International Journal of Cooperative Information Systems
© World Scientific Publishing Company

BRIDGING THE GAP BETWEEN BUSINESS MODELS AND WORKFLOW SPECIFICATIONS

JULIANE DEHNERT
Fraunhofer ISST, Mollstr. 1, 10178 Berlin, Germany
juliane.dehnert@isst.fraunhofer.de

WIL M.P. VAN DER AALST
Department of Information Systems, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, Nederland
w.m.p.v.d.aalst@tm.tue.nl

Received (to be inserted
Revised by Publisher)

This paper presents a methodology to bridge the gap between business process modeling and workflow specification. While the first is concerned with intuitive descriptions that are mainly used for communication, the second is concerned with configuring a process-aware information system, thus requiring a more rigorous language less suitable for communication. Unlike existing approaches the gap is not bridged by providing formal semantics for an informal language. Instead it is assumed that the desired behavior is just a subset of the full behavior obtained using a liberal interpretation of the informal business process modeling language. Using a new correctness criterion (relaxed soundness), it is verified whether a selection of suitable behavior is possible. The methodology consists of five steps and is illustrated using event-driven process chains as a business process modeling language and Petri nets as the workflow specification language.

Keywords: Business Process Modeling, Workflow Modeling, Event-Driven Process Chains, Petri nets, Controller Synthesis .

1. Introduction

Over the last decade information systems have increasingly become "process aware". Note that ERP systems like SAP R/3 are based on reference models describing typical business processes within organizations. However, many processes remain non-standard and require extensive configuration based on process descriptions. This is the reason SAP R/3, like most ERP systems, offers a workflow module. There are also dedicated Workflow Management (WFM) systems such as Staffware[1]. The idea of configuring information systems on the basis of explicit process descriptions is not specific for ERP and WFM systems as is illustrated by emerging languages like BPEL4WS[2,3] in the domain of web services. In any case, an important task is the specification of the workflow process, i.e., providing a process description that can be interpreted by the software to support the corresponding process.

Abstracting from the non-automated parts, many scientific papers neglect the differences between a business process and its supporting workflow. As a consequence of the limited understanding of their differences, no distinction is made between a business process description and a workflow specification. However in practice this difference is highly relevant. Business process descriptions are used for a different purpose and made by different people than workflow specifications.

A business process description is made by domain experts. The objective of a business process description is to provide a basis for communication. The descriptions are used for various purposes. In the everyday life of a company they serve as manuals for process participants or as learning material for newcomers. In business process re-engineering projects they provide a basis for discussion in order to detect optimization potential. In preparation for the use of a WFMS they provide a basis for agreeing on the processes to be supported. The business process description must be understandable for people from very different backgrounds and "knowledge cultures", e.g. heads of departments, department staff, and IT experts. A business process description should be intuitive and leave room for interpretation: the more ways there are to interpret a certain construct the more likely it is that agreement will be reached.

A workflow specification, in contrast, is made by IT-experts. It is used as input for a WFMS and must therefore be machine readable. A workflow specification must be unambiguous and should not contain any uncertainties. This is a necessary requirement in order to analyze and simulate the described processes and to monitor their execution at run-time. A workflow specification also contains details that are relevant for implementation. Whereas it is sufficient for a business process description to cover the set of desired process executions, a workflow specification also determines how these executions are achieved. Thus, the workflow specification incorporates a strategy scheduling the executions supported at run-time.

Still, both process descriptions[a] cover the same matter of interest: the involved tasks and their order. The close relation between the two descriptions suggests deriving one from the other by changing the level of abstraction, e.g., enhancing existing business process descriptions such that they can be used as inputs for a WFMS.

1.1. Problem Description

So far, there is no methodically well-founded process model that bridges the gap between business process- and workflow modeling. One reason can be found in a badly organized communication between domain- and IT-experts. But even if those involved work closely together, the continuous use of business process descriptions

[a]The generic term *process description* will be used for both business process description and workflow specification.

for the modeling of workflow is hardly possible as there is neither a standard modeling language supporting the different abstraction levels nor an exchange format in combination with transformation rules to transform business process descriptions into workflow specifications.

There are languages that have proved to be understandable for the average user. They provide a set of graphical modeling elements which are combined in a straightforward manner. Their semantics is "intuitive" but not formally founded. On the other hand, there is a variety of languages that satisfy the requirements posed by a workflow specification. They have formal, operational semantics and provide concepts to specify aspects close to implementation. While examples of the first class of languages support understandable models they typically lack formal semantics; examples of the second class of languages provide formal semantics but are often not accepted by the modelers. There have been many attempts to bridge the gap between the two concerns: the need for a plain communication basis on the one side and an unambiguous process description, covering details of the implementation, on the other side.

Existing WFMSs follow a pragmatic approach. They often use a proprietary modeling language with an intuitive graphical layout. The underlying semantics lacks a formal foundation. As a consequence, analysis issues, such as proving correctness and reliable execution are not supported at design time. Failure detection is only possible while monitoring the process execution. It is clear that failures that are only detected at run-time may be very costly and may cause all kinds of problems.

Many research approaches address this drawback of practical systems. Considerable efforts went into the formalization of semi-formal modeling languages, such as Activity Diagrams[4,5], Statecharts[6,7], and Event-driven Process Chains[8,9,10,11,12,13]. Still, these approaches do not provide a solution. The formalization removes ambiguities and restricts the expressiveness. This moves the derived description more towards a suitable input for workflow management systems, but does not improve acceptance from modelers. Various interpretations which supported reaching an agreement between the different participants were discarded.

Other approaches try to adapt and/or facilitate the use of formal languages such as Petri nets[14]. Here the most common approach is the introduction of intuitive graphical patterns replacing constructs of the primary language.[15] The goal is to facilitate the understanding of the determined interpretation. Another approach based on Petri nets is implemented within the WFMS of MILANO.[16] Here multiple, interchangeable representations are used as front-end to the modeler. Still the degree of precision is the same.

Coming from either direction, the main idea is to define a comprehensive modeling language which meets all requirements, i.e., to provide concepts that support the different levels of abstraction. As much as such a general language would simplify life, so far none of the proposed languages provides concepts to cover the different levels of abstraction separately, e.g. through a suitable refinement relation. Instead, aspects of both abstraction levels get mingled.

The goal of this work is not to propose “Yet Another Modeling Language”[17], which could bridge the gap, but to take a different and more pragmatic approach.

It is interesting to note that also in the context of recent developments like BPEL4WS the gap between business models and workflow specifications remains. In the context of BPEL4WS[3] the same language is proposed to provide both an executable language and an abstract specification. In our view this is far from realistic; a language like BPEL4WS is at a too low level to be considered as an abstract specification.[2]

1.2. Problem Solving Approach

The idea is to propose a cross-language process model which gradually guides the modeler towards a sound workflow specification. The process model is not based upon one general modeling language but supports the combination of different, existing, and accepted techniques. The proposed procedure starts with the modeling of a business process using an “intuitive”, but semi-formal modeling language. The procedure finally guides the modeler towards a sound workflow specification, which is given in terms of a formal language.

As language for the workflow specification we chose Petri nets. Their suitability for this application domain has been examined and discussed extensively in the literature[15,18]. They combine a graphical representation with a precise formal foundation. Their operational semantics allow the use of the derived process descriptions right away as input format of a WFMS. Examples of existing WFMSs working on the basis of Petri net descriptions are COSA (Software Ley/COSA Solutions/Transflow[19]) and Income (Get Process AG[20]).

As a modeling language for the business process description we refer to Event-driven Process Chains (EPCs), which are fairly widespread. Reasons for their acceptance can be found in their use for the representation of the SAP reference models[21] and their tool-support through the ARIS tool set[22]. But, EPCs are just one of a rich variety of accepted business process modeling languages. We emphasize that the proposed process model is not restricted to that choice, but may be adapted for other semi-formal techniques.

Five steps have to be completed when guiding the modeler from a semi-formal business process description (expressed in terms of an EPC) towards a sound workflow specification based on Petri nets:

1. Business process modeling (EPCs),
2. Transformation into WF-nets,
3. Correctness check and feedback,
4. Strategy determination and

5. Strategy implementation.

The whole process model is illustrated in Figure 1. It has been designed to support a modeler who is probably a domain expert but does not necessarily have high modeling expertise.
The first three steps cover the modeling and the revision of the business process. Only when the resulting process description is correct it is refined until it fits the requirements of a workflow specification. The intermediate step "Transformation into WF-nets" was introduced to provide a formal basis for the application of correctness criteria. Step four and five resolve efficiency aspects. Here, an execution strategy is determined and implemented.

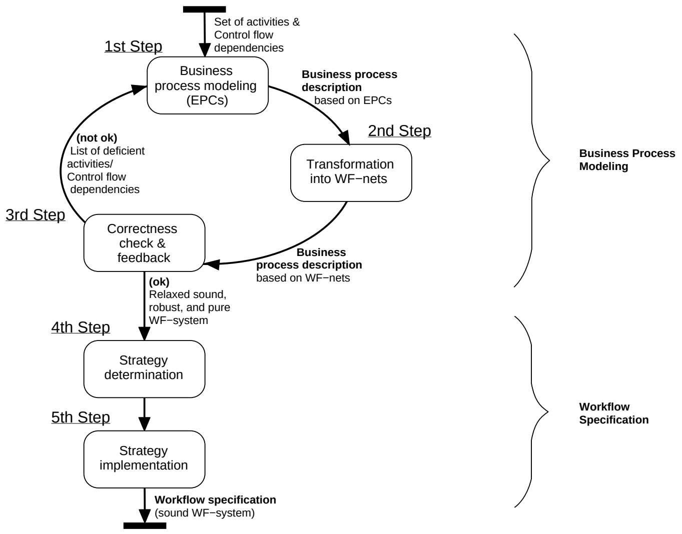

Fig. 1. A process model for workflow modeling.

The key of the process model shown in Figure 1 is the introduction of a pragmatic correctness notion which is gradually refined. It is clear that the final workflow specification must be *sound*.[15] This is a necessary requirement in order to guarantee reliable process execution at run-time. Still, such a strong correctness criterion restricts the modeling capabilities at the level of business process descriptions. Therefore, alleviated correctness criteria are introduced, namely *relaxed soundness* and *robustness*. These two criteria, which describe a subset of soundness, provide an adequate correctness understanding within this first abstraction level and make it possible to provide suitable feedback to the modeler.

Within the described process model, relaxed sound and robust process description are transformed into a sound specification. This is done in step five "Strategy implementation".

1.3. Structure

The remainder of this paper mainly follows the order of the five proposed steps (cf. Figure 1). In the first three sections we focus on the modeling, analysis, and revision of business processes. We present a new method appraising the quality of the derived process descriptions, providing suitable feedback for the modeler. The method incorporates modeling with EPCs (cf. Section 1), mapping EPCs to Petri nets (cf. Section 1) and checking the correctness of derived WF-nets using the new correctness criteria, relaxed soundness and robustness (cf. Section 1). Section 1 focuses on the implementation of a strategy. Applying existing results from Petri net synthesis, allows for the generation of a sound process description on the basis of a relaxed sound and robust specification. The support for the identification and determination of a suitable scheduling strategy is done in Section 1. Finally, in Section 1 the main ideas are summarized. Throughout the paper the proposed procedure is applied to a running example (an order handling process).

