From Business Process Fragments to Workflow Definitions

Carlo Simon
Universität Koblenz-Landau
D-56070 Koblenz
Universitätsstraße 1
simon@uni-koblenz.de

Juliane Dehnert
Fraunhofer ISST, Berlin
D-10178 Berlin
Mollstraße 1
juliane.dehnert@isst.fraunhofer.de

Abstract: The use of Workflow Management Systems allows to automate numerous Business Processes within companies. For this, workflow definitions as executable derivates of business process descriptions must be formulated in a formal specification language. For both - the less formal descriptions of business processes and the workflow definitions - Petri net based approaches are widely used and well understood. In this paper, we study the process of deriving workflow definitions from less formal business process descriptions and even fragments of such descriptions and how to satisfy the correctness of these models among each other.

1 Introduction

In the past years, theoretical work on the process part of Workflow Management Systems (WfMS) was developed mainly into two directions: on the one side, sketchy descriptions of business processes without operational semantics like for example Event-driven Process Chains (EPCs, cf. [KNS92]) are used to help understanding given business process structures. They profit from the ability of humans to develop an intuitive comprehension of visual information and can therefore be used as a medium for discussions. On the other side, formal approaches to process specification like Petri nets [Mu89, RR98] in general or Workflow nets (WF-nets, cf. [Aa98]) in special are used to formalize process specifications which are executable within WfMSs. Beside their characteristic of being executable, they profit from existing analyzing methods which can be used for checking their consistency.

Based on the transformation of sketchy descriptions of business processes into formal and executable specifications, we prove with the aid of a Logic of Actions (LoA) that this transformation is correct. Moreover, we take into account that initial descriptions sometimes only deal with business process details (i.e. fragments of the entire business process like *never deliver without being paid*) that must be integrated into each other for deriving an executable form. Hereby, it must be guaranteed that these specifications are fulfilled by the final implementation of the processes in a WfMS.

Explaining our solution satisfying the mentioned requirements, we use EPCs and WF-nets as exemplary notations. The proposed approach combines existing work on (1) model transformation from EPCs into relaxed sound WF-nets, i.e. a formalization of given EPC descriptions, (2) transformation of relaxed sound WF-nets into sound WF-nets, i.e. executable workflow definitions, and (3) techniques to prove given process implementations

against process specifications investigated for LoA.

The paper is organized as follows: after introducing the modeling of business processes with EPCs by means of an example, we transform this example into a relaxed sound WF-net introducing this net class formally at this opportunity. Afterwards, we relate the results concerning relaxed soundness to findings in LoA and exemplarily specify a business process fragment within this logic. We then enhance the relaxed sound towards a sound WF-net, as only a sound workflow specification will guarantee a smooth execution of the supported business process at run-time. Finally we prove the correctness of the resulting model against the formerly found specifications and close with some concluding remarks.

2 EPC Models of Business Processes

EPCs are a typical representative for a *business process modeling language.* It is part of the Architecture of Integrated Information Systems (ARIS, cf. [Sc94]).

EPCs are a graphical and semiformal modeling language. Although or maybe because they involve ambiguity and vagueness, they are easy to learn and understand. This and the use of EPCs for representing SAP reference models caused that EPCs are fairly widespread.

The basic elements of EPCs are *Functions* used to model the dynamic part of processes and *events* which either trigger functions or indicate their termination. Furthermore, *connectors* are used to build complex process structures among the elementary process elements. They fall into the two categories *split* and *join* each of which are separated into three classes: and ($\wedge$), xor ($\times$), and or ($\vee$). Figure 1 visualizes the basic elements of EPCs.

Figure 1: EPC elements

The composition of EPCs among these elements is restricted by some rules such as:

- Except start and end events, all events and functions have exactly one incoming and one outgoing arc.
- EPCs are connected.
- Events and functions follow each other possibly separated by connectors. [DR01]

As an example, let us consider the EPC "Handling of incoming order" of figure 2 a). It represents a reduced version of a real life process of a telephone company modeling the ordering of a mobile phone which involves two departments: the accountancy is handling the payment and the sales is handling the distribution.

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 cancelled 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 an incoming order" is completed by archiving information on that instance (`archive`).

