

Formalization and verification of event-driven process chains

W.M.P. van der Aalst*

Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, NL 5600 MB, Eindhoven, Netherlands

Received 5 March 1998; received in revised form 1 March 1999; accepted 4 March 1999

Abstract

For many companies, business processes have become the focal point of attention. As a result, many tools have been developed for business process engineering and the actual deployment of business processes. Typical examples of these tools are Business Process Reengineering (BPR) tools, Enterprise Resource Planning (ERP) systems, and Workflow Management (WFM) systems. Some of the leading products, e.g. SAP R/3 (ERP/WFM) and ARIS (BPR), use Event-driven Process Chains (EPCs) to model business processes. Although the EPCs have become a widespread process modeling technique, they suffer from a serious drawback: neither the syntax nor the semantics of an EPC are well defined. In this paper, this problem is tackled by mapping EPCs (without connectors of type \vee) onto Petri nets. The Petri nets have formal semantics and provide an abundance of analysis techniques. As a result, the approach presented in this paper gives formal semantics to EPCs. Moreover, many analysis techniques are available for EPCs. To illustrate the approach, it is shown that the correctness of an EPC can be checked in polynomial time by using Petri-net-based analysis techniques. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Event-driven process chains; Petri nets; Workflow management; Verification

1. Introduction

As a response to increased competitive pressure in the global marketplace, enterprises are looking to improve the way they are running their businesses. The term business process engineering [1] subsumes the set of principles, activities, and tools to support improvements of business processes. At the moment many software tools are available to support and enable business process engineering efforts. Typical examples of these tools are:

- *Business Process Reengineering (BPR) tools*
A BPR tool can be used to model and analyze business processes. The visual representation of processes and the ability to evaluate alternatives support the (re)engineering of business processes.
- *Enterprise Resource Planning (ERP) systems*
ERP systems such as SAP R/3, BAAN, PeopleSoft, and Oracle automate manufacturing processes, organize accountants' books, and streamline corporate departments like human resources. An explicit representation of the business processes is used as a starting point for configuring these systems.
- *Workflow Management (WFM) systems*
A WFM system is a generic software tool, which allows

for the definition, execution, registration and control of workflows (cf. Ref. [2,3]). In essence, the WFM system is a generic building block to support business processes.

BPR tools support the 're-thinking' of business processes, ERP and WFM systems are the software applications that make these reengineered processes possible. Each of these tools requires an explicit representation of the business processes at hand. Most of the business process modeling techniques that are used, are vendor specific, i.e. they are supported by just one tool. Only a few tools use a generic technique such as Petri nets, SADT, IDEF, or EPC.

In this paper, we focus on the process modeling technique used by some of the leading tools in the field of business process engineering. Process models made with this technique are called Event-driven Process Chains (EPCs), (cf. Ref. [4]). EPCs are used in tools such as SAP R/3 (SAP AG), ARIS (IDS Prof. Scheer GmbH), LiveModel/Analyst (Intellicorp Inc.), and Visio (Visio Corp.). SAP R/3 [5,6] is the leading ERP-system and is used in more than 7500 companies in 85 countries. ARIS [7] leads BPR tool market (7000 licenses). LiveModel/Analyst and Visio are also examples of BPR tools based on EPCs.

EPCs have become a widespread process modeling technique, because of the success of products such as SAP R/3 and ARIS. Unfortunately, neither the syntax nor the semantics of an EPC are well defined. As a result, an EPC may be ambiguous. Moreover, it is not possible to check the model

* Tel.: + 31-40-2474-4295; fax: + 31-40-246-3992.

E-mail address: wsinwa@win.tue.nl (W.M.P. van der Aalst)

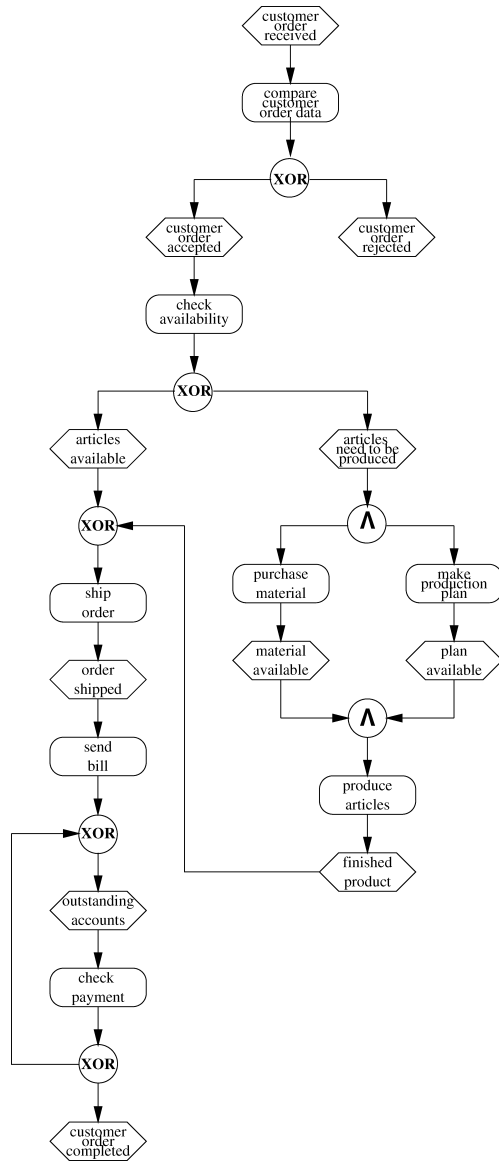


Fig. 1. Modeling of a business process, using EPCs.

for consistency and completeness. These problems are serious, because EPCs are used as the specification of business processes and need to be processed by the ERP and WFM systems. The absence of formal semantics also hinders the exchange of models between tools of different vendors and prevents the use of powerful analytical techniques.

In this paper, these problems are tackled by an approach based on *Petri nets* (cf. [8]). The Petri net formalism is a popular and powerful process modeling technique for the representation of processes which exhibit concurrency, parallelism, synchronization, non-determinism, and mutual exclusion. The building blocks used in an EPC (events, functions and connectors) are close to the building blocks used in a Petri net (places and transitions). In fact, EPCs correspond to a subclass of Petri nets. We will show that it is possible to map an EPC onto a Petri net. This way, we can

use the Petri net formalism to give formal semantics to EPCs. Moreover, we can use advanced Petri-net-based techniques to analyze EPCs. To show the potential of this approach, we present a technique to verify the correctness of an EPC. For this purpose, we use the so-called *soundness property* introduced in Ref. [9]. An EPC is sound if, for any case, the process terminates properly, i.e. termination is guaranteed, there are no dangling references, and deadlock and livelock are absent. We will show that for EPCs the soundness property can be checked in polynomial time. Throughout this paper, we consider EPCs without connectors of type \vee (i.e. OR connectors). The semantics of a join connector of type \vee is not clear and subject to multiple interpretations. Therefore, we discuss the problems associated to such a connector and review the possible solutions.