2. Modeling Business Processes

In this section, the basis for the process model shown in Figure 1 is established. In the first step the business process is modeled from a user perspective. We assume the person carrying out the modeling to be a domain expert with little modeling expertise. A graphical but only semi-formal modeling language is used.

As a typical representative of such a language, we use Event-driven Process Chains (EPCs) of the Architecture of Integrated Information Systems (ARIS).[22] The use of EPCs is not essential. Any other semi-formal modeling technique could be used equally well. EPCs were chosen as they are widely accepted in practice. This is based on their use to describe the SAP reference models[21] and their comprehensive tool support through the ARIS tool set. EPCs leave room for interpretation and hence ambiguities. An informal, i.e. potentially ambiguous, process description may be desired in the beginning, where the main focus is on communication.

2.1. EPC Syntax

The language of EPCs provides the user with a set of graphical notation elements for the representation of (business) functions, events and routing constructs to describe the control-flow. Functions are used to model the dynamic part of the process, and correspond to steps in the process. Sometimes we will also refer to functions as tasks or activities.

Another constructive element is the event. An event either triggers a function or marks the termination of it. Figure 2 shows an EPC containing 9 functions and 11 events. The event `new order` triggers the function `check_credit` whereas the event `order finished` marks the termination of the function `archive`. Furthermore, to describe more complex behavior such as sequential, conditional, parallel, and iterative routing, connectors are introduced. These fall into two categories: splits and joins. In both categories we have AND, XOR and OR connectors.

These elements can be combined in a fairly free manner, including cycles. The composition is only restricted by the following rules:

- There is at least one start and one end event.
- Events and functions have exactly one incoming and one outgoing arc (except start and end events).
- For every two elements there is a path between the two (ignoring the direction of arcs).
- An event is always followed by a function and vice versa (ignoring connectors).

The result of the modeling with EPCs is a process description. It should be read as a pattern describing a set of accepted process executions. Deficient executions are only described implicitly, as the set of executions which do not fit the described pattern.

Fig. 2 shows an EPC modeling the process "Handling of incoming order". It represents a reduced version of a real life process of a telephone company. The process models the ordering of a mobile phone which involves two departments: the accounting department handling the payment and the sales department handling the distribution.

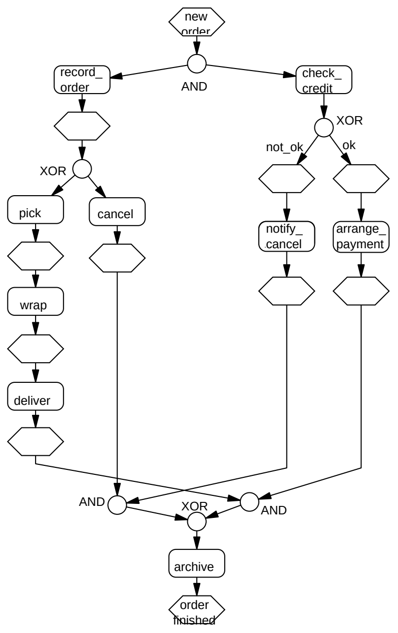

Figure 2: Handling of incoming order.

The process starts with the event `new order`. After that the execution is split into two parallel threads (AND split), the right one models accounting activities, whereas the left one models sales activities. In accounting the customer's creditworthiness is checked first (`check_credit`). The result of this task is either `ok` or `not_ok`. In case the result is positive the payment is arranged (`arrange_payment`), in the latter case the instance is canceled and the customer is notified (`notify_cancel`).

The left path models the tasks on the sales side. After performing task `record_order`, the order is either handled executing tasks `pick`, `wrap` and `deliver`, or `cancel`.

The two AND-connectors at the end make sure that only executions are accepted where both sides, the accountancy and the sales department, either cancel the instance or proceed the order. The process "Handling of incoming order" is completed by archiving information on that instance (`archive`).

2.2. EPC Semantics

EPCs are a semi-formal method of business process modeling. Although they have been applied quite successfully, their authors defined neither a comprehensive and consistent syntax nor the corresponding semantics.[23] With their widespread use, the need for a formal foundation increased. Several approaches propose a formalization. Most of them suggest a mapping on existing techniques, such as Petri nets[10,8,9] or Statecharts[13]. This way it becomes possible to use existing analysis and verification techniques. Other approaches to formalization have been presented.[11,12,24] Here, semantics are defined by means of a transition system. All the approaches have one thing in common: they assign operational semantics (execution semantics) to the EPCs.

This brings us to the main difference between existing approaches and the one proposed here, where non-operational semantics is assigned with EPCs. Supporting non-operational semantics we take the line of reasoning followed in the first publications about EPCs[b].[23,22]

We are confident that the non-operational interpretation of EPCs fits an intuitive modeling understanding and is one reason why EPCs are said to be easy to learn and to understand. Modeling business processes, the modeler normally starts by describing what an execution "should look like", hence describes a set of accepted/good executions. This procedure fits the non-operational semantics. An EPC is interpreted as a pattern that captures accepted executions. Deficient executions are only described implicitly.

Assuming operational semantics, the process description would also describe "how" an execution is reached. This means that not only accepted executions are considered but all possible behavior. Approaches assigning operational semantics with EPC were forced to restrict the modeling facilities of EPCs. In order to exclude

[b]In later publications the syntax of EPCs was enhanced by what is called a *process folder*. Process folders resemble tokens in a Petri net indicating the current state.

faulty behavior, the modeler is required to model in a well-structured way[c] and to avoid ambiguous concepts, such as the OR-connector.

> The EPC given in Figure 2 only describes executions where the two departments work together correctly: they either both accept the order (`AND_accept`) or both reject it (`AND_cancel`). Any other synchronization, e.g. one where the task `notify_cancel` is executed at the accounting side and `pick, wrap` and `deliver` on the sales side is not described by this pattern. As this covers reasonable behavior, the EPC should be considered correct.

The EPC does not determine "how" the accepted executions are achieved. It does not stipulate the order of the two possible choices. So, the EPC from Figure 2 accepts executions where the two departments work in parallel as well as executions where the two departments work sequentially. In an early design phase, this abstraction is beneficial, as it relieves the designer from thinking about efficiency aspects of the execution for the time being.

3. Transformation into Petri Nets

To investigate the correctness of the process description, we formalize EPCs by a mapping onto Petri nets. Petri nets are used as they have a clear and precise definition and a graphical notation similar to that of EPCs. In addition, they provide many existing analysis techniques and tools. As we do not restrict the modeling facilities of EPCs - the ambiguities are deliberately maintained - the resulting Petri nets will have faulty executions.

Using Petri nets we refer to the class of Place/Transition nets and more in particular to Workflow nets (WF-nets).[15,25] WF-nets were tuned to fit the requirements within the domain of workflow management. Petri net theory was exploited to assemble adequate properties and efficient algorithms for that Petri net class.[26,25] We briefly introduce some basic Petri net notions used in the paper.

3.1. Classical Petri Net

The classical Petri net is a directed bipartite graph. The two sorts of nodes are called places and transitions. Places are represented by circles, and transitions by boxes.

Definition 1 (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)[d].

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 f 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.

[c]In a well-structured EPC every split is complemented by a corresponding join.
[d]Unless stated otherwise we always refer to ordinary Petri nets with arc weights equal to one.

A place can contain zero or more *tokens*. Tokens are represented by black dots. The *state* of a Petri net, often referred to as *marking*, is the distribution of tokens over places, i.e. $M : P \longrightarrow I\!N$, assigning to every place the number of tokens $M(p)$ that reside in p. A place p is marked at a marking M iff $M(p) > 0$. To compare markings we define a partial ordering. For any two markings M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P : M_1(p) \leq M_2(p)$.

A marking M changes by firing a transition t. A transition t may fire only if it is *enabled*. A transition t is *enabled* in marking M, written $M \xrightarrow{t}$ iff every input place of t contains at least one token. If a transition t is enabled in marking M, it may fire: one token is removed from every input place and one token is added to every output place. Given a Petri net (P, T, F) and a marking M, we have the following notations:

- $M \xrightarrow{t} M'$: transition t is enabled in M and firing transition t in M results in M'
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1t_2t_3...t_{n-1}$ leads from marking M_1 to marking M_n, i.e. there are markings $M_2, M_3...M_{n-1}$, such that $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} M_3 \xrightarrow{t_3} ...M_{n-1} \xrightarrow{t_{n-1}} M_n$

A marking M_n is *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence $\sigma = t_1t_2...t_n$ such that $M_1 \xrightarrow{\sigma} M_n$. Note, the empty sequence ϵ is enabled at any marking M and satisfies $M \xrightarrow{\epsilon} M$. The set of markings reachable from a marking M is denoted as $R_{PN}(M)$: $R_{PN}(M) = \{M' | M \xrightarrow{*} M'\}$.

A pair (PN, M_i) of a Petri net PN and an initial marking M_i is called a net system (or just a system).

The behavior of a net system can be described via a labeled transition system. A labeled transition system is a directed graph with nodes representing states and edges representing state transitions. The edges of the graph are labeled. The label denotes what happens when the action represented by the edge is taken. Most transition systems have a distinguished node, indicating the initial state.

Definition 2 (Transition system). A Transition System (TS) is a quadruple $TS = (V, L, E, v_{in})$, where V is a non-empty set of states, L is a set of labels, $E \subseteq V \times L \times V$ is a transition relation, and $v_{in} \in V$ is an initial state. A transition system is finite if V and L are finite.

The reachability graph of a system $S = (PN, M_i)$ is the transition system where the states are the reachable markings, the labels are the transitions, the distinguished initial state is the initial marking and the labeled edges are all triples (M, t, M') such that M, M' are reachable markings satisfying $M \xrightarrow{t} M'$.

Definition 3 (Reachability graph). For the system $S = (PN, M_i)$ with $PN = (P, T, F)$, the reachability graph $RG_S = (V, L, E, v_{in})$ is a transitions system with: $V = R_{PN}(M_i)$ as set of states, $L = T$ and $E = \{(M, t, M') | M, M' \in R_{PN}(M_i) \wedge t \in T \wedge M \xrightarrow{t} M'\}$ as set of labeled edges, and $v_{in} = M_i$ as initial state.

As the set of labels and the initial state are implicitly specified by the system $S = (PN, M_i)$ the reachability graph RG_S is determined by the set of reachable markings V together with the set of edges E. We therefore use the shortcut notation: $RG_S = (V, E)$. We will now define some properties for Petri nets.

Definition 4 (Path in PN). In the Petri net $PN = (P, T, F)$, a path from a node x_0 to a node x_n is a non-empty sequence $(x_0, \dots, x_n)$ such that $(x_i, x_{i+1}) \in F$ for $0 \leq i \leq n-1$.

Definition 5 (Strongly connected). A Petri net is strongly connected iff for every pair of nodes x and y, there is a path leading from x to y.

Definition 6 (Pure). A Petri net (P, T, F) is pure iff $(x, y) \in F$ implies $(y, x) \notin F$.

Definition 7 (Free-choice nets). A Petri net $PN = (P, T, F)$ is a free-choice net iff $\forall t, t' \in T : {}^\bullet t \cap {}^\bullet t' = \emptyset \vee {}^\bullet t = {}^\bullet t'$.

Definition 8 (Bounded, safe). A system (PN, M_i) is bounded iff for each place p there is a natural number n such that, for every reachable marking, the number of tokens in p is less than n. The system is safe iff for each place the maximum number of tokens does not exceed 1.