Now, the following problem must be solved: the domain experts do not necessarily have modeling expertise - nonetheless, their models must be made executable in a WfMS. Using a semi-formal notation which does not fully restrict the structure of the models but supports the creativity of the experts when describing their domain is typically helpful and motivates notations like EPC. They describe the involved tasks and their order as observed (or wished) to happen. Thus, the resulting models are to be understood as patterns depicting desired behavior but not as executable specifications. Because of the non-operational semantics, transformations have to be conducted for translating EPCs (or comparable models) into formal models such as Petri nets. We use *relaxed sound Workflow nets* as an intermediary representation giving these informal graphs a formal interpretation. The following section explains this step.

3 Relaxed Sound Workflow Nets

As a means for the description of complex dynamic systems, Petri nets play an important role in the field of Workflow Management. We do not introduce Petri nets formally here but refer to the standard literature on this topic such as [Mu89, RR98].

WF-nets are Petri nets having a unique source place i and a unique sink place o describing that any instance of a workflow specification called *case* (cf. [Aa98]) handled by a WF-net is created when it enters the WfMS and is deleted once it is completely handled by the WfMS. In order to ensure that every *task* (represented by a transition) or *condition* (represented by a place) contributes to the processing of cases, each transition and place must be on some path from i to o.

Definition 1 (WF-net, WF-system) *A Petri net $\mathcal{N} = (P, T, F)$ is a WF-net, if:*
(i) $\mathcal{N}$ 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 $\overline{\mathcal{N}} = (P, T \cup \{t^*\}, F \cup \{(o, t^*), (t^*, i)\})$ is strongly connected.

A WF-system $(\mathcal{N}, i)$ is a WF-net in which place i is initially marked.

In [DR01] we find transformation rules to translate informal EPCs into Petri nets using an intuitive common sense of the basic EPC elements and their connections. The proposed transformation does not resolve ambiguities inherent in the semi-formal process description, but makes them explicit. Evaluating the resulting WF-net, it must then be decided whether the described behavior is meaningful. Note that using an adapted correctness notion, namely relaxed soundness [DR01], such an evaluation is possible, even for sophisticated situations as described in [Ki04].

The application of this approach to the EPC of figure 2 a) results in the WF-system shown in figure 2 b). The WF-net (and therefore the primary EPC) is considered correct if it can be interpreted as follows: it specifies all *business processes* in terms of *sequences of tasks* for which there is a firing sequence from the initial state i to the final state o, such that the

transitions for these tasks occur in the order of such a, so-called *sound*, firing sequence.

Definition 2 (Sound firing sequence) *Let $S = (\mathcal{N}, i)$ be a WF-system. A firing sequence σ is sound iff $i \xrightarrow{\sigma} m$ and $\exists \sigma' : m \xrightarrow{\sigma'} o$.*

The definition of *sound firing sequence* allows a formalization of business processes in terms of WF-system satisfying the property *relaxed soundness*:

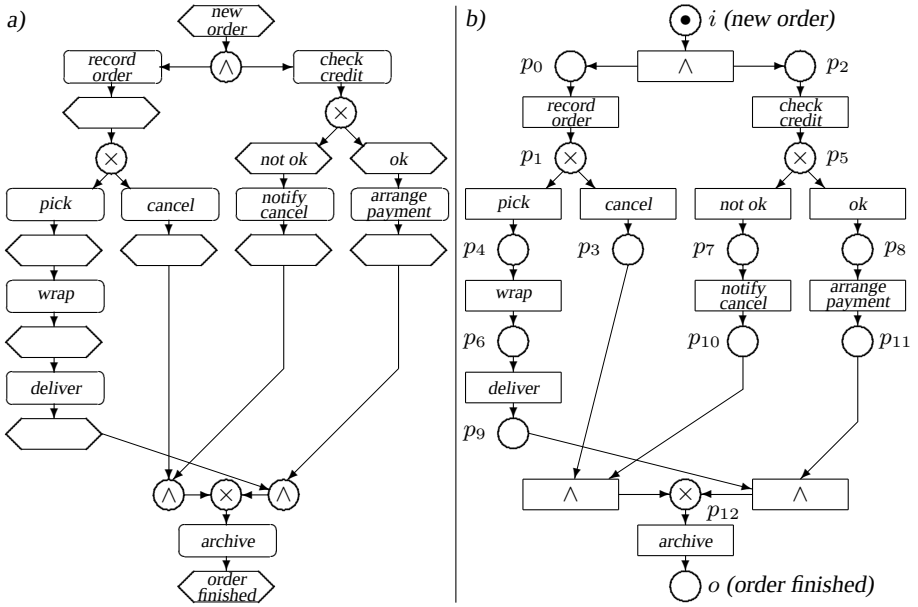

Figure 2: a) EPC and b) WF-net "Handling an incoming order"

