Verification of the SAP Reference Models using EPC Reduction, State Space Analysis, and Invariants

B.F. van Dongen , M.H. Jansen-Vullers, H.M.W. Verbeek , W.M.P. van der Aalst

Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.

Abstract

A reference model is a generic conceptual model that formalizes recommended practices for a certain domain. Today, the SAP reference models are among the most comprehensive reference models, including over 4000 entity types and covering over 1000 business processes and inter-organizational scenarios. The SAP reference models use Event-driven Process Chains (EPCs) to model these processes and scenarios. Like other informal languages, EPCs are intended to support the transition from a business model to an executable model. For this reason, researchers have tried to formalize the semantics of EPCs. However, in their approaches, they fail to acknowledge the fact that in EPCs constructs exist that require human judgment to assess correctness. This paper aims to acknowledge this fact by introducing a two-step approach. First, the EPC is reduced using universally accepted reduction rules. Second, the reduced EPC is analyzed using a mixture of state-space analysis, invariants, and human judgment. This approach has been implemented in a tool, and using this tool we show that the SAP reference models contain errors, which clearly shows the added value of this verification approach.

Key words: Process mining, Petri nets, Event-driven Process Chains, Verification, SAP R/3, Reference Models.

Email addresses:
b.f.v.dongen@tm.tue.nl (B.F. van Dongen),
m.h.jansen-vullers@tm.tue.nl (M.H. Jansen-Vullers),
h.m.w.verbeek@tm.tue.nl (H.M.W. Verbeek),
w.m.p.v.d.aalst@tm.tue.nl (W.M.P. van der Aalst).

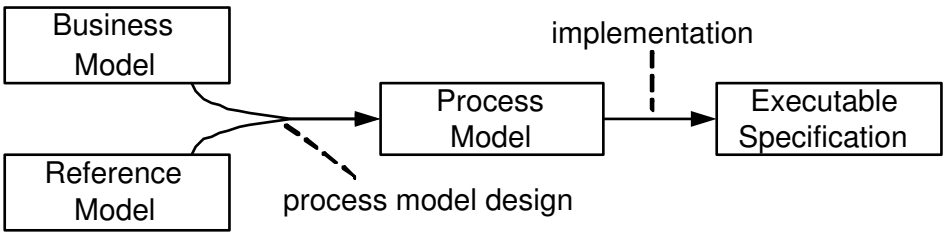

Fig. 1. Phases in the configuration of a PAIS

1 Introduction

1.1 Reference models

Today, *process-aware information systems* (PAISs) such as Enterprise Resource Planning (ERP) [1] systems and Workflow Management (WFM) [2,3] systems are used to support a wide range of operational business processes. On an operational level, these systems are often configured on the basis of a process model. The design of such a process model is a complicated and error prone task, and since process models that are designed in difference companies are often very similar, databases with such process models have been developed. These databases are called *reference models*, as they can be used as a reference during process design. Today, reference models exist for many different applications.

Together with the business model of a company, a reference model is selected that fits the process under consideration best. During the process model design phase, a designer customizes the reference model to fit the business model of the company. The result of this customization phase is an abstract specification of the process in terms of a customized process model. In the implementation phase, this model is used to implement an executable model for a specific information system, such as SAP R/3. These phases are presented in Figure 1. Since all steps between selecting a reference model and producing an executable specification are performed by humans, errors could very well be introduced.

Unfortunately, the use of reference models does not eliminate the possibility of introducing errors into the process model. It should, however, assist the designer in such a way that errors are less likely to be introduced. Thus, it is of the utmost importance that the selected reference model is correct itself. Note that errors introduced in the process model may have severe consequences if they are discovered as late as in the implementation phase or even if the system is operational, let alone if they are never discovered at all.

To be able to detect possible errors in process models, many authors have

developed verification methods. Basically, all these verification methods can be used to check whether a process model is *correct*, in other words, they can be used to check for *correctness* of a process model. However, the process model might only be incorrect for situations that will never occur in practice. Consider, for example, a process model that can be triggered by a telephone call or the receipt of a letter. If the designer knows that the call and letter exclude each other, then the process model is allowed to be incorrect for both triggers. Note that this exclusion information is available to the designer, and that the aforementioned verification methods are not capable of handling this crucial information.

1.2 SAP

This paper focuses on the correctness of reference models for a specific information system, namely SAP R/3 (or just SAP). We selected SAP as it uses *Event-driven Process Chains* (EPCs) [1,4,5] as its modeling language. EPCs are used in a large variety of other systems, and many verification approaches exist for EPCs. The SAP reference models are available in the ARIS for MySAP database from the ARIS Toolset, a commercial product of IDS-Scheer. Using these SAP reference models, we present our verification approach and show that many of the SAP reference models are correct and can indeed be used without any problems. However, we also show that some of the models should be used with care, i.e., only if the environment in which they are used satisfies certain conditions they are correct. Furthermore, we show that several reference models are structurally incorrect, i.e., they need to be revised before they can be used as reference models. With respect to these errors, we investigate some common causes, and show how designers could avoid these errors.