This paper builds on the results presented in Refs. [9,10]. The application of Petri nets to workflow modeling is also described in Refs. [11–15]. In Germany, several research groups have been working on the formalization of EPCs [16–21]. Researchers of both the University of Hamburg [17,18] and the University of Saarland [16] have investigated the relation between the EPCs and the Petri nets. There are several differences between these approaches and the approach described in this paper. First of all, in this paper, the translation to Petri nets is formalized. Secondly, our approach is based on the classical Petri net instead of a high-level variant. Finally, we provide tools and techniques to check the soundness property in polynomial time.

2. Event-driven process chains

EPCs are an intuitive graphical business process description language introduced by Keller, Nüttgens and Scheer in 1992 [4]. The language is targeted to describe processes on the level of their business logic, not necessarily on the formal specification level, and to be easy to understand and use by business people. The methodology inherited the name from the diagram type shown in Fig. 1. Such a diagram shows the control flow structure of the *process* as a *chain of events* and functions, i.e. an event-driven process chain.

An EPC consists of the following elements:

- *Functions*
The basic building blocks are functions. A function corresponds to an activity (task, process step), which needs to be executed.
- *Events*
Events describe the situation before and/or after a function is executed. Functions are linked by events. An event may correspond to the post-condition of one function and act as a precondition of another function.
- *Logical connectors*
Connectors can be used to connect activities and events. This way the flow of control is specified. There are three

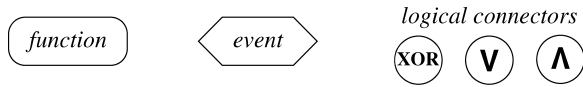


Fig. 2. The building blocks of an EPC.

types of connectors: \wedge (and), XOR (exclusive or) and \vee (or).

These building blocks are shown in Fig. 2.

The process modeled in Fig. 1 models the processing of a customer order. The process starts with the event *customer order received*. First the customer order data is checked, and as a result of this function the order is either rejected or accepted. The XOR connector models the fact that after the execution of the function *compare customer order data* one of the two events (*customer order accepted* or *customer order rejected*) holds. If the order is rejected, the process stops. For each accepted order, the availability is checked. If the articles are not available, then two functions are executed in parallel: *purchase material* and *make production plan*. After executing both functions, the articles are produced. If either the event *articles available* or the event *finished product* holds, then the function *ship order* can be executed. After the order has been shipped, the bill is sent. After sending the bill, it is checked whether the bill has been paid (function *check payment*). If the check has a positive result, the process is completed (event *customer order completed*). Otherwise the check is repeated until the result is positive. The example shown in Fig. 1 shows that EPCs are easy to read. It illustrates why EPCs have been accepted as a modeling technique by many persons involved in business process engineering projects.

EPC shown in Fig. 1 is a so-called basic EPC. It is possible to extend EPCs with entities (things in the real world), business objects (e.g. data), and organizational units. This way it is possible to model the input and output of a function in terms of entities, business objects, and parts of the organization. Moreover, it is possible to specify allocation rules and responsibilities. These extended event-driven process chains (eEPCs) are supported by tools such as ARIS and SAP R/3. In this paper, we abstract from these extensions and focus on the control flow.

3. Formalization of EPCs

Not every diagram composed of events, functions and connectors is a correct EPC. For example, it is not allowed to connect two events to each other (cf. Ref. [4]). Unfortunately, a formal syntax for EPCs is missing. In this section, we give a formal definition of an EPC. This definition is based on the restrictions described in Ref. [4] and imposed by tools such as ARIS and SAP R/3. This way we are able to specify the requirements that an EPC should satisfy.

Definition 1 (Event-driven process chain (1)). An EPC is a five-tuple (E, F, C, T, A) :

- E is a finite set of events,
- F is a finite set of functions,
- C is a finite set of logical connectors,
- $T \in C \rightarrow \{\wedge, \text{XOR}, \vee\}$ 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). The type of each connector is given by the function $T : T(c)$ 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 notation.

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 $\langle n_1, n_2, \dots, n_k \rangle$, so that $\langle n_i, n_{i+1} \rangle \in A$ for $1 \leq i \leq k-1$. p is elementary iff for any two nodes n_i and n_j on p , $i \neq j \Rightarrow n_i \neq n_j$.

The definition of the 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 T , we also partition C into C_{\wedge} , C_{\vee} , and C_{XOR} . The sets C_J and C_S are used to classify the connectors into join connectors and split connectors.

Definition 3 ($N, C_{\wedge}, C_{\vee}, C_{\text{XOR}}, \bullet, C_J, C_S, C_{EF}, C_{FE}$). Let $\text{EPC} = (E, F, C, T, A)$ be an *event-driven process chain*.

- $N = E \cup F \cup C$ is the set of nodes of EPC.
- $C_{\vee} = \{c \in C \mid T(c) = \vee\}$.
- $C_{\wedge} = \{c \in C \mid T(c) = \wedge\}$.
- $C_{\text{XOR}} = \{c \in C \mid T(c) = \text{XOR}\}$.
- 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.
- $C_{EF} \subseteq C$ so that $c \in C_{EF}$ if and only if there is a path $p = \langle n_1, n_2, \dots, n_{k-1}, n_k \rangle$, so that $n_1 \in E$, $n_2, \dots, n_{k-1} \in C$, $n_k \in F$, and $c \in \{n_2, \dots, n_{k-1}\}$.
- $C_{FE} \subseteq C$ so that $c \in C_{FE}$ if and only if there is a path

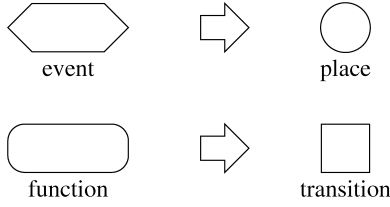


Fig. 3. Events are mapped onto places and functions are mapped onto transitions.

$$p = \langle n_1, n_2, \dots, n_{k-1}, n_k \rangle, \quad \text{so that} \quad n_1 \in F, \\ n_2, \dots, n_{k-1} \in C, n_k \in E, \text{ and } c \in \{n_2, \dots, n_{k-1}\}.$$

Consider for example the logical connector c connecting the function *check availability* and the events *articles available* and *articles need to be produced* in Fig. 1. Connector c is a split connector of type XOR connecting one function and two events, i.e. $c \in C_{\text{XOR}}$, $c \in C_S$, and $c \in C_{\text{FE}}$. Definition 3 enables us to specify the additional requirements an EPC should satisfy.