Definition 3 (Relaxed soundness) *A workflow system $S = (\mathcal{N}, i)$ is called relaxed sound iff each transition of $\mathcal{N}$ is an element of some sound firing sequence: $\forall t \in T : \exists m, m' : (i \xrightarrow{*} m \xrightarrow{t} m' \xrightarrow{*} o)$.*

This definition still leaves room for ambiguities since it does not demand the precision of workflow definitions as they are required for their execution within a WfMS. In contrast to *soundness* [Aa98], it does not require all firing sequences to be sound, but only requires all tasks to be covered by at least one sound firing sequence. Therefore relaxed soundness only evaluates the intended behavior. Undesired behavior is disregarded. Having a relaxed sound representation of business processes allows proving these models against process specifications. While usual logics are *state* oriented, the Logic of Actions introduced in the following section is *process* oriented and therefore appropriate for our needs.

4 A Logic of Actions and Its Representation as Petri Nets

Sound firing sequences as they have been defined for WF-nets have a counterpart within the *Logic of Actions (LoA)* (cf. [Si01]). Within this logic, *processes* are formally built upon elementary processes in which *actions* either *occur* or *are forbidden.* Such elementary processes can occur in sequence or coincidentally with each other. Hereby, actions must not occur and are being forbidden within the same process - otherwise, they contradict each other.

With respect to our business process of figure 2,

$$\begin{array}{l}(\texttt{new order} \otimes \texttt{check credit} \otimes \texttt{record order} \otimes \\ \ (\texttt{cancel} \oplus \texttt{notify cancel}) \otimes \\ \texttt{archive} \otimes \texttt{order finished})\end{array}$$

describes the execution of one case where the mentioned actions occur in sequence except `cancel` and `notify cancel` which in this case occur synchronously. Other cases are represented by LoA processes in a comparable way.

Modules, the formulas of the Logic of Actions, are the *syntax* of this logic. They are interpreted by sets of processes which can be executed by the module. These processes define the *semantics* of a module.

Formally, modules are defined as *elementary* (an action occurs or is being forbidden) or as *composite* using the operators coincidence, sequence, (exclusive) alternative, conjunction, and complement. Furthermore, iterations can be formulated within this logic.

As an example, we consider the business process of figure 2 again.

$$\begin{array}{l}\Big[\texttt{new order} \boxleft \\ \quad \Big[\big[\texttt{record order} \boxleft [\texttt{pick} \boxleft \texttt{wrap} \boxleft \texttt{deliver}] \boxplus \texttt{cancel}\big] \boxwedge \\ \quad\ \big[\texttt{check credit} \boxleft [\texttt{notify cancel} \boxplus \texttt{arrange payment}]\big]\Big] \boxleft \\ \texttt{archive} \boxleft \texttt{order finished}\Big]\end{array}$$

is read as follows: each process starts with a `new order`. Afterwards, two sub processes are both executed ($\boxwedge$) which deal with payment and sales. Within these sub processes, exclusive alternatives ($\boxplus$) can be observed. Finally, each process ends with archiving and finishing the order.

These preliminary definitions of processes and modules allow us to formulate and argue about processes in general. Beside this, modules of LoA can be represented as Petri nets, so called *Module nets.* An explicit *start* transition (this is the source for each process) puts tokens into an initially empty Module net. Afterwards, the transitions of the Module net fire until a *goal* transition (this is the sink) reproduces the empty initial marking. All transitions firing in the meantime are interpreted by actions. The sequence in which the transitions are firing indicates the process executed by a Module net.