3.2. WF-nets

Workflow nets (WF-nets) have been used to apply Petri net theory to process modeling and analysis.[15] A WF-net is a Petri net which has a unique source place (i) and a unique sink place (o). This corresponds to the fact that any case[e] handled by the process description is created if it enters the WFMS and is deleted once it is completely handled by the WFMS.

In such a net, a task is modeled by a transition and intermediate states are modeled by places. A token in the source place i corresponds to a "fresh" case which needs to be handled, a token in the sink place o corresponds to a case that has been handled. The process state is defined by the marking. In addition, a WF-net requires all nodes (i.e. transitions and places) to be on some path from i to o. This ensures that every task (transition) or condition (place) contributes to the processing of cases.

Definition 9 (WF-net). A Petri net $PN = (P, T, F)$ is a WF-net, if:
(i) PN has two special places, i and o. Place i is the only source (${}^\bullet i = \emptyset$) and place o is the only sink ($o^\bullet = \emptyset$).
(ii) Let $t^* \notin T$. The short-circuited net $\overline{PN} = (P, T \cup \{t^*\}, F \cup \{(o, t^*), (t^*, i)\})$ is strongly connected.

3.3. Transforming EPCs into Workflow Nets

The transformation of EPCs into Petri nets uses three steps. First, the elements of the EPC are mapped onto Petri-net modules. In the second step, rules are provided to combine the different modules to form a complex process model. A third step

[e] An instance of a workflow specification is referred to as a "case".[15]

becomes necessary if the primary EPC had more than one start and/or end event. In that case, the derived WF-net must be supplemented by additional in- and output places to satisfy the WF-net syntax.

3.3.1. Step 1: Mapping EPC elements to Petri-net modules

During the first step every EPC element is mapped onto corresponding elements on the Petri net-side. The mapping is illustrated in Figure 3.

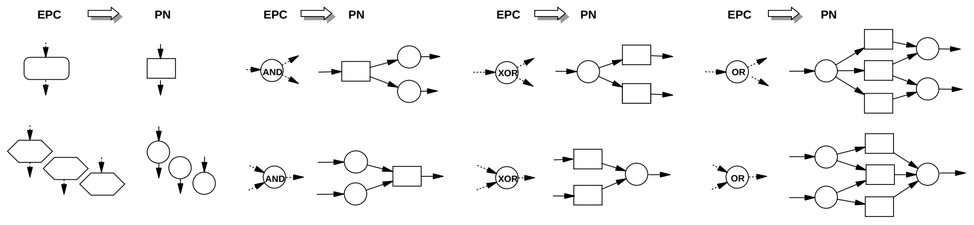

Figure 3: Step 1: Mapping EPC elements onto Petri-net modules.

Events and functions are transformed into places and transitions respectively including in- and outgoing arcs. Routing constructs such as `AND-split`, `AND-join`, `XOR-split`, `XOR-join`, `OR-split` and `OR-join` are mapped onto small Petri-net modules. The Petri-net modules describe the behavior of the routing constructs explicitly. This is particularly relevant for the OR, because its semantics has not been described consistently.

3.3.2. Step 2: Module combination

In the second step, the single Petri-net modules are joined to form a connected Petri net. Depending on the interface of the adjacent modules, one of the following combination rules is applied:

Case1: If input and output elements are of the same kind (e.g. both places) then the elements are unified.

Case2: If input and output elements are different (place and transition) then the arcs are fused.

Figure 4 illustrates the first and second step of the transformation of EPCs into Petri nets. The figure shows an EPC with an OR-join on the left and its Petri net-translation on the right side. The EPC as well as the Petri net have the semantics: `C` can be reached if either `A` or `B` or both occur[f]. In the EPC all these different cases are described through one connector. In the Petri-net module all possibilities are modeled explicitly via the transitions t_A, t_{AB} and t_B.

[f]If `A` and `B` occur one after the other `C` can also be reached twice.

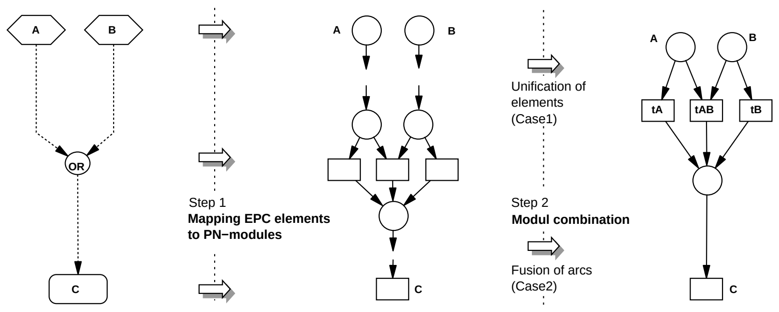
A
B
OR
C
Step 1
Mapping EPC elements to PN−modules
Unification of elements (Case1)
Step 2
Modul combination
Fusion of arcs (Case2)
tA
tAB
tB

Figure 4: Transformation of the OR-connector.

3.3.3. Step 3: Adding unique input/output places

Applying Step 1 and Step 2, an EPC is translated into a Petri net but not necessarily into a WF-net. If the EPC contained more than one start, and/or end event, the resulting net may have more than one start and/or sink place. There are no EPC syntax-rules that restrict the number of start and end events. Moreover, if there are several start events (or end events), it is not clear whether they are mutually exclusive or parallel. Therefore, a new start place and/or a new sink place is added. These new places are connected to the Petri net so that the places representing the primary start events (or end events) of the EPC are initialized (cleaned up). The connection of the new places to the primary places is not trivial and depends on the relation of the corresponding events in the EPC.

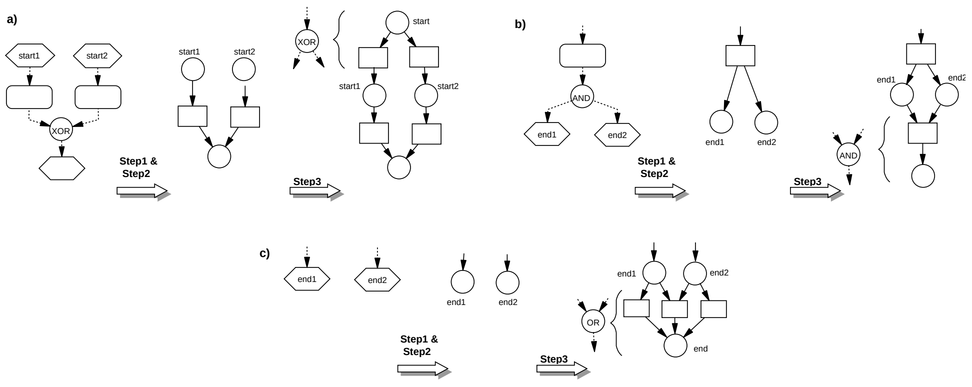

Figure 5: Step 3: Adding new start and sink places.

One way to determine the relation would be to track the paths, starting from the different start events (end events), until they join[g]. The connection of the new place

[g]The paths finally join. The EPC syntax rules state that: For every two elements there is a path

with the primary places would than be a Petri net module that corresponds to the connector complementing the one that was found.

Consider two start events that are connected via an XOR-join. They are treated as mutually exclusive. The two corresponding start places in the Petri net will be linked to the new inserted place by a complementing XOR-split. This was illustrated in part a) of Figure 5. Part b) gives an example connecting a new end place to existing end places. Here, an AND-join was inserted complementing the AND-split.

The general case may be more difficult. There could be more than only two different start events (end events) with paths possibly meeting in various connectors of different type. To avoid a lengthy procedure we propose to link different start places by a Petri net module corresponding to an OR-split and different end places by a Petri net module corresponding to an OR-join (see part c) of Figure 5). This way all possible dependencies are covered. Note that some of them may not reflect scenarios that are actually possible.

Applying steps 1 to 3, an EPC is transformed into a WF-net. The transformation is unique, in the sense that each EPC refers to only one WF-net. Drawing conclusions from the construction of the WF-net to its structure, it can mainly be said that it is pure. Depending on the primary EPC, the WF-net may contain cycles and choices that do not satisfy the free-choice property.

The proposed transformation approach is slightly more general than the transformations described in literature.[8,9] The rules presented here can also be applied to transform EPCs where connectors follow each other immediately, e.g., the AND- and XOR-split in the beginning of Figure 2. Another advantage of this approach is that the Petri net resulting from Step 2 does not contain any places or transitions not corresponding to elements of the EPC. The transformation rules described in literature[9,10,8] all contain rules which explicitly introduce new pseudo places and transitions to meet the Petri net syntax; the resulting Petri net may contain elements which have no counterpart in the application domain. In contrast to any other approach, a simple mechanism was provided to aggregate several start and end events which may be connected over different paths.

> The transformation was applied to the example from Figure 2. The derived WF-net is shown in Figure 6. For convenience, the Petri-net modules which correspond to the routing constructs of the EPC have been highlighted using dotted rectangles.

We emphasized that the choice of EPC's for modeling business processes is not essential. The approach can be adapted for other semi-formal modeling techniques as well. A further prominent example are the Activity Diagrams of the UML. Although, the new Release UML 2.0[27] envisages a Petri net interpretation for Activity Diagrams, the semantic is not formally founded. Many of the introduced concepts

between the two (ignoring the direction of arcs)

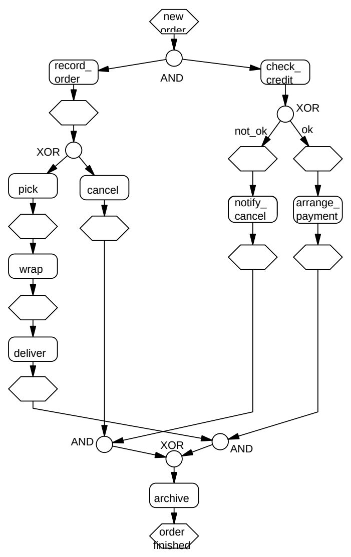

Figure 6: WF-net: "Handling an incoming order".

(e.g. *Final Flow Node, Interruption Zone, Exceptions*) do not allow an unambiguous Petri net interpretation. Another popular concept of Activity Diagrams are *Swimlanes*, which are used to distinguish task with respect to their organizational association. Using swim-lanes it is very unlikely that the process can be arranged in a well-structured way. This and the contained ambiguities again suggest to apply a weaker correctness property than soundness. Wendlandt[28] proposes rules to transform Activity Diagrams into WF-nets. According to the proposed methodology, ambiguities are maintained but made explicit. To support the modeler, an UML-profile was defined restricting the provided elements to these that have been considered useful for the domain of business process modeling based on Petri nets.

4. Analysis of the Derived WF-net

In the third step of the proposed procedure the modeled process is checked for some desired properties a process description should satisfy. The correctness check is done on the basis of the derived WF-net. Petri nets provide formally founded semantics and enable the use of existing tools. From the results, conclusions are drawn for the initial process description based on EPCs.
The most common property applied to WF-nets is soundness. We will argue that soundness although necessary for workflow specifications is not adequate to appraise the quality of business process descriptions.

4.1. Soundness

Van der Aalst first introduced soundness as a correctness criterion for WF-nets.[15]

It was argued that this criterion covers a minimum set of requirements which a process description should satisfy. A WF-net is sound if termination is guaranteed and there are no residual tokens, deadlocks or livelocks. Furthermore, there are no dead transitions. Residual tokens denote work that remained pending although the execution of the case had already terminated. Deadlocks indicate situations where the execution got stuck, and livelocks indicate situations where the process is unable to make any real progress. Dead transitions indicate tasks that do not contribute to the processing of workflow instances, as they are never executed. Next, we show soundness as defined in literature.[15] Note that there is an overloading of notation: the symbols i and o are used both as identifiers for the start and end places as well as to depict the markings where these places contain one token. The relevant meaning can be derived from the context.