Definition 4 (Event-driven process chain (2)). An EPC i.e. $\text{EPC} = (E, F, C, T, A)$ satisfies the following requirements:

- 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 $e \in E$, so that $|\bullet e| = 0$ (i.e. a start event).
- There is at least one event $e \in E$, so 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 $|c \bullet| \geq 1$.
- The graph induced by EPC is weakly connected, i.e. for every two nodes $n_1, n_2 \in N$, $(n_1, n_2) \in (A \cup A^{-1})^*$.
- C_J and C_S partition C , i.e. $C_J \cap C_S = \emptyset$ and $C_J \cup C_S = C$.
- C_{EF} and C_{FE} partition C , i.e. $C_{\text{EF}} \cap C_{\text{FE}} = \emptyset$ and $C_{\text{EF}} \cup C_{\text{FE}} = C$.

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 the 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. For every two nodes n_1 and n_2 , there is a path from n_1 to n_2 (ignoring the direction of the arcs). A connector c is either a join connector ($|\bullet c| = 1$ and $|c \bullet| \geq 2$) or a split connector ($|\bullet c| \geq 1$ and $|c \bullet| = 1$). 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. It is easy to verify that the EPC shown in Fig. 1 is syntactically correct, i.e. all the requirements

stated in Definition 4 are satisfied. In the remainder of this paper, we assume all the EPCs to be syntactically correct.

Note that $\{C_J, C_S\}$, $\{C_{\text{EF}}, C_{\text{FE}}\}$, and $\{C_V, C_{\text{XOR}}, C_\wedge\}$ 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 $C = C_{\text{EF}} \cup C_{\text{FE}}$, and C_\wedge , C_{XOR} and C_V are pair-wise disjoint and $C = C_V \cup C_{\text{XOR}} \cup C_\wedge$. In principle there are $2 \times 2 \times 3 = 12$ kinds of connectors! In Ref. [4] two of these 12 constructs are not allowed: a split connector of type C_{EF} cannot be of type XOR or \vee , i.e. $C_S \cap C_{\text{EF}} \cap C_{\text{XOR}} = \emptyset$ and $C_S \cap C_{\text{EF}} \cap C_V = \emptyset$. 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 this paper, we will not impose this restriction, because in some cases it is convenient to model such a choice between functions.

4. Mapping EPCs onto Petri nets

Definitions (1) and (4) only relate to the syntax of an EPC and not to the semantics. In this section, we define the semantics in terms of a Petri net. As Petri nets have formal semantics, it is sufficient to map EPCs onto the Petri nets to specify the behavior unambiguously.

Note that some users may have problems with these formal semantics. They are used to drawing diagrams, which only give an intuitive description: the actual behavior is approximated by hiding details and exceptions. A precise and detailed diagram may be undesirable in the early stages of the (re)design of a business process. However, users of EPCs should not feel restricted by the presence of formal semantics. The formal semantics can be ignored during the early stages of the (re)design process. However, if EPCs are used as a starting point for analysis (e.g. simulation) or implementation using a WFM or ERP system, it is vital to have diagrams, which specify the behavior unambiguously.

In the remainder of this section, we assume the reader to be familiar with Petri nets. Appendix A introduces basic concepts, such as the firing rule, firing sequences, reachability, liveness, boundedness, strongly connectedness, and S -coverability. For an introduction to Petri nets the reader is referred to Ref. [8] or Ref. [22].

Fig. 3 shows the basic strategy that is used to map EPCs onto Petri nets: events correspond to places and functions correspond to transitions. The translation of connectors is much more complex. A connector may correspond to a number of arcs in the Petri net or to a small network of places and transitions. Fig. 4 shows the rules that are used to map connectors onto Petri net constructs. The behavior of a connector of type XOR corresponds to the behavior of a place, i.e. a connector of type XOR *agrees* with a node of type 'place' in the Petri net. A connector of type \wedge agrees with a node of type 'transition'. If the type of a *join* connector agrees the type of the *output* node in the corresponding Petri net, the connector is replaced by two or more arcs. For example, a join connector of type \wedge corresponds to a

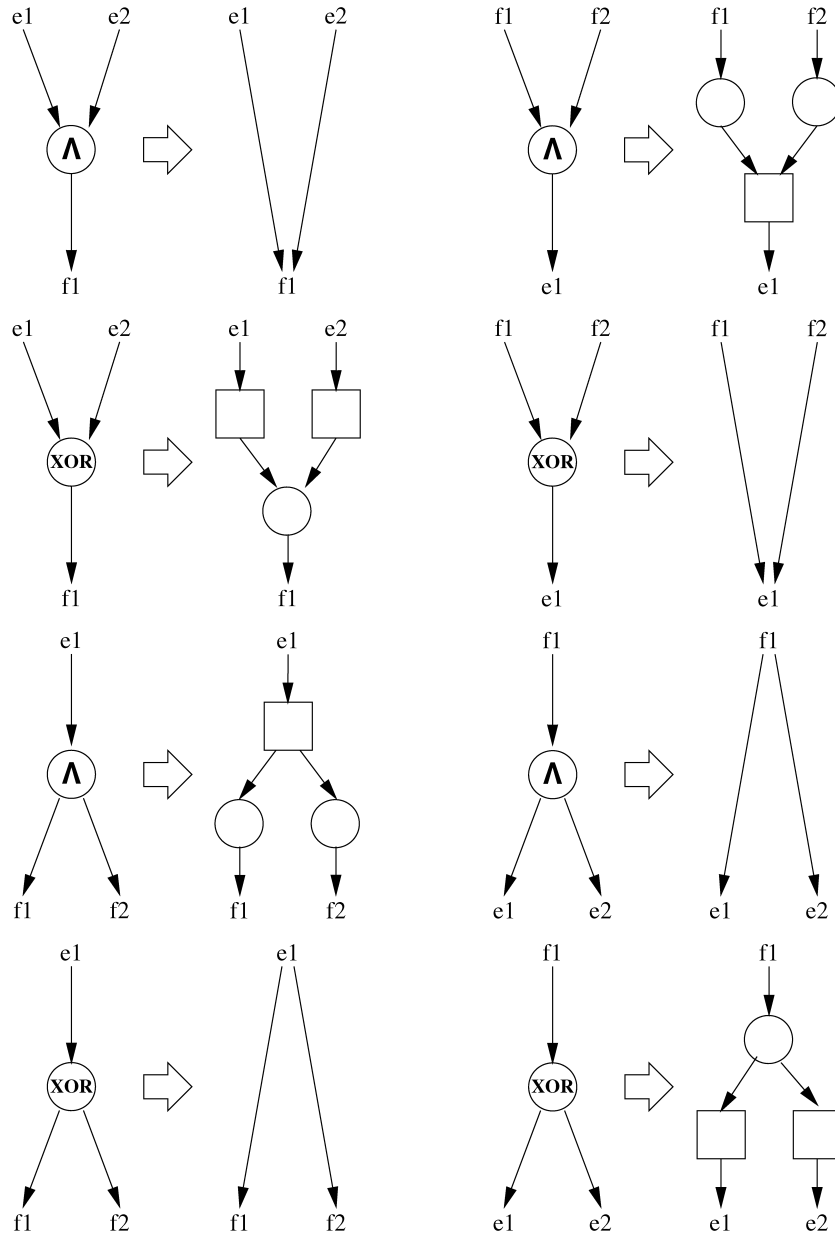


Fig. 4. Mapping connectors onto places and transitions.

number of arcs in the Petri net if and only if the output node is a transition (see Fig. 4). If the type of a join connector and the type of the output node do not agree, the connector is replaced by a small network. If the type of a *split* connector does not agree with the type of the *input* node in the Petri net, the connector is replaced by a small network. Otherwise, the connector is replaced by a number or arcs.