Definition 4 (Module net) *A Module net $\mathcal{M} = (\mathcal{N}, \iota)$ consists of a Petri net $\mathcal{N} = (P, T, F)$ with start transition s ($\bullet s = \emptyset$), goal transition g ($g\bullet = \emptyset$), and an interpretation function ι over T with $\iota(s) \mapsto$ start, $\iota(g) \mapsto$ goal and for all other transitions t*

$\iota(t) \mapsto E_{\mathcal{A}}$, a (possibly empty) set of non-contradicting elementary processes over a set of actions $\mathcal{A}$.

A firing sequence $\langle T_1, \ldots, T_n \rangle$ in which some transitions might fire coincidentally in sets T_j reproducing the empty initial marking of a Module net $\mathcal{M} = (\mathcal{N}, \iota)$ is a process of $\mathcal{M}$.

This definition shows that relaxed sound WF-nets and Module nets which have been developed independently from each other represent the same kind of processes which can both be expressed in terms of LoA. The distinctions (sink and source places vs. transitions) are due to the special purposes of both types of nets: WF-nets are used to describe cases starting in an initial state i while Module nets are used to specify sets of processes all starting with firing s. However, the structure of both types of nets can easily be transformed into one another as shown in [De03, p. 82]. For this, tasks and actions are considered to be comparable concepts. The possibility to explicitly forbid the occurrence of actions means an extension of Module nets to WF-nets, an extension into the other direction does not exist. As a consequence, we can open the proving methodologies of LoA also for WF-nets.

Proving in LoA bases on a comparison of the process sets specified by modules. In order to make them comparable they have to be completed. For this, we augment Module net implementations by an action which occurs or is forbidden if this action is only mentioned within one branch of an alternative. Hereby, every action of a Module net either occurs or is being forbidden in any process.

Based on these considerations, we can formulate whether and how an implementation fulfills a specification:

Definition 5 (Sound and Complete) *Let S be a specification formulated as a module over a given set of actions $\mathcal{A}$ and I an implementation formulated also as a module over the same set of actions $\mathcal{A}$.*

- *I is sound with respect to S iff I implies S ($I \vdash S$), i.e. if for the process sets Π of the mutually completed modules $\Pi(\mathcal{C}_S[I]) \subseteq \Pi(\mathcal{C}_I[S])$ holds true.*
- *I is complete with respect to S iff $S \vdash I$.*

Instead of comparing the process sets of modules S and I, we use their Module net implementations $\mathcal{M}[S]$ and $\mathcal{M}[I]$ to which we apply the following theorem proven in [Si01, pp. 56-57] and the fact that the conjunction of modules is implemented by joining the participating Module nets.

Theorem 1 (Verification of Wanted and Forbidden Behavior) *For two modules S and I as specified in definition 5, the following holds true:*

$$I \vdash S \Leftrightarrow \Pi(\mathcal{M}(I \boxed{\triangle} S)) = \Pi(\mathcal{M}(\mathcal{C}_S[I]))$$
$$I \vdash S \Leftrightarrow \Pi(I \boxed{\triangle} \overline{S}) = \emptyset$$

The first part of theorem 1 can be used for *direct* and the second for *indirect proving.*

The ability to implement the conjunction operation on Module nets is of central importance. This is done by *joining* all transitions of the participating nets which share the same interpretation and by ruling out that one action might occur and is being forbidden in the same process.

Now, we are applying this findings to our example: as a *fragment* of our considerations concerning the transformed business process in figure 2, we have the information that there never must be a `deliver` and a `cancel` task within one process. In terms of LoA this is specified as $\overline{S} = [\texttt{deliver}\boxed{\wedge}\texttt{cancel}]$, i.e. $\overline{S}$ specifies forbidden behavior which a correct implementation of our business process must prevent. Figure 3 shows the implementation of $\overline{S}$ as a Module net and a WF-net.

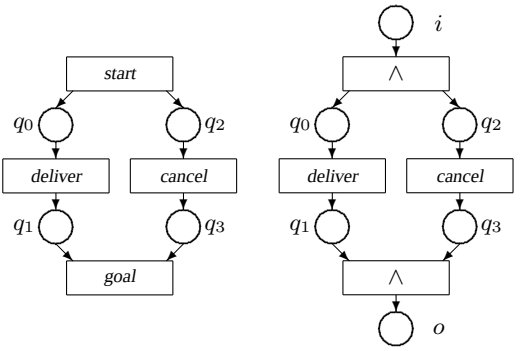

Figure 3: Module net and WF-net of $\overline{S}$

S (the specification of the wanted behavior) is fulfilled, if in application of theorem 1 part 2 the join of the specification of the forbidden behavior $\overline{S}$ and the original WF-net results in a net without any sound firing sequence. Figure 4 shows the net resulting from this join. We can show by both, simulation and analysis that within the joined net no such firing sequence exists. We therefore conclude that our model of figure 2 fulfills our specification and prevents the forbidden behavior S from occurring.