Definition 10 (Soundness). A WF-system $S = (PN, i)$ is sound iff:

(i) For every state M reachable from state i, there is a firing sequence leading from state M to state o (option to complete). Formally: $\forall M : (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$.

(ii) State o is the only state reachable from state i with at least one token in place o (proper termination). Formally: $\forall M : (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$

(iii) There are no dead transitions in S. Formally: $\forall t \in T \exists M, M' : (i \xrightarrow{*} M \xrightarrow{t} M')$

We consider an EPC to be sound if the corresponding WF-net is sound. The soundness test was implemented within the Petri net-tool Woflan.[26] Woflan not only states whether the process description is sound or not, but also provides the modeler with further information in order to support the location of deficient parts of the WF-net.

The impact of soundness as the correctness measure for EPCs is substantial. Soundness imposes operational semantics. As a consequence, the modeler is required to think about the "how" of the execution, which involves the consideration of efficiency aspects. In the introduction it was argued that the specification of business processes should be as abstract as possible. The consideration of efficiency aspects requires detailed information about the process, e.g. duration and costs of tasks as well as the availability of resources. This information requires a much deeper insight and a higher level of detail than is available or desired for the modeling of business processes. Here, the focus is on communication. The objective is to come to a common process understanding between all participants clarifying the "what". All further information unnecessarily complicates the description.

Soundness can only be achieved through a restriction of the EPC modeling facilities. The requirement for soundness demands the modelers to restrict themselves to well-structured EPCs and to avoid ambiguous concepts, such as the OR-connector. These restrictions reduce the expressiveness of EPCs and furthermore impose higher requirements on the modeling knowledge of the domain experts.

The WF-net that resulted from the translation of the EPC "Handling an incoming order" was shown in Figure 6. The WF-system is not sound. An example for an unsound firing sequence is:

- `AND_split, record_order, pick, wrap, check_credit, not_ok, deliver, notify_cancel`.

Here, the case deadlocked having tokens in place `p9` and `p10`.

In order to receive a sound WF-net, the EPC specification needs to be changed in such a way that all executions of the corresponding WF-net terminate properly, i.e. residual tokens are avoided, as well as livelocks and deadlocks. To change the EPC "Handling an incoming order" accordingly, the EPC has to be re-arranged in a well-structured way. This change is not trivial and the feedback generated through the soundness check (e.g. by Woflan) only provides limited support for the redesign. Figure 7 shows a revised EPC and the corresponding, now sound, WF-net.

Still the impact of the changes is quite substantial. The revised EPC does not represent the distributed responsibility assignment between the two departments anymore. It furthermore fixes a certain scheduling strategy. Parallel execution of the sales and the accounting tasks has been excluded. In every possible execution the task `check_credit` is now performed before the delivery. Last but not least redundancy was introduced through the duplication of the function `record_order`.

To support a non-restricted modeling with EPCs, an adjusted correctness criterion, namely *relaxed soundness*, was introduced in our earlier work.[29] Relaxed soundness supports non-operational semantics and therefore allows the modeler to postpone decisions considering the efficiency of the process execution as long as possible, i.e. close to implementation. Furthermore, it does not restrict the EPC modeling facilities.

4.2. Relaxed Soundness

Relaxed soundness was intended to represent a more pragmatic view of correctness. It is weaker (in a formal sense) than soundness and therefore easier to accomplish. Relaxed soundness does not impose the need to avoid situations with residual tokens or livelocks/deadlocks. Therefore, it is suitable to check WF-nets which have been derived through the transformation of (not necessarily well-structured) EPCs containing OR-connectors. The idea behind relaxed soundness is that for each transition there is a sound firing sequence, i.e. a sequence that can be carried forward such that it terminates properly. We will define the term *sound firing sequence* to explain the differences between the criteria soundness and relaxed soundness in formal terms.

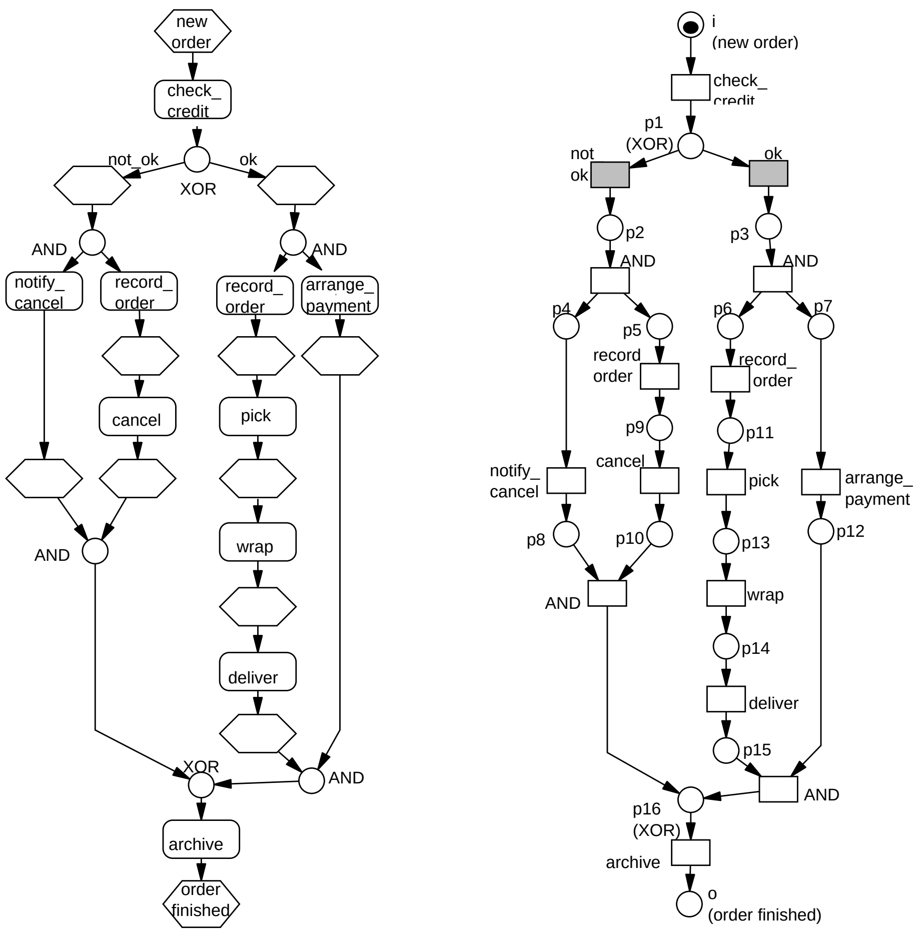

Figure 7: Revised EPC/WF-net "Handling of incoming order".

Definition 11 (Sound firing sequence). Let $S = (PN, i)$ be a WF-system. A firing sequence σ is sound if $i \xrightarrow{\sigma} M$ and $\exists\sigma'$,M$\xrightarrow{\sigma'} o$.

A sound firing sequence can be extended, such that marking o is reached. Correspondingly, an *unsound* firing sequence is a firing sequence which ends in a marking from which marking o is not reachable.

Whereas in a sound WF-net *all* firing sequences are sound, relaxed soundness only requires that there are so many sound firing sequences that each transition is contained in one of them. Note, that the classical notion of soundness subsumes relaxed soundness.

Definition 12 (Relaxed soundness). A workflow system $S = (PN, i)$ is relaxed sound iff each transition of PN is an element of some sound firing sequence. $\forall t \in T \quad \exists M, M' : (i \xrightarrow{*} M \xrightarrow{t} M' \xrightarrow{*} o)$.

Intuitively, relaxed soundness means that there are enough executions which terminate properly (i.e. state o was reached and there are no residual tokens) so that every transition is covered. A relaxed sound WF-net may have other firing sequences which do not terminate properly, but deadlock before termination or leave tokens in the net. In spite of that, relaxed soundness is still reasonable, because it requires that all relevant behavior is described correctly.

We argue that this criterion is closer to the intuition of the modeler. It does not force the modeler to think about all possible executions and then to care for proper termination.

4.2.1. Checking relaxed soundness

Using relaxed soundness as a correctness measure for EPCs, non-operational semantics are supported. The corresponding WF-net is checked only to allow for reasonable behavior. Interpreting relaxed soundness of a WF-net within the terms of the initial EPC, every function can be executed reaching a desired set of end events.

The relaxed soundness test was implemented within Petri net tools such as LoLA (Low Level Petri Net Analyzer)[30] and Woflan[26]. Parsing the reachability graph, they decide whether a given WF-system is relaxed sound or not. Both algorithms are based on a finite reachability graph. Therefore, the WF-system must have been checked for boundedness before. Cyclic paths within the resulting WF-net are allowed as far as they do not introduce unbounded behavior.

The results from the correctness check of the WF-net can be transferred directly to the initial EPC model. If the result of the relaxed soundness check is positive, we can conclude that the EPC represents reasonable behavior. If the result is negative, the modeler gets a list of transitions which are not contained in any sound firing sequence. According to the proposed transformation, every deficient transition either corresponds to a task or to a connector within the EPC. This means that either the task or one of the possible choices described by a connector is not included in an execution that terminates properly. It can be concluded that the corresponding part in the EPC needs improvement. In other words, as a general rule we have to consider transitions that are not contained in some sound firing sequence when we are looking for parts of the process that need revision.

This way, precise feedback is provided which will help the modeler to improve the primary process description until the corresponding WF-net fits the property.

Note, the feedback may become less precise if a transition at the beginning of the WF-net is not contained in any sound firing sequence. Then, all following transitions would be denoted as deficient as well. In such a case the modeler should start to check the failure prone EPC-elements in the order of occurrence within the failure message.

The process specification shown in Figure 6 is relaxed sound. The following two sound firing sequences contain all transitions:

- `AND_split, record_order, pick, wrap, check_credit, ok, deliver, arrange_payment, AND_accept, archive` and
- `AND_split, check_credit, not_ok, notify_cancel, record_order, cancel, AND_cancel, archive`.

Relaxed soundness states that all desired behavior has been covered. Still, it does not exclude faulty executions. Using the resulting process description as workflow specification, i.e. as basis for the process execution support at run-time, the faulty executions must be inhibited, i.e. the relaxed sound WF-net must be made sound.

Consequently, the question arises whether relaxed soundness provides a sufficient prerequisite for the generation of a sound specification. Regarding a workflow system as a stand-alone application the answer is positive[31], but workflow systems are reactive systems. They run in parallel with their environment, respond to inputs from the environment and produce output events which take effect back in the environment.

Starting from these assumptions, it turned out that relaxed soundness does not suffice as basis to generate a sound specification. A further correctness criterion is needed, indicating that the process can act robustly to (all) possible events coming from the environment. In earlier publications[31] it was shown that if the derived WF-net is relaxed sound and *non-controllable choice robust* a sound WF-net can be generated.

4.3. Non-controllable Choice Robustness

Regarding a workflow system as a reactive system, we require the resulting WF-system furthermore to be non-controllable choice robust (short: robust). This criterion, which was initially introduced by the first author[32], provides a means to describe robustness of a system against all possible requests from the environment.