Fig. 4 does not give any constructs for connectors of type \vee . The semantics of join connectors of type \vee are not clear. This problem is tackled in Section 6. For the moment, we assume all the connectors to be of type \wedge or XOR. Based on this assumption the formalization of the mapping is rather straightforward.

Definition 5 (\mathcal{N}). Let $\text{EPC} = (E, F, C, T, A)$ be an EPC with $C_v = \emptyset$ and $A \cap (C \times C) = \emptyset$. $\mathcal{N}(\text{EPC}) = (P^{\text{PN}}, T^{\text{PN}}, F^{\text{PN}})$ is the Petri net generated by EPC: $P^{\text{PN}} = E \cup (\bigcup_{c \in C} P_c^{\text{PN}})$, $T^{\text{PN}} = F \cup (\bigcup_{c \in C} T_c^{\text{PN}})$, and $F^{\text{PN}} = (A \cap ((E \times F) \cup (F \times E))) \cup (\bigcup_{c \in C} F_c^{\text{PNF}})$. See Table 1 for the definition of P_c^{PN} , T_c^{PN} , and F_c^{PNF} .

The places in the Petri net correspond either to events or to constructs needed to model the behavior of a connector in the EPC. The transitions correspond to functions or are the result of the translation of a connector. Each connector $c \in C$ corresponds to the places, transitions and/or arcs listed in Table 1. In Table 1 it is assumed that the connectors are only connected to functions and events, i.e. $A \cap (C \times C) = \emptyset$.

Table 1
Mapping an EPC connector $c \in C$ onto places, transitions, and arcs

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

Although it is possible to extend Table 1 with additional rules for connections between connectors, we use an alternative approach. Every arc connecting two connectors is replaced by an event and a function, i.e. fake events and functions are added to the EPC before the translation to a Petri net. Fig. 5 illustrates the approach that is used to handle arcs in $A \cap (C \times C)$. The arc between the XOR-join (join connector of type XOR) and the AND-join (join connector of type \wedge) is replaced by *function X* and *event X* and three arcs. The arc between the AND-join and the XOR-split is also replaced by a function, an event and three arcs.

Fig. 6 shows the Petri net, which corresponds to the EPC shown in Fig. 1. Note that the arc between the two XOR connectors is replaced by an event and a function, and

mapped onto an additional place and transition in the Petri net. In this case there was no real need to add these additional nodes. However, there are situations, where adding events and functions is the only way to model the control flow properly.

It is easy to see that for any $\text{EPC} = (E, F, C, T, A)$ satisfying the requirements in Definition 4, $\mathcal{N}(\text{EPC}) = (P^{\text{PN}}, T^{\text{PN}}, F^{\text{PN}})$ is a Petri net, i.e. $P^{\text{PN}} \cap T^{\text{PN}} = \emptyset$ and $F^{\text{PN}} \subseteq (P^{\text{PN}} \times T^{\text{PN}}) \cup (T^{\text{PN}} \times P^{\text{PN}})$. Moreover, the Petri net is free-choice (see Definition 12).

Lemma 1. *Let $\text{EPC} = (E, F, C, T, A)$ be an event-driven process chain and $\text{PN} = \mathcal{N}(\text{EPC})$ the Petri net generated by EPC. PN is free-choice.*

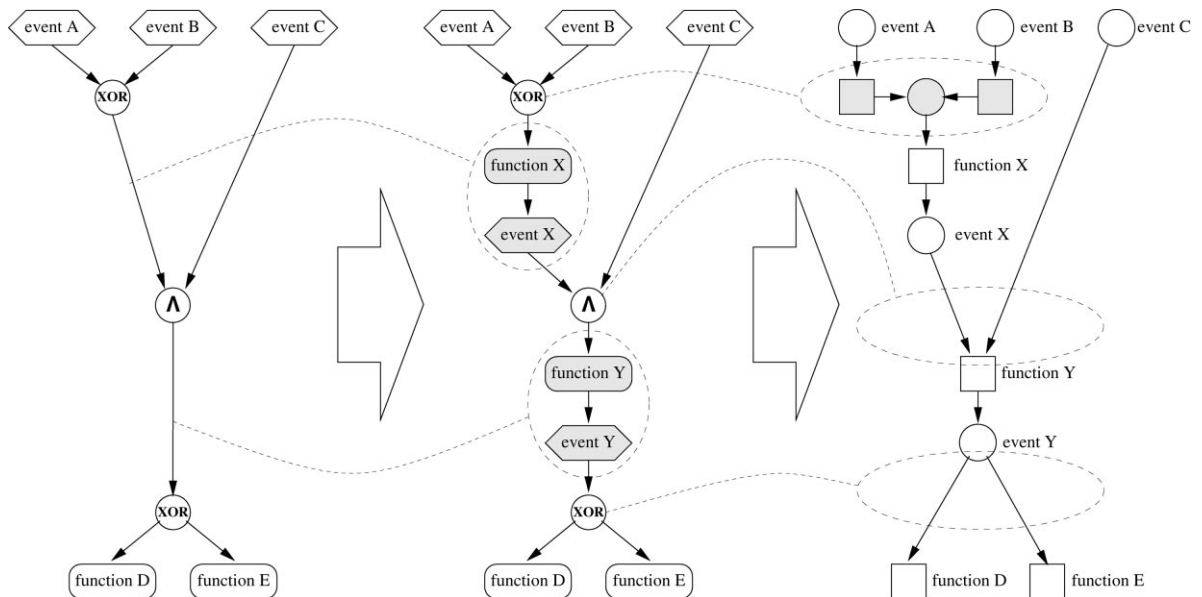


Fig. 5. The arcs between connectors are replaced by events and functions before the EPC is mapped onto a Petri net.

Proof. We have to prove that for any two transitions t_1, t_2 sharing an input place, $\bullet t_1 = \bullet t_2$. Therefore, we have to check every place with two or more output arcs. An event cannot have more than two output arcs. The only way to obtain a place with multiple output arcs is the mapping of XOR-split connectors onto Petri net constructs (see Fig. 4). However, the rules given in Table 1 guarantee that the output transitions have identical sets of input places. Therefore, the Petri net is free-choice. \square

The free-choice Petri nets [23] are a class of Petri nets for which strong theoretical results and efficient analysis techniques exist. For example, the well-known Rank Theorem [23] allows for the efficient analysis of liveness and boundedness. In free-choice Petri nets it is not allowed to mix choice and synchronization (see Definition 12). Lemma 1 shows that in an EPC choice and synchronization are separated. On the one hand, Lemma 1 is a positive result, because it shows that the EPC correspond to a sub-class with many desirable properties. On the other hand, the result shows that the expressive power of EPCs is limited compared to Petri nets. It is not possible to model advanced control structures, which involve a mixture of choice and synchronization (cf. [9,10]).