5 Executable Workflow Nets

In general, a relaxed sound WF-net cannot be used to support a business process at runtime. Relaxed soundness only states that all intended behavior was covered, but does not exclude faulty executions. To use the WF-net as input for a WfMS, all firing sequences must be sound. In this paper we will call a WF-net satisfying this requirement *executable* (corresponds to soundness as defined in [Aa98]. For transforming a relaxed sound WF-net into an executable WF-net all not sound firing sequences of the WF-system must be prevented from occurring. In [DZ04] methods from controller synthesis have been applied to restrict the behavior of WF-systems to sound firing sequences only. The idea is to compute places that supervise or control the behavior of the Petri net. The introduced places, called

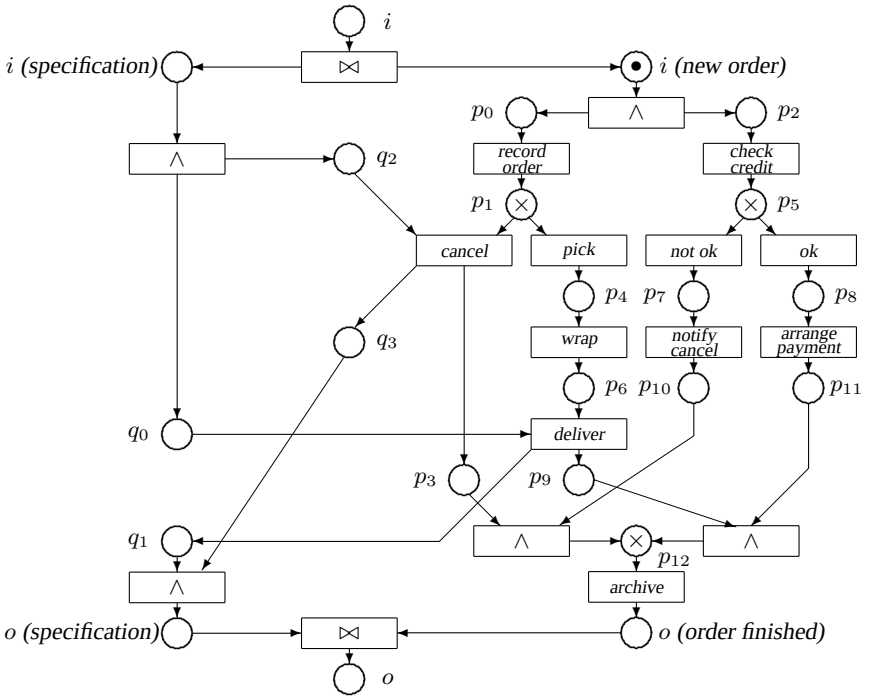

Figure 4: Join of forbidden behavior and the WF-net of figure 2

controller places or monitors, avoid entering a set of "forbidden states"[1]. The information needed for their computation can be gained in various ways, e.g. from place invariants [YMLA96], general mutual exclusion conditions (GMECs) [GDS92], or sets of forbidden markings [GRX02]. The places are added to the primary WF-net. Applying an algorithm implemented in the tool Synet [Ca97] to our running example "Handling of incoming order" two controller places pc_1 and pc_2 are computed. The derived WF-net is shown in figure 5.

Obviously, extending a given relaxed sound WF-net by additional places in order to prevent non-sound firing sequences is concise with the findings of LoA. Joining the original net and the augmented executable net must always result in the executable net again. If we now consider a relaxed sound net (as a result of a modeling process) as a specification S for the executable WF-net I (which is nothing but an implementation of the relaxed sound net), we can consequently apply theorem 1 part 1 and conclude that I is sound with respect to S.