1.3 EPCs

Because this paper is about EPCs, it seems appropriate to use a (meta-level) EPC to describe our approach, and to use this EPC to explain the issues at hand. Figure 2 shows this EPC, which will be discussed in more detail later on. The actual verification process consists of two steps. First, we use reduction rules to eliminate the "easy" constructs for which it is generally accepted that they are correct. This paper introduces these reduction rules later on, for now it suffices to mention that all functions and events except initial and final events are removed, and that local choices and trivial synchronization constructs are eliminated. If the EPC at hand reduces to the trivial EPC, it is correct. Otherwise, we take the second step. In the second step, we translate the EPC

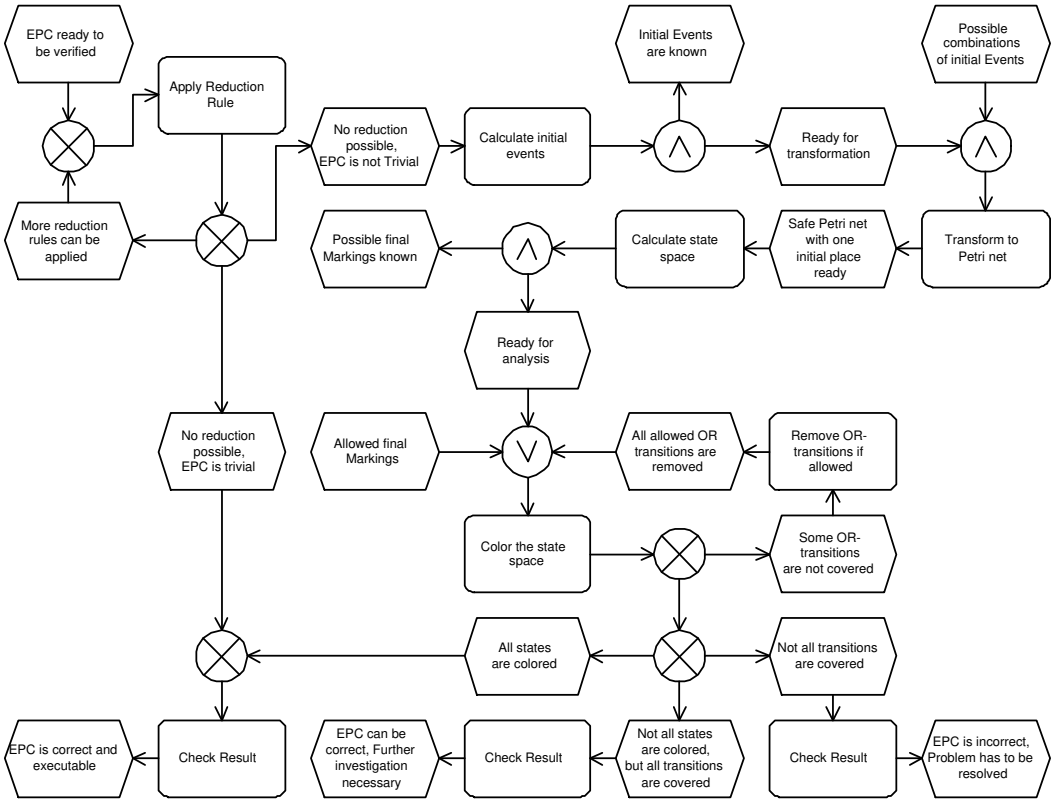

Fig. 2. EPC describing the EPC verification process.

into a Petri net and use the theory of workflow nets [6,2] and related concepts (such as soundness [6] and relaxed soundness [7,8]) to verify the reduced EPC. If the state space of the Petri net can be constructed within reasonable time, the second step provides the designer with one of the following three answers:

The Petri net is *sound*: The original EPC is correct and no further reviewing is necessary.

The Petri net is *relaxed sound* (but not sound): The original EPC can be correct, but the designer needs to assess some problematic constructs. If the designer assesses the EPC to be incorrect, corrections are necessary.

The Petri net is *not relaxed sound*: The original EPC is incorrect, regardless of the designer's assessment. Corrections are necessary.

The construction of the state space may be time-consuming (especially if the model is incorrect). Fortunately, techniques exist that do not depend on the state space and that can show the EPC to be not relaxed sound: *transition invariants* (T-invariants) [9–11]. Our approach can use these transition invariants instead of relaxed soundness in case the state space turns out to be too complex to be constructed within reasonable time. These transition invariants however only give sufficient conditions to decide that a process is not sound, it does not guarantee soundness, therefore using transition invariants alone would not suffice.