4.3.1. Reflecting the interaction with the environment

The process description will be used to support the execution of a case at run-time. Then a workflow engine, also referred to as workflow controller, is used to schedule the case according to the rules specified in the process description. Operating on the process description, the workflow controller decides which enabled transition is executed and when. Still, the workflow controller may not enforce the firing of any type of transitions. Transitions that are beyond the control of a workflow engine are transitions that depict behavior of the environment. The interaction with the environment takes place via incoming external events or via the evaluation of external information. The reactive system has to respond to external events and needs to incorporate all possible outcomes of the information evaluation. An external event could be an incoming query, an acknowledgment from a customer, a message from another company, information from a business partner or just a timeout. Examples for the evaluation of external information are the question about available capacities, the check for credit-worthiness of a customer and the identity check of a co-operating partner.

Possible results of an information evaluation as well as the occurrence of external events should be reflected within the process specification. This is essential as the further execution differs with respect to various results or different external events. The influence of the environment was represented through a differentiation between transitions.[32] The set of transitions T of a WF-net $PN = (P, T, F)$ was split into disjoint sets of *controllable* and *non-controllable* transitions: $T = T_{NC} \cup T_{CON}$ and $T_{NC} \cap T_{CON} = \emptyset$. *Controllable* transitions (T_{CON}) model transitions whose executions are covered by the local workflow control. The firing of *non-controllable* transitions (T_{NC}), cannot be forced by the local workflow control but depend either on the evaluation of external data or on the kind of incoming event. *Controllable* transitions are denoted by white boxes, and *non-controllable* transitions are represented by gray boxes.

We assume non-controllable transitions to be free-choice and to not conflict with controllable transitions. This restricted modeling reflects the fact that the behavior of the environment cannot become disabled through the local control. In the reminder we will consider only WF-nets which satisfy these restrictions.

To define the criterion "non-controllable choice robust", we will look at our problem as a game between the workflow controller and the environment as an opponent who is trying to interfere with the process execution, so that an unsound firing sequence is generated. The question is whether the workflow controller can always win the game, i.e. react to the moves of the adversary and thus terminate properly.

A game is defined as a tuple $(\mathcal{G}, \phi)$ consisting of a game graph $\mathcal{G}$ and a temporal formula ϕ expressing the winning condition.[33] Adapted to the WF context it is natural to define the game graph on the basis of the reachability graph $RG = (V_{RG}, E_{RG})$ of a system $S = (PN, i)$. As the definition of a game graph requires a finite set of states, we only consider bounded systems, i.e. systems having a finite reachability graph. The moves of the different players are reflected through the labels at the state transitions. If the label t of a state transition is a controllable transition $t \in T_{CON}$ then a controller move is represented. If the state transition is labeled with a non-controllable transition $t \in T_{NC}$, it corresponds to a move by the environment. With regard to the labels, the state transitions are referred to as controllable or non-controllable state transitions.

A play on the game graph RG corresponds to a path ρ in RG starting in i (i.e. a firing sequence of the WF-net). The winner of a play is fixed by the winning condition ϕ. The first player wins a play ρ if ρ satisfies ϕ, while the opponent wins the game if the play satisfies $\neg\phi$. A strategy for a given game is a rule that tells a player how to choose between several possible actions in any game position. A strategy for the WF-controller is determined by a fragment of the game graph, having only controllable state transitions leaving the fragment.

Definition 13 (Strategy). Let $(\mathcal{G}, \phi)$ be a game. Let $\mathcal{G} = \mathcal{RG} = (\mathcal{V}, \mathcal{E})$ be the reachability graph of a WF-system $S = (PN, i)$. $SG \subseteq RG$, with $SG = (V_{SG}, E_{SG})$, $V_{SG} \subseteq V$, and $E_{SG} \subseteq E$, is a strategy if:

1. $i \in V_{SG}$
2. For each $M \in V_{SG}$ there is a directed path from i to M.
3. SG is self-contained with respect to the possible moves of the adversary: for all $M \in V_{SG}, t \in T_{NC}, M' \in V$: if $(M, t, M') \in E$ then $(M, t, M') \in E_{SG}$.

A strategy is a winning strategy[34] if the player[h] always wins no matter what the environment does. As wining condition we are interested in a basic condition[i] on determining *reachability*, namely "proper termination": $\phi = \Box\Diamond o$ - in LTL parlance.[35]

Definition 14 (Winning strategy). Let $(\mathcal{G}_{\mathcal{RG}}, \phi)$ be a game defined on the reachability graph RG of a system $S = (PN, i)$ with $\phi = \Box\Diamond o$. Let SG be a strategy for the WF-controller. SG is a winning strategy if it only contains paths that satisfy ϕ. Therefore, SG additionally meets the following requirements:

1. $o \in V_{SG}$
2. For each $M \in V_{SG}$ there is a directed path from M to o in SG.

This definition of the term strategy, which is usual in controller synthesis[33] is not very strong. It means that certain choices may never be presented to the environment. For our setting, a stronger understanding is necessary. A strategy should incorporate the requirement that all possible moves by the environment have to be covered at least once. This corresponds to the requirement that it should be possible to react to *all* possible moves of the environment.

Definition 15 (Complete strategy). Let $(\mathcal{G}_{\mathcal{RG}}, \phi)$ be a game defined on the reachability graph RG of a WF-system $S = (PN, i)$. Let SG be a strategy for the WF-controller. The strategy SG is called *complete* if all possible moves of the adversary are covered. Formally: $\forall t \in T_{NC} : \exists M, M' \in V_{SG}, (M, t, M') \in E_{SG}$.

Based on these notions, it is now possible to express non-controllable choice robustness of a WF-system.

Definition 16 (Non-controllable choice robustness). Let $(\mathcal{G}_{\mathcal{RG}}, \phi)$ be a game defined on the reachability graph RG of a WF-system $S = (PN, i)$ with $\phi = \Box\Diamond o$. The system S is non-controllable choice robust (short: robust) iff there exists a complete winning strategy SG for the workflow controller.

A WF-system is robust if there is a fragment of the reachability graph which starts in i, ends in o, contains at least one t-labeled state transition for any non-controllable transitions $t \in T_{NC}$, and has only controllable state transition leading out of the

[h] All terms are defined from the perspective of player "workflow controller".

[i] For a survey of possible winning conditions the reader is referred to the work of Thomas.[33]

fragment[j]. Assuming progress for non-controllable transitions, the existence of such a fragment guarantees that it is possible to reach state o (terminate properly) independent from the influence of the environment. While all non-controllable transitions (all possible moves of the adversary) are covered by the fragment, there is always a way to react and to terminate properly. Hence, if a WF-system is robust, the workflow controller can guarantee proper termination independently from all possible moves by the adversary. In earlier work[31] an algorithm checking robustness was provided. It decides whether a given WF-system $S = (PN, i)$ is robust and in the positive case returns the maximum and complete winning strategy $SG = (V_{SG}, E_{SG})$. If the algorithm aborts with the result "not robust", then there are non-controllable transitions which may inhibit proper termination. The deficient transitions are notified to the modeler who has to revise the corresponding elements within the initial specification. The algorithm mainly works as follows. It initially marks all states that potentially belong to the desired fragment and then progressively removes mistaken candidates. Potential states are all lying on a path from state i to state o. Illegal states are states from where non-controllable state transitions leave the fragment. The algorithm stops if the iteration of this procedure does not identify any illegal states any more.
For illustrating examples of non-robust and robust process descriptions as well as the precise algorithm the reader is referred to earlier work of the first author.[32,31]

4.3.2. Checking Robustness

A necessary prerequisite for the robustness check is the indication of non-controllable transitions in the WF-net. As EPCs do not provide equivalent concepts, this step needs the intervention of the modeler. Remember that moves of the environment either reflect incoming events or the outcome of decisions. We will investigate the EPC-syntax again to find pointers indicating the occurrence of either of the two.

Incoming events: EPCs have an imprecise event concept. Recall that an event either triggers a function or marks the termination of it. Therefore there is a distinction between the trigger-event and the supply-event.[22] Within the modeling, this distinction is blurred by the use of a simplifying event node (recall that events and functions are depicted as alternating nodes).

This simplification suggests that both events coincide. If this is not the case, the modeler often emphasizes the need for an extra input (e.g. to indicate the time distance between two functions), by introducing an extra event. This input event has one outgoing and no incoming arc, and therefore looks like a start event of the EPC. When trying to identify non-controllable transitions in the WF-net, these extra events can be used as pointer, as they indicate non-controllable transitions, i.e. moves of the environment triggered by an external event. This kind of pointer was illustrated in Figure 8 .

[j] Sometimes, we will refer to this fragment as "robust fragment"

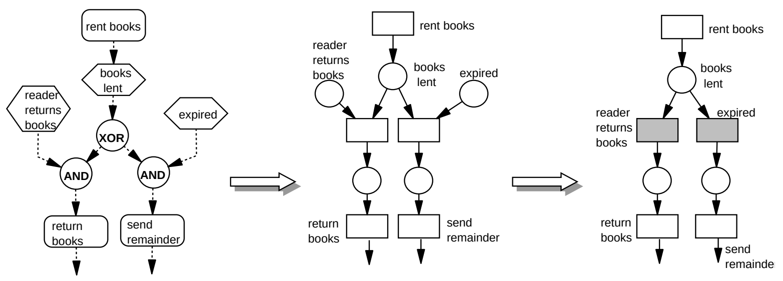

Figure 8: Pointer that can be used to discover non-controllable transitions

Outcome of decisions: Further non-controllable transitions are introduced in the WF-net if a decision based on external information was modeled in the EPC. Within EPCs, decisions are modeled with the help of an XOR-connector. A further hint to detect decisions based on external information is given through the inscriptions of the function preceding and the events following the XOR connector. Whereas the function is denoted with notions such as “test” or “check”, the succeeding events often refer to the decision criteria (e.g. $> amount$). For a pointer of this kind we refer to the inscriptions on the accounting side in the EPC “Handling an incoming order” (Figure 2). Note, that the XOR-connector on the sales side is not transformed into a non-controllable choice. The XOR-connector models the decision between the functions `pick` and `cancel`, which do not refer to any external event nor to the evaluation of external information.

A beneficial side of the identification of non-controllable transitions is the possible simplification of the EPC-PN transformation. The order of the transformation was the following: 1. Mapping EPC elements to Petri-net modules, 2. Module combination, and 3. Adding unique input/output places. The described step, identifying the non-controllable transitions, is inserted between steps two and three. As a consequence the places that must be merged in the last step may be reduced.

There are only two non-controllable transitions in the WF-net “Handling an incoming order” (Figure 2), namely `ok` and `not_ok`. These two reflect the outcome of a decision (task `check_credit`) and may hence not be controlled by the local workflow control. Applying the robustness algorithm to the reachability graph RG_{PN} a robust fragment was found. The reachability graph of the WF-system is depicted in Figure 9. The non-controllable state transitions are marked with a bow. The robust fragment $SG \subseteq RG$ was highlighted by showing the associated states and state transitions in bold.

The existence of the robust fragment SG states that, although the WF-system is not sound, it is possible to control the execution such that, only sound executions are chosen. Robustness states that this is possible independent of the moves of the environment (modeled by the non-controllable transitions).