Alternatively to the above mentioned approach a modeler might decide to redesign the business process into an executable version. Figure 6 shows such a model as an EPC and its transformation into a WF-net. Within this example, soundness of the WF-net is achieved by solving the alternative first and using AND-splits afterwards.

[1] 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.

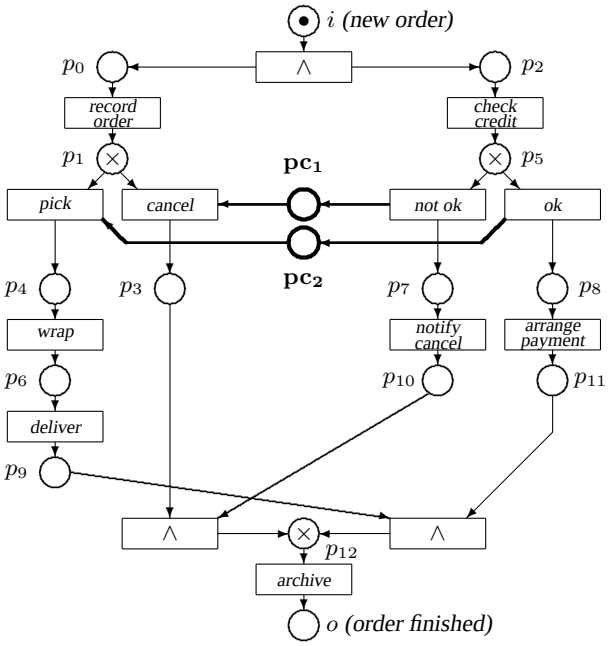

Figure 5: Executable WF-net "Handling an incoming order" with controller places

In order to find out whether this variation of the business process is still concise with the model of figure 2 we have to join both nets.

The resulting net is presented in figure 7. It can be simplified drastically deleting the redundant places p_4, p_5, and p_6 (i.e. places for which another place exists with the same preset and postset) and by looking for implication invariants as defined by [CPA94]. These implication invariants establish a relation between the marking of two sets of places P' and P'' such that tokens on P' imply the existence of tokens on P''.

The implication invariants $m(\{q_1, q_3\}) \Rightarrow m(\{p_7\})$ and $m(\{q_2, q_6\}) \Rightarrow m(\{p_8\})$ allow a deletion of places p_7 and p_8 because they do not restrict the firing of transitions `notify cancel` and `arrange payment`.

Because of the invariants $m(\{i(\textit{specification}), p_2\}) \Rightarrow m(\{i(\textit{new order})\})$, $m(\{p_2, q_0, q_1, q_2, q_4, q_5\}) \Rightarrow m(\{p_0\})$ and $m(\{q_7, q_9, q_{11}, q_{14}, q_{15}\}) \Rightarrow m(\{p_3, p_9, p_{10}, p_{11}, p_{12}\})$ we can also delete all other places labelled with p (except p_2) because these places are only used to control the firing of internal transitions. The control of transitions with associated tasks is controlled by transitions labelled with q already. Remaining sequences of places and purely structural transitions at the beginning and end of the WF-net can be summarized into a single i place and a single o place.

The resulting net is exactly the executable net of figure 7 which consequently fulfills the specification formulated by a modeler in terms of a relaxed sound net.