5. Verification of EPCs

The correctness, effectiveness, and efficiency of the business processes supported by ERP or WFM systems are vital to the organization. An information system, which is based on erroneous EPCs may lead to serious problems such as angry customers, back-log, damage claims, and loss of goodwill. Flaws in the design of an EPC may also lead to high throughput times, low service levels, and a need for excess capacity. Therefore, it is important to analyze the EPC before it is put into production. In this section, we focus on the *verification* (i.e. establishing the correctness) of EPCs. The bridge between EPCs and the Petri nets presented in this paper allows for the use of the powerful analysis techniques that have been developed for Petri nets [8]. Linear algebraic techniques can be used to verify many properties, e.g. place invariants, transition invariants, and (non-)reachability. Other techniques such as coverability graph analysis, model checking, and reduction techniques can be used to analyze the dynamic behavior of the EPC mapped onto a Petri net. In this section, we will show that the Petri-net-based techniques can be used to analyze the so-called *soundness property*. To simplify the definition of this property, we restrict ourselves to *regular* EPCs.

Definition 6 (Regular). An EPC is regular if and only if:

- (i) EPC has two special events: e_{start} and e_{final} . Event e_{start} is a source node: $\bullet e_{\text{start}} = \emptyset$. Event e_{final} is a sink node: $e_{\text{final}} \bullet = \emptyset$.
- (ii) Every node $n \in N$ is on a path from e_{start} to e_{final} .

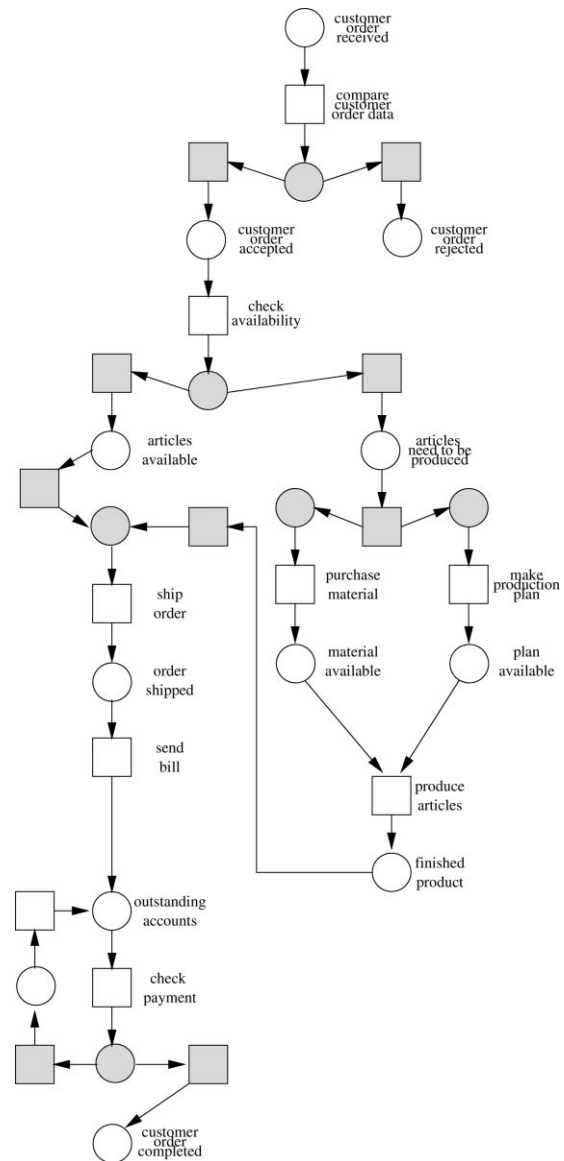


Fig. 6. The EPC of Fig. 1, mapped onto a Petri net.

The identification of e_{start} and e_{final} allows for a clear definition of the initial state and the final state. An EPC with multiple start events (i.e. events without any input arcs) or multiple final events (i.e. events without any output arcs) can easily be extended with an initialization and/or a termination part so that, the first requirement is satisfied. The second requirement demands that every event or function is in the scope bordered by e_{start} and e_{final} . If the original EPC is extended with an initialization and/or a termination part so that the first requirement is satisfied, then the second requirement is quite natural. If the second requirement is not satisfied, then the EPC is: (1) composed of completely disjunct parts, (2) it has parts, which are never activated, or (3) parts of the EPC form a trap. As the EPC describes the life cycle of a case (i.e. a process instance), the two requirements are reasonable. The life cycle should have a clear begin event and end event, and all the steps should be on

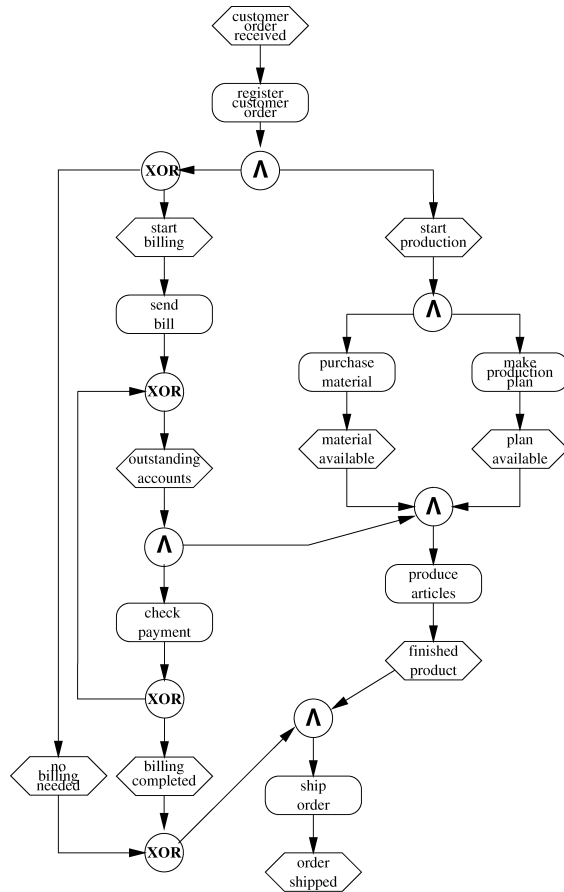


Fig. 7. An erroneous EPC.

a path between these two events. In the remainder of this paper, we assume all EPCs to be regular.

An EPC describes a procedure with an initial state and a final state. The procedure should be designed in such a way that it always terminates properly. Moreover, it should be possible to execute any given function by following the appropriate route through the EPC.