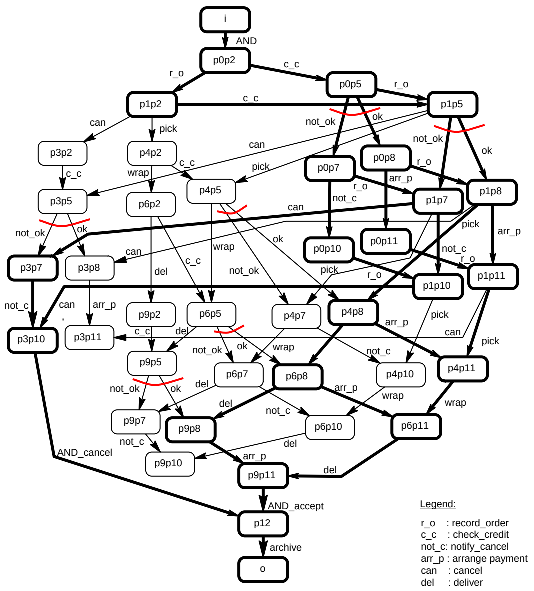

Figure 9: Reachability graph and robust fragment.

Once the process description satisfies relaxed soundness and robustness the business process modeling is completed. The derived process description reflects a set of accepted executions which can be enforced independent from the moves of the environment. As interface to the modeler an informal modeling technique was used. During the first phase this was appreciated as it promotes a common understanding between various participants. During the next phase the process descriptions shall be enhanced such that it can be used as workflow specification, i.e. as input for a WFMS supporting the execution of the business process at run-time. For this issue a formally founded modeling language is needed supporting operational semantics. Therefore, we focus on the Petri-net representation and enhance the corresponding WF-net description, which was so far only a byproduct.

5. Strategy Implementation

The modeler has so far been guided towards a relaxed sound and robust process specification. Relaxed soundness indicates the existence of enough sound firing sequences. Robustness indicates the existence of a strategy which guarantees sound execution independently from the moves of the environment. Still, the modeled process needs not be sound. There may be firing sequences that do not terminate properly.

Within the next steps (4 and 5 of the proposed procedure) a relaxed sound and robust specification is transformed into a sound specification. Therefore, it is necessary to restrict the set of all possible firing sequences to only sound ones.

The restricted set of firing sequences must not only be sound but must also belong to a winning strategy that can be enforced against possible interactions from the environment. This is guaranteed if the WF-system is robust. Therefore the goal is reached as soon as we find a Petri net with a reachability graph isomorphic to the robust fragment.

Applying methods from the area of Petri net synthesis [36,37,38,39,40,41] there are two possible approaches. The first simply applies the methods from Petri net synthesis generating a Petri net on the basis of the robust fragment. The result is a WF-net with a behavior isomorphic or bisimilar to the fragment. This approach is always applicable but has the disadvantage that the derived WF-net and the initial WF-net may differ considerably. For the second approach methods from Petri net synthesis were adapted for Petri net controller synthesis. Rather than constructing a new net, it is proposed to compute changes of the initial net so as to restrict its behavior to the one described by the robust fragment. This solution increases the possibility of recognizing the initial process description within the resulting one. Drawback here is that it can be applied only if the underlying transition system satisfies some further constraints. We will outline the two approaches in the next two subsections. The interested reader is referred to earlier work for a more detailed description.[31]

5.1. Petri Net Synthesis

The *synthesis problem* for Petri nets is tackled by deciding whether a given graph is isomorphic to the reachability graph of some Petri net. The synthesis problem was raised in 1990 by Ehrenfeucht and Rozenberg.[36] Subsequently, it was shown that an elementary net system can be synthesized on the basis of regions from a (sequential) transition system satisfying some separation conditions, namely from an elementary transition system.[37] Thereafter, the synthesis problem has been solved for other classes of Petri nets.[42,40,43,44] The solutions all use regions.[45] Regions are subsets of states, which may be interpreted as *atomic* nets, i.e. nets consisting of a single place together with its input and output transitions. A Petri net can be composed using atomic nets and joining them at common transitions.

Every transition system can be broken down into a finite number of subsets of regions. Fitting together the atomic nets which correspond to the regions, however, does not necessarily result in a Petri net that is isomorphic (in terms of behavior) to the initial transition system. The challenge was to decide constructively the existence of a Petri net with a reachability graph isomorphic to a given transition system.

Applying Petri net synthesis for the generation of sound WF-nets we start with the transition system defined through the robust fragment. Although the reachability graph satisfies the properties of an elementary transitions system, the robust fragment may not. In any case the fragment does satisfy the standard axioms of a transitions system: 1) no self loops, 2) every event has an occurrence, and 3) every state is reachable from the initial state. This can be concluded by construction of the WF-net and the definition of SG. If the fragment additionally satisfies the axioms of an elementary transition system, the basic algorithm introduced in can be applied.[37] The result is a WF-net with isomorphic behavior.

A more general synthesis algorithm was introduced in later publications.[39] Its application is not limited to elementary transition systems but covers the full class of transition system by means of transition splitting. The behavior of the resulting Petri net is not necessarily isomorphic to the initial transition system but bisimilar. The algorithm was implemented within the tool Petrify.[46] Applying this enhanced algorithm to any robust fragment a sound and safe WF-net with bisimilar behavior is generated.[31]

Although the derived WF-net is sound it may not always be adequate for our purpose. The behavior of the initial WF-system became restricted to sound firing sequences only. The changed behavior should be confirmed by domain experts before the corresponding workflow-specification is put to use. Therefore, the final process description should be discussed again between domain experts. This may cause difficulties as the derived WF-net probably bears little resemblance with the initial WF-net. This is due to a number of issues. First of all, generating a Petri net just on the basis of a transition system neglects all graphical information as well as place labels. Second, the algorithm generates a minimal saturated net, which means that implicit places in the initial net (perhaps introduced for a better structuring) are omitted. Third, synthesized nets are always safe. Therefore the derived WF-net may become quite large. This is because for a k-bounded place in the initial WF-net, k places would be synthesized in the resulting WF-net. Finally, in the resulting WF-net even the graphical ordering of transitions may have changed. Transitions that were primarily ordered with respect to their appropriate organizational unit are reordered with respect to their actual occurrence. The only guaranteed correspondence between initial and resulting WF-net is through transition labels.

The altered appearance of the resulting process description complicates the recognition of the initial modeled process. The domain experts have to understand a new process description in order to discuss and agree on the final behavior.

To remedy this problem an alternative procedure is proposed. It is based on the application of results of the synthesis of Petri nets controllers. This research field applies results from Petri net synthesis generating only these parts of a net which guarantee some constraints specified in advance. Rather than constructing a new net, it is proposed to compute changes of the initial net so as to restrict its behavior to the one described by the robust fragment. This solution has the advantage that domain experts can recognize the WF-net more easily, as only some changes have to be considered.

5.2. Petri Net Controller Synthesis

The objective of Petri net controller synthesis is to compute a set of new places for a given Petri net which supervise or control the behavior of the Petri net, avoiding entering a set of "forbidden states"[k]. The introduced places are called controller places[47] or monitors[48].

Contrary to Petri net synthesis, where a whole net is synthesized, in this application domain only some designated places, namely the controller places, have to be synthesized. Adding these places to the initial net, the behavior is restricted. As these places are not contained in the initial net there are no corresponding regions so far. The information needed for their computation can be gained in various ways, e.g. from place invariants[47], general mutual exclusion conditions (GMECs)[48], or sets of forbidden markings[41]. We will here look more closely at the approach based on sets of forbidden markings.[41]

Here, the information needed for the computation of the missing regions is derived from the state transitions transgressing the *legal behavior*. The legal behavior corresponds to the partial reachability graph from where all desired states remain reachable. In our case the legal behavior corresponds to the robust fragment SG containing only sound firing sequences. It is clear that state transitions leaving the legal behavior have to be prevented.

The algorithm, which is based on the reachability graph of a pure[l] and bounded WF-system works as follows. First, the set of forbidden state transitions is determined. It consists of all state transitions which leave the robust fragment. For every of these instances an equation system is established which is used to compute a controller place inhibiting this forbidden state transition. The equation system consists of three equations: the event separation condition - an equation which in terms of the incidence matrix describes the interdiction of the corresponding state transition, the *Marking equation lemma*, and the general property of T-invariants. The latter two should still hold as the introduction of new places need not change the legal behavior.

Note, the equation systems are solvable only if the underlying transition system fulfills the axioms of a general transition system. This is the major drawback of this approach as, so far, there is no result showing that the fragment always satisfies further axioms, and hence coincides with a special transition system. However, all relaxed sound and robust WF-systems investigated by the authors produced fragments even satisfying the axioms of elementary transition systems.

The equation systems of different instances may have common solutions. As a result, the number of controller places needed is generally much smaller than the number of forbidden state transitions. The set of controller places together with the associated flow-relations determine the synchronization pattern. The computation of the pattern was implemented within the tool Synet.[49] For a precise description

[k]An additional place can only restrict the behavior because the place can block transitions but it cannot enable transitions which are not enabled in the net without the place.
[l]Remember that the Petri nets derived through the EPC-PN transformation are pure by construction.

of the algorithm, its theoretical background and prerequisites for its application the reader is referred to the PhD thesis of the first author.[31]

The synchronization pattern is finally incorporated into the initial WF-system. As effect the forbidden state transitions become disabled, while the rest of the behavior stays the same. If all controllable transitions of the initial WF-net are

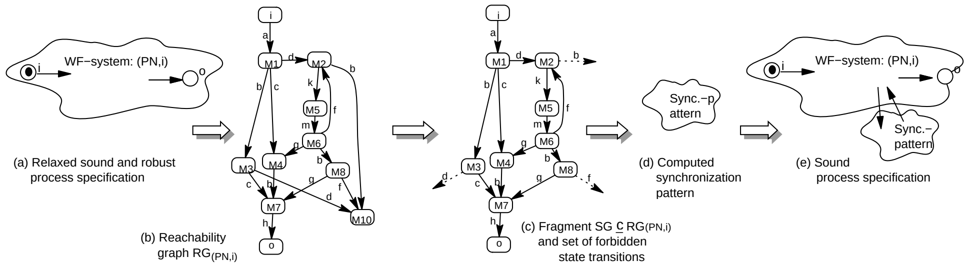
WF−system: (PN,i)
i
o
(a) Relaxed sound and robust process specification
(b) Reachability graph $RG_{(PN,i)}$
M1
M2
M3
M4
M5
M6
M7
M8
M10
(c) Fragment SG ⊆ $RG_{(PN,i)}$ and set of forbidden state transitions
Sync.−pattern
(d) Computed synchronization pattern
(e) Sound process specification

Figure 10: Applying controller synthesis for workflow modeling.

part of the robust fragment[m] - the derived workflow specification will satisfy the conditions that characterize a sound process description: *(1) option to complete*, *(2) proper termination*, and *(3) no dead transitions* (cf. Def. 10). Figure 10 illustrates the application of controller synthesis for workflow modeling.

The resulting Petri net again fulfills the properties of a WF-net. It is a strongly connected net-system having one source and one sink place. Note that in rare cases, arc inscriptions (i.e., weights) belonging to the newly inserted places are greater than 1.

We will apply the algorithm to our running example “Handling an incoming order”. For the computation of the synchronization pattern we refer to the robust fragment as highlighted in Figure 9.

Figure 11 shows the computed synchronization pattern as well as its integration into the process description from Figure 6. The resulting specification is sound. There are no firing sequences that deadlock or do not terminate properly. Using this process specification as basis for the workflow-controller a reliable process execution at run-time can be guaranteed.