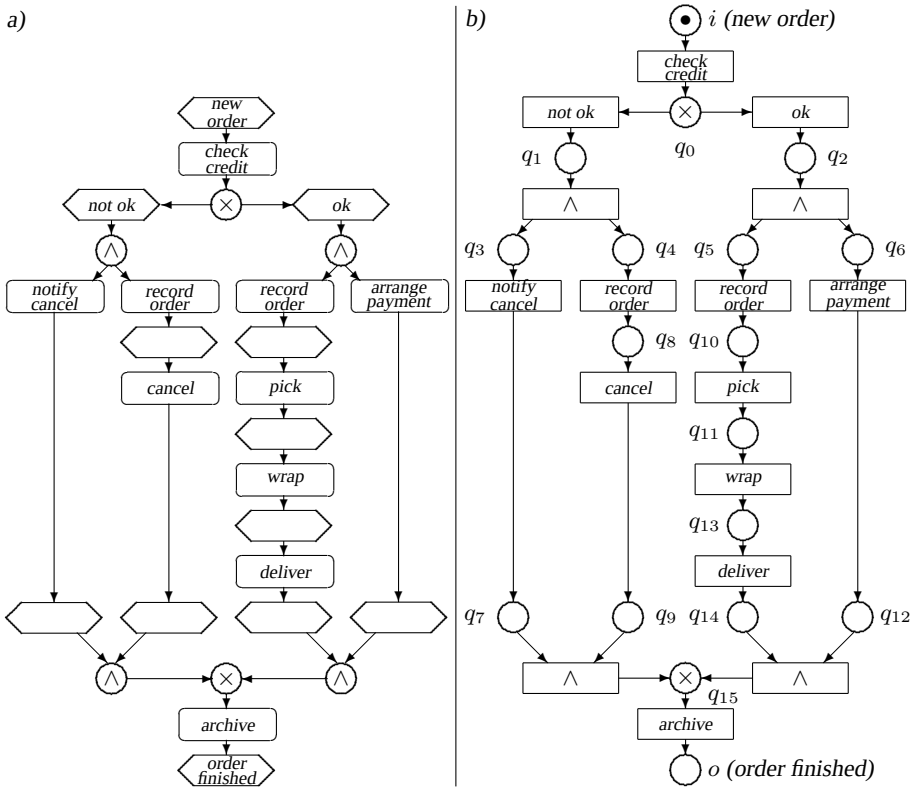

Figure 6: Sound EPC a) and WF-net b) "Handling an incoming order"

6 Concluding Remarks

In this paper we revealed the relationship between relaxed soundness of business process descriptions and the concept of process specifications investigated for a Logic of Actions. The close link between the both can be used in especially two ways:

Proving methodologies of LoA can be opened for WF-nets: exemplarily this is demonstrated by verifying process fragments (even of forbidden behavior) against an implementation formulated as a relaxed sound WF-net.

Comparing process sets by applying LoA proving techniques: exemplarily this is demonstrated by verifying that two executable WF-nets (hereby one resulting from a transformation process) specify the same behavior.

Our future work will be in investigating how to use this approach for *distributed modeling*. Hereby, the application of the join-operator of LoA allows the merging of modeling results of various domain experts (e.g. reflecting the behavior in different departments) within a single model. The resulting net will probably be relaxed sound, however can then be transformed into a sound and executable specification as demonstrated in this paper.

