Report: ALR-2002-033, AVAYA labs. 2002.

[9] L. Brehm, A. Heinzl, M.L. Markus, 2000, Tailoring ERP Systems: A Spectrum of Choices and their Implications, Proceedings of the 34th Hawaii International Conference on System Sciences. Maui, Hawaii, 3-6 January 2000.

[10] S. Brinkkemper, Requirements Engineering for ERP: Requirements Management for the Development of Packaged Software, Proceedings of the 4th International Symposium on Requirements Engineering. Limerick, Ireland, 7-11 June 1999.

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

[12] M. Daneva, Practical Reuse Measurement in ERP Requirements Engineering, Proceedings of the 12th International Conference CAiSE 2000, eds., B. Wangler and L. Bergman. Stockholm, Sweden, June 5-9, Lecture Notes in Computer Science 1789, 2000, 309-324.

[14] J. Dehnert, P. Rittgen, Relaxed Soundness of Business Processes, In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE 2001), volume 2068 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2001, 157-170.

[15] T. Dolan, R. Weterings, J.C. Wortmann, Stakeholder-Centric Assessment of Product Family Architecture: Practical Guidelines for Information System Interoperability and Extensibility, In International Workshop on Software Architectures for Product Families (IW-SAPF-3), volume 1951 of Lecture Notes in Computer Science, Springer Verlag, 2000, 225-245.

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

[17] T. Forsberg, G. Roenne, J. Vikstroem, Process Modeling in ERP Projects – a discussion of potential benefits. Intentia R&D 2002.

[18] U. Frank, Conceptual Modelling as the Core of the Information Systems Discipline - Perspectives and Epistemological Challenges, Proceedings of the America Conference on Information Systems – AMCIS '99, Milwaukee, 1999, 695-698.

[19] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sabastiani: Formal Reasoning Techniques for Goal Models. Journal on Data Semantics, 2004, 1(1), 1-20.

[20] J. A. Gulla, T. Brasethvik, On the Challenges of Business Modeling in Large Scale Reengineering Projects,. Proceedings of the 4th International Conference on Requirements Engineering, Schaumburg, Ill., 19-23 June 2000, 17-26.

[21] G. Halmans, K. Pohl, Communicating the variability of a software-product family to customers. Software and Systems Modeling, 2(1) 2003 15-36.

[22] I. John, D. Mutig, Tailoring Use Cases for Product Line Modeling. Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 - REPL '02. Technical Report: ALR-2002-033, AVAYA labs 2002.

[23] G. Keller, M. Nüttgens, A.W. Scheer, Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK), Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[24] H. Klaus, M. Rosemann, G.G. Gable, What is ERP?, Information System Frontiers (2000) 2 (2) 141-162.

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

[26] F. van der Linden, eds., International Workshop on Software Architectures for Product Families, volume 1951 of Lecture Notes in Computer Science, Berlin 2000.

[27] T. von der Massen, H. Lichter, Modeling Variability by UML Use Case Diagrams. Proceedings of the International Workshop on Requirements Engineering for Product Lines 2002 - REPL '02. Technical Report: ALR-2002-033, AVAYA labs. 2002.

[28] V.B. Misic, J.L. Zhao, Evaluating the Quality of Reference Models, in A.H.F. Laender, S.W. Liddle, V.C.Storey, eds., in Proceedings of the 19th International

Conference on Conceptual Modeling. Salt Lake CIt, Utah, USA, 9-12 October 2000. Lecture Notes in Computer Science 1920, 484-498.

[29] P. Rittgen, Modified EPCs and their Formal Semantics. Technical Report 99/19, University of Koblenz-Landau, Koblenz, Germany, 1999.

[30] C. Rolland: Aligning Business and System Functionality Models. Proceedings of the 3rd Pre-ICIS Workshop on Process Management and Information Systems. Eds.: J. Akoka, I. Comyn-Wattiau and M. Favier. Washington D.C., 12 December 2004, 1-10.

[31] C. Rolland, N. Prakash, Bridging the Gap between Organisational Needs and ERP Functionality, Requirements Engineering, 5 (3) (2000) 180-193.

[32] M. Rosemann, ERP software: characteristics and consequences, in J. Pries-Heje et al., eds., in Proceedings of the 7th European Conference on Information Systems, 23-25 June 1999, Copenhagen.

[33] M. Rosemann, Using Reference Models within the Enterprise Resource Planning Lifecycle. Australian Accounting Review, 10(3) 2000 19-30.

[34] M. Rosemann, Application Reference Models and Building Blocks for Management and Control (ERP Systems). in: Handbook on Enterprise Architecture. P. Bernus, L. Nemes, G. Schmidt, eds., Springer-Verlag: Berlin et al., 2003, 595-616.

[35] M. Rosemann, G. Shanks, Extension and Configuration of Reference Models for Enterprise Resource Planning Systems. Proceedings of the 12th Australasian Conference on Information Systems - ACIS 2001, G. Finnie, D. Cecez-Kecmanovic, B. Lo, eds., Coffs Harbour, 4-7 December 2001, 537-546.

[36] F. Rump, Geschäftsprozessmanagement auf der Basis Ereignisgesteuerter Prozessketten. Reihe Wirtschaftsinformatik, Teubner Verlag, Germany, 1999.

[37] Online Documentation mySAP ERP, SAP AG, (help.sap.com)., 2005.

[38] A.-W. Scheer, ARIS – Business Process Modelling. 3rd ed., Berlin et al., 2000.

[39] L. Silverston, The Data Model Resource Book, Volume 1, A Library of Universal Data Models for All Enterprises, revised edition, 2001.

[40] L. Silverston, The Data Model Resource Book, Volume 2, A Library of Data Models for Specific Industries, revised edition, 2001.

[41] P. Soffer, B. Golany, D. Dori: ERP modelling – a comprehensive approach. Information Systems, 28 (2003), 673-690.

[42] P. Soffer, Y. Wand: Goal-Driven Analysis of Process Model Validity. In: Advanced Information Systems Engineering. Proceedings of the 16th International Conference (CAiSE 2004). Eds.: A. Persson, J. Stirna. Riga, Latvia, June 2004, 521-535.

[43] M. Verbeek, On Tools & Models, in: Dynamic Enterprise Innovation. Establishing Continuous Improvement in Business. 3rd ed., R. van Es, ed., 1998.