Note, that the only further approach applying methods from Petri net Synthesis in process modeling was proposed by Agostini and De Michelis.[50] The authors propose to use both the Petri net and the corresponding reachability graph as interface to the modeler and use the basic algorithm[37] to transfer between both descriptions.

[m]Although very unlikely, there is a chance that the robust fragment does not contain all *controllable* transitions.[31] Synthesizing controller places, these transitions would become disabled. In the consequence the resulting WF-net has dead transitions and will therefore only satisfy the soundness conditions *(1)* and *(2)*.

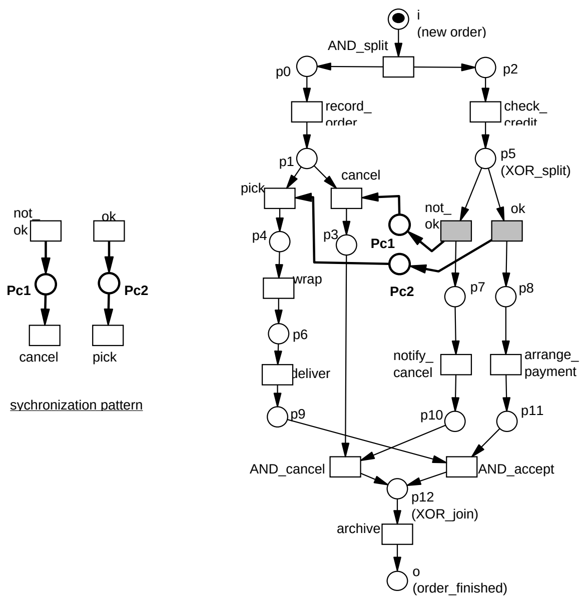

Figure 11: WF-net "Handling an Incoming order" with integrated synchronization pattern.

Adequate for their modeling approach is the Petri net class of *Elementary Net Systems*. In contrast to our approach all process models are assumed to be acyclic, free-choice and sound. Interaction with the environment is not represented.

6. Strategy Determination

Transforming a relaxed sound WF-system into a sound WF-system, the behavior is restricted to a subset of the sound firing sequences. The choice for a certain subset determines a strategy, which in turn determines the efficiency of the process execution. In general, strategies can be optimistic or pessimistic. The fragment computed through the robustness algorithm so far only determined pessimistic strategies. Pessimistic strategies wait for decisions to be taken in advance in order to avoid faulty situations. Following a pessimistic strategy, the process execution is made sequential. In contrast, optimistic strategies support parallel execution of depending threads but accept additional costs in some cases through the need for recovery.
In this section we will discuss alternative strategies and their implementation.

The decision for a certain strategy is based on expert knowledge or long term statistical evaluations. The selection and realization of a strategy should preferably be one of the last steps in the modeling of workflows, as corresponding information (the occurrence probability of a certain failure, costs of failure compensation, duration and cost of tasks, or priorities) will often only become available then, and may even change at run-time. Their late incorporation allows flexibility if priorities change. It will not be necessary to revise the whole procedure starting from new requirements, but modeling results from earlier phases may be reused.

We will consider again the example "Handling an incoming order". The WF-net in Figure 6 does not yet suggest a certain scheduling strategy. The two departments can work in parallel or sequentially. In Figure 11 a pessimistic strategy was implemented. In favor of avoiding deadlocks only sequentialized executions are supported. The customer check is always executed before the sales department may start the delivery process. In case both, the customer check and the delivery process take a long time, this would be very inefficient. This is especially undesirable if it is very rare that a customer check results in a *not_ok*. Therefore, this way of pessimistic scheduling would be annoying. It would be more efficient to just start the delivery of the order to the customer hoping the customer check will be *ok*, i.e. following an optimistic approach. Only in the rare case that the decision *not_ok* was taken, the order should be returned to stock and should be canceled after all.

Starting from a relaxed sound process description, we will now investigate how different scheduling strategies can be supported. Beside the implementation of a fixed strategy, the proposed procedure also facilitates the identification of useful strategies.

A relaxed sound process description determines a set of desired executions. Still, it does not describe "how" the desired executions are achieved. This decision is made by selecting a strategy. We have seen that a strategy corresponds to a special fragment of the reachability graph (cf. Def. 13). In the last section it was shown how a strategy was implemented, restricting the behavior of the relaxed sound WF-system to the corresponding fragment. If the implemented strategy was complete and winning, the WF-controller could, by following the prescribed rules, guarantee a sound process execution at run-time independent from the moves of the environment. If the initial relaxed sound WF-system is robust, there is a complete winning strategy, i.e. a fragment of the reachability graph which satisfies the corresponding requirements. Still, there may be sound firing sequences in the initial WF-system which are not supported by any robust fragment. These executions, although sound, would not be supported if the corresponding strategy becomes implemented. The problem with these executions is that proper termination cannot be guaranteed because if the environment interferes the system may end in a deadlock.

Looking again at our running example, the firing sequence

- `AND_split, record_order, pick, wrap, check_credit, ok, deliver, arrange_payment, archive`.

is sound but not contained in the robust fragment and hence not supported by the sound WF-net shown in Figure 11.

If the domain experts consider these executions to be considerably more efficient than the ones covered, another strategy must be found. The new strategy should cover these sound executions. It does not suffice to merely combine the desired executions. The corresponding fragment will not satisfy the properties of a strategy. It is not self-contained with respect to non-controllable transitions, cf. Def 13. There are non-controllable transitions that lead from the fragment to states that indicate a deadlock.

In order to support the desired set of sound executions, the fragment must be enhanced. New behavior must be incorporated that makes it possible to recover from deadlocks. This is achieved by adding tasks to the process description which *compensate* the results of previous tasks. For their specification further information must be compiled, regarding:

- the states from which compensation is possible,
- compensating tasks, and
- states to which the process is rolled back after compensation.

The specification of the compensating tasks cannot be automated but must be done by domain experts. The knowledge for the recovery behavior is based on the application context in combination with efficiency considerations and cannot be determined by a predefined set of rules.

Adding transitions to recover from bad markings (such as deadlocks, livelock or markings with residual tokens) the resulting WF-net should be again relaxed sound and robust. Support for this process revision can be gained again through steps 1 to 3 of the proposed process model.
Once the recovery behavior has been added successfully, the robust fragment is computed on the basis of the enhanced WF-system. It now contains the desired firing sequences. It also contains some new sound firing sequences which enable recovery if a (former) deadlock is reached.

In the next step the strategy that corresponds to the derived robust fragment, is implemented. Again, this is done applying either of the synthesis methods described in Section 1. The result is a sound WF-system. We will illustrate the implementation of an optimistic strategy by means of our running example "Handling an incoming order". The process description is adapted such that, the desired parallel executions are supported as well.

For the example, we assume that all tasks within the sales department that occur before the delivery can be reset without extraordinary charges. This affects tasks `pick` and `wrap`. Corresponding compensation tasks are `return` and `unwrap`. After the item has been returned to stock, the instance should be `canceled`. Task `deliver` is considered to be non-reversible. The enhanced EPC, incorporating the recovery behavior, as well as the corresponding WF-net are shown in Figure 12. Notice that the integrated tasks only show one possible way of modeling the recovery behavior.

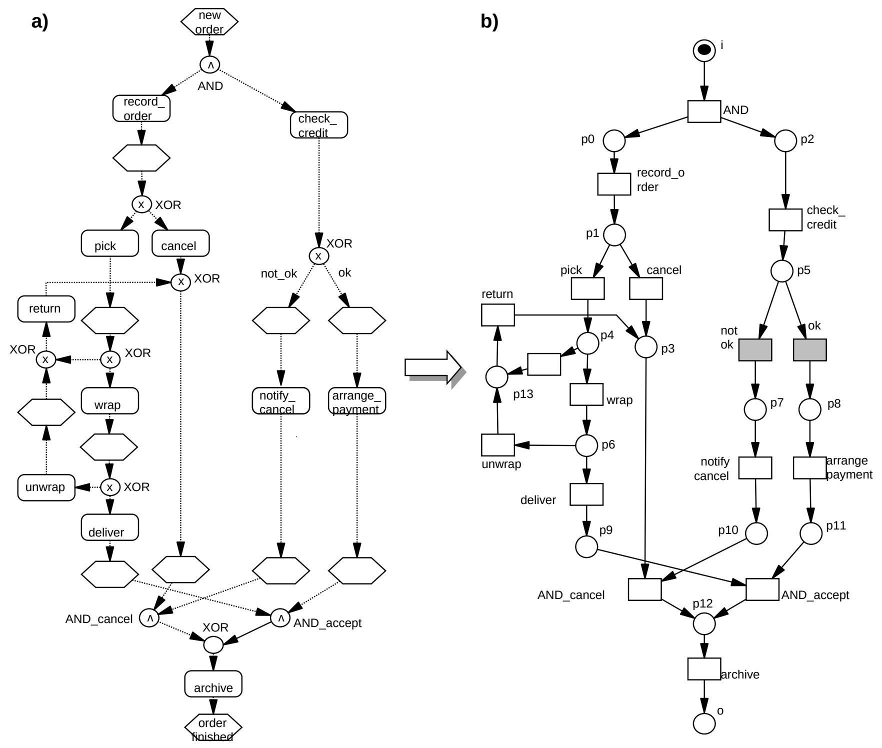

Figure 12: Process descriptions incorporating possible recovery behavior (a) EPC (b) WF-net.

The resulting WF-system is again relaxed sound and robust. The robust fragment $SG \subseteq RG$, is shown in Figure 13(a). All sound firing sequences of the initial, relaxed sound WF-system (cf. Figure 6) are maintained. Furthermore, some additional, less efficient executions are accepted too. Implementing the derived synchronization pattern, cf. Figure 13(b), results in the sound WF-system shown in Figure 13(c).

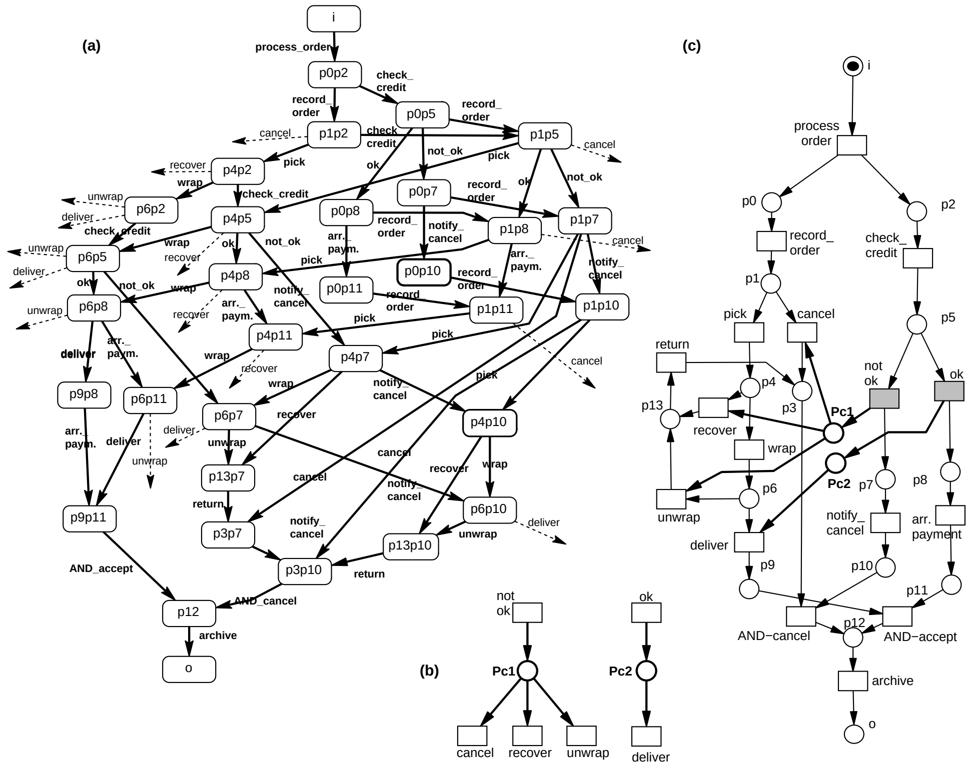

Figure 13: (a) Fragment $SG \subset RG$ with leaving state transitions, (b) synchronization pattern, and (c) sound WF-system.