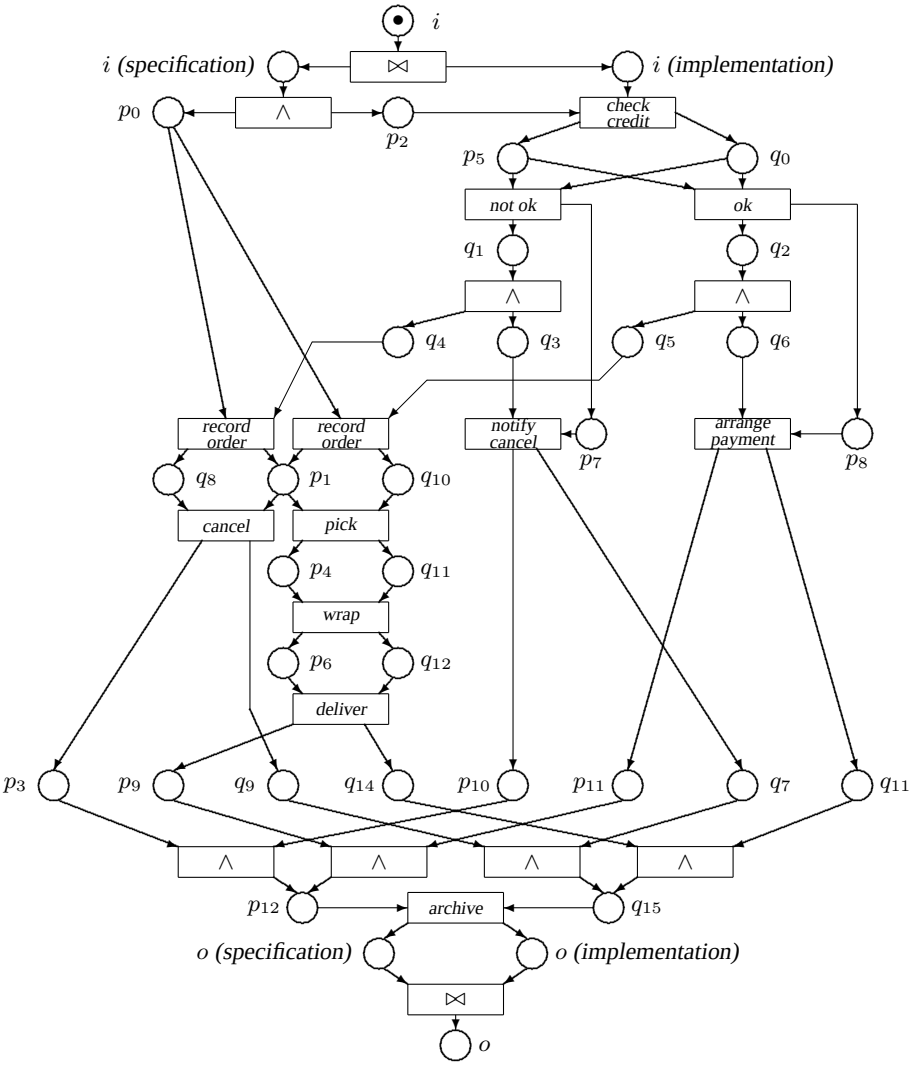

Figure 7: Join of the relaxed sound and executable WF-nets “Handling an incoming order”

References

[Aa98] Aalst, W.: The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*. 8(1):21–66. 1998.

[Ca97] Caillaud, B.: 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.

[CPA94] Couvreur, J. M. und Paviot-Adet, E.: New structural invariants for petri nets analysis. In: Valette (Hrsg.), *Application and Theory of Petri Nets 1994*. Lecture Notes in Computer Science 815. S. 199–218. Springer. 1994.

[De03] Dehnert, J.: *A Methodology for Workflow Modeling - From business process modeling towards sound workflow specification*. PhD thesis. TU Berlin. 2003.

[DR01] Dehnert, J. und Rittgen, P.: Relaxed Soundness of Business Processes. In: Dittrich, K., Geppert, A., und Norrie, M. (Hrsg.), *Advanced Information System Engineering, CAISE 2001*. volume 2068 of *LNCS*. S. 157–170. Springer. 2001.

[DZ04] Dehnert, J. und Zimmermann, A.: Making Workflow Models Sound Using Petri Net Controller Synthesis. In: *International Conference on Cooperative Information Systems*. 2004. to appear.

[GDS92] Giua, A., DiCesare, F., und Silva, M.: Generalized mutual exclusion constraints on nets with uncontrollable transitions. In: *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*. S. 974–979. Chicago, IL. 1992.

[GRX02] Ghaffari, A., Rezg, N., und X.Xie: Live and Maximally Permissive Controller Synthesis Using the Theory of Regions. In: Caillaud, B., Darondeau, P., Lavagno, L., und Xie, X. (Hrsg.), *Synthesis and Control of Discrete Event System*. S. 155–166. Kluwer Academic Press. 2002.

[Ki04] Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Desel, J., Pernici, B., und Weske, M. (Hrsg.), *Business Process Management (BPM 2004)*. volume 3080 of *Lecture Notes in Computer Science (LNCS)*. Potsdam, Germany. 2004. Springer.

[KNS92] Keller, G., Nüttgens, M., und Scheer, A. W.: Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89. University of Saarland, Saarbrücken. 1992.

[Mu89] Murata, T.: Petri Nets: Properties, Analysis, and Applications. *Proc. of the IEEE*. 77(4):541–580. April 1989.

[RR98] Reisig, W. und Rozenberg, G. (Hrsg.): *Lectures on Petri Nets I: Basic Model / II: Applications*. volume 1491/1492 of *LNCS*. Springer. 1998.

[Sc94] Scheer, A. W.: *Business Process Engineering, ARIS-Navigator for Reference Models for Industrial Enterprises*. Springer. 1994.

[Si01] Simon, C.: *A Logic of Actions and Its Application to the Development of Programmable Controllers*. PhD thesis. Universität Koblenz-Landau. 2001.

[YMLA96] Yamalidou, K., Moody, J., Lemmon, M., und Antsakli, P.: Feedback control of Petri nets based on place invariants. *Automatica*. 32(1):15–28. 1996.