Definition 7 (Sound). A regular EPC is sound if and only if:

- (i) For every state M reachable from the initial state (i.e. the state, where event e_{start} is the only event that holds), there exists a firing sequence leading from state M to the final state (i.e. the state, where event e_{final} is the only event that holds).
- (ii) The final state (i.e. event e_{final} is the only event that holds) is the only state reachable from the initial state, where event e_{final} holds.
- (iii) There are no dead functions, i.e. for each function f there is a firing sequence, which executes f .

The correctness criterion defined in Definition 9 is the minimal requirement any EPC should satisfy. A sound

EPC is free of potential deadlocks and livelocks. If we assume fairness (cf. [24]), then the first two requirements imply that eventually the final state will be reached. (Note that this is a result of the combination of soundness and the free-choice property [25].) Consider for example the EPC shown in Fig. 1. To make the EPC regular, the two sink events (*customer order completed* and *customer order rejected*) are joined by a termination part (i.e. an XOR-join connector, a function, and an event). By checking all possible firing sequences it is possible to verify that the EPC is sound, i.e. it is guaranteed that eventually every customer order that is received will be handled completely.

Fig. 7 describes an alternative process to handle customer orders. The EPC shown in Fig. 7 is not sound. If the billing process is skipped (i.e. the event *no billing needed* holds after executing the function *register customer order*), then the EPC will deadlock (the input event *outstanding accounts* of the AND-join preceding the function *produce articles* will never hold). If multiple payment checks are needed, then the fake event (the event added to connect the AND-split and the AND-join) will hold after termination.

For the small examples shown in this paper it is easy to see, whether an EPC is sound or not. However, for the complex EPCs encountered in practice it is far from trivial to verify the soundness property. Fortunately, Petri-net-based techniques and tools can be used to analyze the soundness property. Inspection of the coverability graph [8,22] of the Petri net, which corresponds to the EPC, is sufficient to decide soundness. For complex EPCs, the coverability graph may become very large. This phenomenon is known as the ‘state explosion problem’. A typical EPC with 80 tasks and events can easily have more than 200 000 states. Although today’s computers have no problem analyzing coverability graphs of this size, there are more advanced techniques which exploit the structure of the Petri net generated by an EPC. These techniques allow for very efficient decision procedures. Before presenting such a procedure, we first list some properties that hold for any Petri net generated by a sound EPC.

Proposition 1. Let $\text{EPC} = (E, F, C, T, A)$ be a sound event-driven process chain and $\text{PN} = N(\text{EPC})$ the Petri net generated by the EPC. Let $\overline{\text{PN}}$ be PN with an extra transition t connecting e_{final} to e_{start} and let M be the initial state with one token in e_{start} .

- $\overline{\text{PN}}$ is strongly connected,
- $\overline{\text{PN}}$ is S -coverable,
- $(\overline{\text{PN}}, M)$ is live,
- $(\overline{\text{PN}}, M)$ is bounded, and
- $(\overline{\text{PN}}, M)$ is safe.

Proof. $\overline{\text{PN}}$ is strongly connected because every node is on a path from e_{start} to e_{final} and e_{final} is connected to e_{start} via t^* .

PN is a WF-net (see Ref. [9]). Therefore, soundness

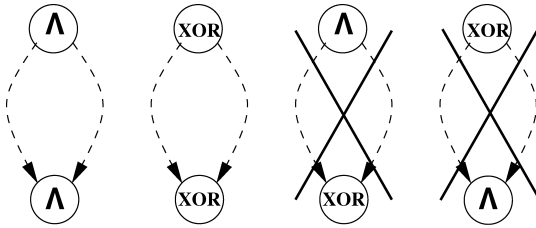


Fig. 8. Well-structuredness is based on the distinction between good constructs (left) and bad constructs (right).

coincides with liveness and boundedness (see Theorem 11 in Ref. [9]).

(\overline{PN}, M) is free-choice, live, and bounded. In Ref. [26] it is shown that this implies that \overline{PN} is S -coverable and (\overline{PN}, M) is safe. \square

Building on the results presented in Ref. [9], we can prove that soundness can be verified in polynomial time.

Theorem 1. *An EPC can be checked for soundness in polynomial time.*

Proof. An EPC corresponds to a free-choice WF-net (see Ref. [9]). A WF-net is sound if and only if the extended net (\overline{PN}, M) is live and bounded. Liveness and boundedness can be decided in polynomial time (use the well-known Rank Theorem [23]). Hence, soundness can be verified in polynomial time. \square

Theorem 1 shows that it is possible to extend tools such as ARIS and SAP R/3 with efficient decision procedures to verify the correctness of an EPC. To guide the user in finding and correcting flaws in the design of an EPC, it is also possible to supply additional diagnostics based on the structure of the EPC/Petri net. For example, it is useful to check whether the EPC is *well-structured*.

Definition 8 (Well-structured). An EPC is well-structured iff for any pair of connectors $c_1 \in C_S$ and $c_2 \in C_J$, so that one of the nodes is in C_Λ and the other in C_{XOR} and for any pair of elementary paths p_1 and p_2 leading from c_1 to c_2 , $\alpha(p_1) \cap \alpha(p_2) = \{c_1, c_2\} \Rightarrow p_1 = p_2$.¹

Fig. 8 illustrates the concept of well-structuredness. An EPC is not well-structured if an AND-split is complemented by an XOR-join, or an XOR-split is complemented by an AND-join. Consider for example the EPC shown in Fig. 7. The XOR-split before *start billing* is complemented by the AND-join before *ship order*: there is an elementary path from this XOR-join to the AND-join via *no billing needed* and there is an elementary path from this XOR-join to the AND-join via *start billing, send bill, outstanding accounts,*

produce articles, and *finished product*. The fact that there is an imbalance between the splits and the joins reveals the source of the error in the design of the EPC shown in Fig. 7. The XOR-split can bypass the synchronization point before *produce articles*.

It is possible to have a sound EPC, which is not well-structured. Nevertheless, well-structuredness is a desirable property. If possible, non-well-structured constructs should be avoided. If such a construct is really necessary, then the correctness of the construct should be double-checked, because it is a potential source of errors.

Another diagnostic which might be useful is a list of all S -components (see Definition 15). If an event or a function corresponds to a place or transition not in any S -component, then this indicates an error. Note that a Petri net which corresponds to a sound EPC is S -coverable (see Proposition 1).

Woflan [27] is an example of a Petri-net-based analysis tool, which can be used to decide soundness. *Woflan* (WORKFLOW ANalyzer) has been designed to verify process definitions, which are downloaded from a WFM system. At the moment there are two workflow tools which can interface with *Woflan*: COSA (COSA Solutions/Software-Ley, Pullheim, Germany) and Protos (Pallas Athena, Plasmolen, The Netherlands). *Woflan* generates the diagnostics mentioned in this paper (and many more). In the future we hope to provide a link between *Woflan* and EPC-based tools such as ARIS and SAP R/3.