7. Summary and Conclusion

In this paper we proposed a rigorous approach for modeling functional workflow requirements. The process model shown in Figure 1 supports and guides the modeler from a semi-formal description of business processes towards sound workflow specifications, thus helping to bridge the gap between business process modeling and workflow specification.

The proposed process model is based on a combination of different modeling languages. A semi-formal modeling language is used as interface to the domain expert. As a prominent example, widely accepted in practice, we used Event-driven Process Chains (EPCs).

For workflow specification we have used Petri nets, in particular WF-nets. The strength of Petri-nets lies in their formal foundation which has resulted in theoretical results, analysis techniques, and tools.

The proposed approach acknowledges the need to describe business processes at different levels of abstraction and combines the advantages of different modeling languages that proved to fit the respective requirements.

It is clear that such a cross-language process model must direct particular attention to a smooth transformation between the techniques used. This was achieved by providing a set of rules transforming the semi-formal process descriptions based on EPCs into WF-nets. The proposed transformation does not restrict the modeling facilities of the primary technique and maintains the various interpretations, making them explicit.

The key concept for the proposed process model is the use of pragmatic correctness criteria, namely *relaxed soundness* and *robustness*. These two criteria fit the correctness requirements within this first abstraction level and make it possible to provide precise feedback to the modeler.

Relaxed soundness guarantees that some reasonable behavior is covered. It does not exclude the existence of deficient executions. Robustness focuses on the interaction between a WF-system and its environment, i.e., unlike most other approaches the WF-system is considered to be a reactive system. It assures that proper termination of the process is always possible despite of the moves of the environment.

The resulting process description cannot yet be used as a basis for the execution support. It may still contain undesired executions. These undesired executions are eliminated in the last steps of the proposed procedure. Here the relaxed sound and robust process description is refined towards a sound WF-net. Applying methods from Petri net synthesis, this refinement step can be done more or less automatically.

A strategy is implemented by restricting the behavior of the described process to only sound executions. The choice for a strategy (either optimistic or pessimistic) determines the efficiency of the process execution. Selecting a strategy as late as possible has several advantages: It facilitates an intuitive modeling by relieving the modeler from thinking about efficiency aspects already at design time. This is especially desirable as corresponding information (the occurrence probability of a certain failure, costs of failure compensation, or priorities) will often become available (and may even change) only at run-time. Their late incorporation therefore extends the possibility to reuse modeling results under changing priorities.

The resulting process description defines the tasks involved and determines their order. The specification is sound. Using the process description as a basis for the execution support during run-time reliable processing can be guaranteed.

In this paper, we did not discuss complexity results for the approach shown in Figure 1. From a computational complexity point of view, the first two steps are easy. The third step, however, requires the calculation of relaxed soundness and robustness. In our current approach this requires the construction of the coverability graph. The theoretical worst-case complexity of generating the coverability graph is non-primitive recursive space.[51] Similarly, the fourth step and the required feedback loops (e.g., adding compensation) may be intractable for large process models.

Future work will aim at improving the current algorithms (e.g., improving efficiency) and fine-tuning the overall approach. Even more important will be the application of the results presented in this paper. Thus far, we only applied the approach to relatively small examples. To apply the approach in practice we need to map the resulting workflow specification in terms of WF-nets onto a concrete workflow management system. For some systems this may be trivial (apart from adding information on the organizational, data, and application perspectives), for others this may be much more complicated.[52]

8. References

1. Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 1999.
2. W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.
3. BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, Siebel Systems. *Business Process Execution Language for Web Services (Version 1.1*, 2003.
4. R. Eshuis and R. Wieringa. A Real-Time Execution Semantics for UML Activity Diagrams. In H. Hussmann, editor, *4th Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2001)*, volume 2029 of *LNCS*, pages 76–90. Springer, 2001.
5. R. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modeling*. PhD thesis, University of Twente, NL, 2002.
6. D. Wodtke and G. Weikum. A Formal Foundation For Distributed Workflow Execution Based on State Charts. In *Int. Conf. on Database Theory*, 1997.
7. R. Eshuis and R. Wieringa. Requirements Level Semantics for UML Statecharts. In S. F. Smith and C. L. Talcott, editors, *Formal Methods for Open Object-Based Distributed Systems IV - Proc. FMOODS'2000, September, 2000, Stanford, California, USA*. Kluwer Academic Publishers, 2000.
8. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.
9. D. Moldt and J. Rodenhagen. Ereignisgesteuerte Prozessketten und Petrinetze zur Modellierung von Workflows. In *Visuelle Verhaltensmodellierung verteilter und*

nebenläufiger Software-Systeme, volume 24/00-I of *Fachberichte Informatik*, pages 57–63, 2000.

10. P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event-driven Process Chains. In J. Desel and M. Silva, editors, *Application and Theory of Petri nets*, volume 1420 of *LNCS*, pages 286–305. Springer, Berlin, 1998.
11. M. Nüttgens and F. J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In J. Desel and M. Weske, editors, *Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, LN in Informatics. GI-Edition, 2002.
12. F. J. Rump. *Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten. Formalisierung, Analyse und Ausführung von EPKs.* Teubner, Stuttgart, 1999.
13. P. Muth, Wodtke D., J. Weissenfels, A. Kotz Dittrich, and G Weikum. Enterprise-wide workflow management based on state and activity charts. In A. Dogac, L. Kalinichenko, T. Özsu, and A. Sheth, editors, *Advances in Workflow Management Systems and Interoperability*, LNCS. Springer-Verlag, 1998.
14. C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, TU Darmstadt, Darmstadt, 1962.
15. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
16. A. Agostini and G. de Michelis. A Light Workflow Management System Using Simple Process Models. *Computer Supported Cooperative Work*, 9(3/4):335–363, 2000.
17. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. Accepted for publication in *Information Systems*, and also available as QUT Technical report, FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2003.
18. N.R. Adam, V. Atluri, and W-K. Huang. Modeling and Analysis of Workflows Using Petri Net. *Journal of Intelligent Information System, Special Issue on Workflow and Process Management*, 10(2):131–158, 1998.
19. Software-Ley. *COSA 3.0 User Manual.* Software-Ley GmbH, Pullheim, Germany, 1999.
20. Get Process AG, Oberwil, Switzerland. *INCOME Suite.*
21. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation.* Addison-Wesley, Reading MA, 1998.
22. A. W. Scheer. *Business Process Engineering, ARIS-Navigator for Reference Models for Industrial Enterprises.* Springer, 1994.
23. G. Keller, M. Nüttgens, and A. W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, University of Saarland, Saarbrücken, 1992.
24. W.M.P. van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80, Trier, Germany, November 2002. Gesellschaft für Informatik, Bonn.
25. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net based Techniques. In *Business Process Mangement: Models, techniques, and Empirical Studie*, volume 1806 of *LNCS*, pages 161–183. Springer Verlag, 2000.
26. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
27. Unified Modeling Language: Superstructure (version 2.0 Final Adopted Specification), 2003.

28. D. Wendlandt. Workflow-Management mit UML-Aktivitätsdiagrammen und deren Verifizierung. Master's thesis, Technical University Berlin, 2004.
29. J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.L. Dittrich, A. Geppert, and M.C. Norrie, editors, *Advanced Information System Engineering, CAISE 2001*, volume 2068 of *LNCS*, pages 157–170. Springer, 2001.
30. K. Schmidt. LoLA: A Low Level Analyser. In *Proc. Int. Conf. Application and Theory of Petri net*, volume 1825 of *LNCS*, pages 465–474, 1999.
31. J. Dehnert. *A Methodology for Workflow Modeling - From business process modeling towards sound workflow specification.* PhD thesis, TU Berlin, 2003.
32. J. Dehnert. Non-controllable choice robustness: Expressing the controllability of workflow processes. In J. Esparza and C. Lakos, editors, *23rd Int. Conf. on Application and Theory of Petri Nets*, volume 2360 of *LNCS*, pages 121–141. Springer, 2002.
33. W. Thomas. On the synthesis of strategies in infinite games. In *Symposium on Theoretical Aspects of Computer Science*, pages 1–13, 1995.
34. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL '89. Proceedings of the sixteenth annual ACM symposium on Principles of programming languages*, pages 179–190, New York, 1989. ACM Press.
35. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer, New York, 1992.
36. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures. Part II: State Spaces of Concurrent Systems. *Acta Informatica*, 27(4):343–368, 1990.
37. M. Nielsen, G. Rozenberg, and P. S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96(1):3–33, April 1992.
38. J. Desel and W. Reisig. The Synthesis Problem of Petri Nets. *Acta Informatica*, 33(4):297–315, 1996.
39. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, 1998.
40. E. Badouel, P. Darondeau, and B. Caillaud. Distributing Finite Automata through Petri Net Synthesis. *Formal Aspects of Computing*, 13:447–470, 2002.
41. A. Ghaffari, N. Rezg, and X.Xie. Live and Maximally Permissive Controller Synthesis Using the Theory of Regions. In B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie, editors, *Synthesis and Control of Discrete Event System*, pages 155–166. Kluwer Academic Press, 2002.
42. M. Mukund. Petri Nets and Step Transition Systems. *International Journal of Foundations of Computer Science*, 3(4):443–478, 1992.
43. E. Badouel, L. Bernadinello, and P. Darondeau. Polynomial algorithms for the synthesis of bounded nets. In *Caap'95*, volume 915 of *LNCS*, pages 647–679, 1995.
44. L. Bernardinello, D. De Michelis, K. Petruni, and S. Vigna. On synchronic structure of transition systems. In J. Desel, editor, *Structures in Concurrency Theory*, pages 11–31. Springer, 1996.
45. E. Badouel and P. Darondeau. Theory of regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Model*, volume 1491 of *LNCS*, pages 529–586. Springer, 1998.
46. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997. download from http://www.lsi.upc.es/$\tilde{}$jordic/petrify/distrib/.
47. K. Yamalidou, J. Moody, M. Lemmon, and P. Antsakli. Feedback control of Petri nets based on place invariants. *Automatica*, 32(1):15–28, 1996.
48. A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion constraints on nets

with uncontrollable transitions. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 974–979, Chicago, IL, 1992.

49. B. Caillaud. Synet : A Tool for the Synthesis of Bounded Petri-Nets, Applications. Technical Report RR-3155, Inria, Institut National de Recherche en Informatique et en Automatique, 1997.
50. A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. In *Business process Management*, volume 1806, pages 218–233. Springer, 2000.
51. J. Esparza and M. Nielsen. Decidability Issues for Petri Nets: A Survey. *Journal of Information Processing and Cybernetics*, 30:143–160, 1994.
52. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.