6. Connectors of type \vee

In the second half of this paper (i.e. Sections 4 and 5), we abstracted from \vee -connectors. The reason for this abstraction is the fact that the semantics of join connectors of type \vee are not clear. Consider for example the EPC shown in Fig. 9. If event X holds, function A or B needs to be executed. There are three possibilities: (1) function A , (2) function B , or (3) function A and function B are executed. Although the intention of the construct shown in Fig. 9 is clear, it is difficult to give formal semantics for the \vee -connectors. Fig. 10 shows a Petri net, where the \vee -connectors are mapped onto places and transitions in a straightforward manner. The transitions S_A , S_AB and S_B model the OR-split by a choice between the three possibilities. The OR-join is modeled in a similar way. Clearly, the Petri net shown in Fig. 10 is not the correct way to model the EPC in Fig. 9. If S_AB fires, J_A and J_B may fire after the completion of both functions. Moreover, event Y may hold before both functions are actually executed. The core of the problem is that it is difficult to determine whether a join connector of type \vee should synchronize or not.

There are several ways to deal with connectors of type \vee .

- First of all, it is possible to *refine* the EPC until all connectors of type \vee are replaced by connectors of type \wedge and/or XOR. This may lead to an ‘explosion’ of

¹ The alphabet operator α is defined as follows. If $p = \langle n_1, n_2, \dots, n_k \rangle$, then $\alpha(p) = \{n_1, n_2, \dots, n_k\}$.

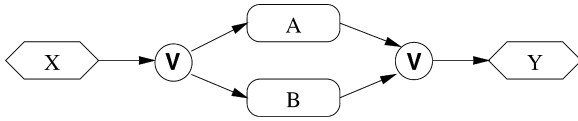


Fig. 9. An EPC showing the use of \vee connectors.

the EPC. Note that if there are n functions connected to each other by \vee -connectors, then there are $2^n - 1$ possible combinations. For example, for 10 functions, there are 1023 possible combinations.

- Secondly, it is possible to *couple* OR-splits and OR-joins, i.e. the selection made by the split connector of type \vee is used by the join connector of type \vee to see when all inputs have arrived. The semantics of such a concept can be expressed in terms of a Petri net. Consider for example Fig. 10: the synchronization problem can be resolved by adding a place connecting S_A to J_A , a place connecting S_AB to J_AB , and a place connecting S_B to J_B .
- Finally, it is possible to *adapt* the *firing rule* with respect to transitions, which correspond to a join connector of type \vee . A transition t of a set of transitions X , which corresponds to an OR-join is enabled if and only if (1) the input places of $t(\bullet t)$ are marked and (2) it is not possible that additional input places of $X(\bullet X)$ will become marked by firing other transitions. In other words, postpone the join until the moment that it is clear that the input is maximal.

None of the solutions seems to be very satisfactory. The first solution will lead to large and unreadable EPCs. The second solution only works if every OR-split is complemented by an OR-join, i.e. the EPC is symmetric with respect to connectors of type \vee . The last solution leads to situations which are difficult to interpret and is difficult to formalize. Further research is needed to tackle the problem in a more satisfactory manner.

7. Conclusion

In this paper, we have presented formal semantics for EPCs. Although EPCs have become a widespread process modeling technique, such a basis was missing. By mapping EPCs onto Petri nets, we have tackled this problem. In

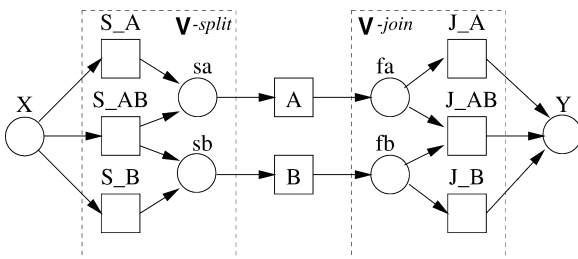


Fig. 10. A Petri net which corresponds to the EPC in Fig. 9.

addition, many analysis techniques have become available for EPCs. This has been demonstrated by a decision procedure, which verifies the correctness (i.e. soundness) in polynomial time. The results presented in this paper give designers a handle to construct correct EPCs.

The approach presented in this paper can also be applied to extensions of EPCs, e.g. eEPC [7] and oEPC [28]. These extensions typically add: (1) a data view, (2) an organizational view, (3) and a functional view to the process view described by the traditional EPC. For the verification of the process view it is reasonable to abstract from these additional aspects. The process by itself should be correct. The consistency between the various views can be verified using techniques outside the scope of this paper. See Ref. [10] for a similar discussion in the context of WFM.

At the moment we are working on two directions for further research. First of all, we are working on a tool to support the approach presented in this paper. We hope to extend Woflan so that the EPCs exported by ARIS and SAP R/3 can be analyzed directly. Secondly, we are looking for a more satisfactory way to deal with connectors of type \vee .

Appendix A. Petri nets

This appendix introduces the basic Petri net terminology and notations.² The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between the two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

Definition 9 (Petri net). A Petri net is a triple (P, T, F) :

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

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t . Place p is called an *output place* of transition t iff there exists a directed arc from t to p . We use $\bullet t$ to denote the set of input places for a transition t . The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g. $p\bullet$ is the set of transitions sharing p as an input place.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e. $M \in P \rightarrow \mathbb{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1 , two tokens in p_2 , one token in p_3 and no tokens in p_4 . We can also represent this state as follows: $p_1 + 2p_2 + p_3$. To compare states, we define a partial ordering. For any two states M_1 and M_2 , $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$.

² Note that states are represented by weighted sums and note the definition of (elementary) paths.

Note that we restrict ourselves to arcs with weight 1. In the context of EPCs it makes no sense to have other weights, because places correspond to conditions. In a Petri net corresponding to a correct (i.e. sound) EPC a place will never contain multiple tokens (i.e. the net is safe). The states with multiple tokens in one place are the result of design errors. To capture these errors, we need to consider non-safe nets.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

- (1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.
- (2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token in each output place p of t .

Given a Petri net (P, T, F) and a state M_1 , we have the following notations:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2 .
- $M_1 \xrightarrow{t} M_2$: there is a transition t so that $M_1 \xrightarrow{t} M_2$.
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \dots t_{n-1}$ leads from state M_1 to state M_n , i.e. $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \dots \xrightarrow{t_{n-1}} M_n$.

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence $\sigma = t_1 t_2 \dots t_{n-1}$ so that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is also allowed, i.e. $M_1 \xrightarrow{*} M_1$.

We use (PN, M) to denote a Petri net with an initial state M . A state M' is a *reachable state* of (PN, M) iff $M \xrightarrow{*} M'$. Let us define some properties for Petri nets.

Definition 10 (Live). A Petri net (PN, M) is live iff for every reachable state M' and every transition t , there is a state M'' reachable from M' , which enables t .

Definition 11 (Bounded, safe). A Petri net (PN, M) is bounded iff for each place p there is a natural number n so that for every reachable state the number of tokens in p is less than n . The net is safe iff for each place the maximum number of tokens does not exceed 1.

Definition 12 (Free-choice). A Petri net is a free-choice Petri net iff for every two transitions t_1 and t_2 , $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$.

Definition 13 (Path). Let PN be a Petri net. A path C from a node n_1 to a node n_k is a sequence $\langle n_1, n_2, \dots, n_k \rangle$ so that $\langle n_i, n_{i+1} \rangle \in F$ for $1 \leq i \leq k-1$.

Definition 14 (Strongly connected). A Petri net is strongly connected iff for every pair of nodes (i.e. places and transitions) x and y , there is a path leading from x to y .

Definition 15 (S-coverable). A Petri net $PN = (P, T, F)$ is *S-coverable* iff for each place p there is subnet $PN_s = (P_s, T_s, F_s)$ so that: $p \in P_s, P_s \subseteq P, T_s \subseteq T, F_s \subseteq F, PN_s$ is strongly connected, PN_s is a state machine (i.e. each transition in PN_s has one input and one output arc), and for every $q \in P_s$ and $t \in T : (q, t) \in F \Rightarrow (q, t) \in F_s$ and $(t, q) \in F \Rightarrow (t, q) \in F_s$.

A subnet PN_s which satisfies the requirements stated in Definition 15 is called an *S-component*. PN_s is a strongly connected state machine so that for every place q : if q is an input (output) place of a transition t in \overline{PN} , then q is also an input (output) place of t in PN_s .

References

- [1] A.W. Scheer, Business Process Engineering, Reference Models for Industrial Enterprises, Springer, Berlin, 1994.
- [2] S. Jablonski, C. Bussler, Workflow Management: Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, 1996.
- [3] P. Lawrence (Ed.), Workflow Handbook 1997, Workflow Management Coalition Wiley, New York, 1997.
- [4] G. Keller, M. Nüttgens, 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.
- [5] H. Bancroft, H. Seip, A. Sprengel, Implementing SAP R/3: how to introduce a large system into a large organization, 1997.
- [6] J. Hernandez, The SAP R/3 Handbook, 1997.
- [7] A.W. Scheer, Business Process Engineering, ARIS-Navigator for Reference Models for Industrial Enterprises, Springer, Berlin, 1994.
- [8] T. Murata, Petri nets: properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989) 541–580.
- [9] W.M.P. van der Aalst, Verification of workflow nets, in: P. Azema, G. Balbo (Eds.), Application and Theory of Petri Nets 1997, *Lecture Notes in Computer Science*, 1248, Springer, Berlin, 1997, pp. 407–426.
- [10] W.M.P. van der Aalst, The application of Petri nets to workflow management, The Journal of Circuits, Systems and Computers 8 (1) (1998) 21–66.
- [11] W.M.P. van der Aalst, Three good reasons for using a Petri-net-based workflow management system, in: T. Wakayama (Ed.), Information and Process Integration in Enterprises: Rethinking documents, *The Kluwer International Series in Engineering and Computer Science* Kluwer Academic Publishers, Norwell, 1998, pp. 161–182, Chapter 10.
- [12] W.M.P. van der Aalst, K.M. van Hee, Workflow Management: Modellen, Methoden en Systemen (in Dutch), Academic Service, Schoonhoven, 1997.
- [13] C. Ellis, K. Kedara, G. Rozenberg, Dynamic change within workflow systems, in: N. Comstock, C. Ellis, (Eds.), Conference on Organizational Computing Systems, ACM SIGOIS, ACM, Milpitas, CA, Aug 1995, pp. 10–21.
- [14] C.A. Ellis, G.J. Nutt, Modelling and enactment of workflow systems, in: M.A. Marsan (Ed.), Application and Theory of Petri Nets 1993, *Lecture Notes in Computer Science*, 691, Springer, Berlin, 1993, pp. 1–16.
- [15] G. De Michelis, C. Ellis, G. Memmi (Eds.), Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms, Zaragoza, Spain, June 1994.

- [16] R. Chen, A.W. Scheer, Modellierung von Prozessketten mittels Petri-Netz Theorie, Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 107 (in German), University of Saarland, Saarbrücken, 1994.
- [17] P. Langner, C. Schneider, J. Wehler, Ereignisgesteuerter Prozessketten und Petri-Netze, University of Hamburg, Hamburg, 1997.
- [18] P. Langner, C. Schneider, J. Wehler, Petri net based certification of event driven process chains, in: J. Desel, M. Silva (Eds.), Application and Theory of Petri Nets 1998, *Lecture Notes in Computer Science*, 1420, Springer, Berlin, 1998, pp. 286–305.
- [19] M. Nüttgens, Event-driven Process Chain (EPC)/Ereignisgesteuerte Prozesskette (EPK), <http://www.iwi.uni-sb.de/nuettgens/EPK/epk.htm>
- [20] F. Rump, Erreichbarkeitsgraphbasierte Analyse ereignisgesteuerter Prozessketten, Technischer Bericht, Institut OFFIS, 04/97 (in German), University of Oldenburg, Oldenburg, 1997.
- [21] F. Rump, Arbeitsgruppe Formalisierung und Analyse ereignisgesteuerter Prozessketten (EPK), <http://www-is.informatik.uni-oldenburg.de/~sim;epk/>
- [22] W. Reisig, Petri Nets: An Introduction, Monographs in Theoretical Computer Science: An EATCS Series, 4, Springer, Berlin, 1985.
- [23] J. Desel, J. Esparza, Free Choice Petri Nets, Cambridge Tracts in Theoretical Computer Science, 40, Cambridge University Press, Cambridge, 1995.
- [24] R. Valk, Infinite behaviour and fairness, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties, *Lecture Notes in Computer Science*, 254, Springer, Berlin, 1987, pp. 377–396.
- [25] E. Kindler, W.M.P. van der Aalst, Liveness, fairness, and recurrence (under review), 1998.
- [26] W.M.P. van der Aalst, Structural characterizations of sound workflow nets, Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996.
- [27] W.M.P. van der Aalst, D. Hauschildt, H.M.W. Verbeek, A Petri-net-based tool to analyze workflows, in: B. Farwer, D. Moldt, M.O. Stehr (Eds.), Proceedings of Petri Nets in System Engineering (PNSE'97), Hamburg, September 1997. University of Hamburg (FBI-HH-B-205/97), pp. 78–90.
- [28] M. Nüttgens, V. Zimmerman, Geschäftsprozessmodellierung mit der objektorientierten Ereignisgesteuerten Prozesskette (oEPK), in: M. Maicher, H.J. Scheruhn (Eds.), Informationsmodellierung-Branchen, Software-und Vorgehensreferenzmodelle und Werkzeuge, 1998, pp. 23–36.