To understand the rationale of our approach, it is important to see that we address the issue of verification from a designer's perspective instead of from a formal perspective, where the result would always be sound or not sound without any need for interpretation of the process model. Instead, our verification process consists of two main parts. First we take the EPC that is defined by a process designer and, using simple reduction rules, we reduce the net. As a result, the designer can focus on the possibly problematic areas. Second, we translate the result into a Petri net and use variants of existing Petri-net-based verification techniques to give feedback to the designer. By using a more relaxed correctness notion (similar to relaxed soundness [7,8]) and, if needed, human judgment, the correctness of the model is assessed. Especially the explicit use of human judgement is a new factor in the area of process verification.

1.4 Tool

We have developed a tool for the analysis of EPCs using the approach described in this paper. The tool has been implemented as a verification plug-in in the Process Mining (ProM) Framework [1], which can exchange process models with a variety of other software tools including the ARIS toolset and PNML-enabled Petri net tools.

1.5 Agenda

Section 2 discusses related work. Section 3 discusses necessary concepts like EPCs, Petri nets, state spaces, transition invariants, etc. Section 4 introduces a set of powerful but simple reduction rules for EPCs that will be used in the first step of our approach. Section 5 introduces the mapping from EPCs onto Petri nets and discusses the verification process (second step) in more detail. Section 6 describes the use of transition invariants if the state space of the reduced EPC happens to be too complex. Section 7 presents the analysis tool and discusses some initial experiences when applying the tool to real-life processes. Section 8 applies our tool on the SAP reference models and discusses the results. Section 9 concludes the paper.

[1] See `www.processmining.org` for more information and to download the software.

2 Related work

Since the mid-nineties, a lot of work has been done on the verification of process models, and in particular workflow models. In 1996, Sadiq and Orlowska [12] were among the first to point out that modeling a business process (or workflow) can lead to problems like livelock and deadlock. In their paper, they present a way to overcome syntactical errors, but they ignore the semantical errors. Nowadays, most work that is conducted is focusing on semantical issues, i.e., "will the process specified always terminate" and similar questions. Therefore, in this section we start by introducing related work on verification of models. Furthermore, we present additional EPC-related literature, and end with the main differences with regard to our approach.

2.1 Verification

The work on verification that has been conducted in the last decade can roughly be put into three categories. In this section, we present these categories and give relevant literature for each of them.

2.1.1 Formal models

In the first category we consider the work that has been done on the verification of modeling languages with formal semantics. One of the most prominent examples of such a language are Petri nets [9,10,13]. Since Petri nets have a formal mathematical definition, they lend themselves to a large extent for formal verification methods. Especially in the field of *workflow management*, Petri nets have proven to be a solid theoretical foundation for the specification of processes. This, however, led to the need of verification techniques, tailored towards Petri nets that represent workflows. In the work of Van der Aalst and many others [6,7,14–16], these techniques are used extensively for verification of different classes of workflow definitions. However, the result is the same for all approaches:

> Given a process definition, the verification tool provides an answer in terms of "correct" or "incorrect".

However, not all modeling languages have formal semantics. On the contrary, the most widely used modeling techniques, such as UML and EPCs, are merely an informal representation of a process. Therefore, these modeling techniques require a different approach towards verification.

2.1.2 Informal models

Modeling processes in a real-life situation is often done using less formal languages. People tend to understand informal models more easily, and even if models are not executable, they can help a great deal when discussing process definitions. However, at some point in time, these models usually have to be translated into a specification that can be executed by an information system. This translation is usually done by computer scientists, which explains the fact that researchers in that area have been trying to formalize informal models for many years now. Especially in the field of workflow management, a lot of work has been done on translating informal models to Petri nets. Many people have worked on the translation of EPCs to Petri nets, cf., [8,17–19]. The basic idea of these authors however is the same:

> Restrict the class of EPCs to a subclass for which we can generate a sound Petri net.

As a result, the ideas are appealing from a scientific point of view, but not always as useful from a practical point of view.

Also non-Petri-net based approaches have been proposed for the verification of informal modeling languages. One of these ideas is *graph reduction*. Since most modeling languages are graph-based, it seems a good idea to reduce the complexity of the verification problem by looking at a reduced problem, in such a way that correctness is not violated by the reduction, i.e. if a model is not correct before the reduction, it will not be correct after the reduction, and if the model is correct before the reduction, it will be correct after the reduction. From the discussion on graph reduction techniques started by Sadiq and Orlowska in 1999 [20,21] [2] and followed up by many authors including Van der Aalst et al. in [22] and Lin et al. in [23], it becomes clear that again the modeling language is restricted to fit the verification process. In general this means that the more advanced routing constructs cannot be verified, while these constructs are what make informal models easy to use.

The tendency to capture informal elements by using smarter semantics is reflected by recent papers, cf. [8,18,24–26]. In these papers, the problem is looked at from a different perspective. Instead of defining subclasses of models to fit verification algorithms, the authors try to give a formal semantics to an informal modeling language. Even though these authors have different approaches, the goal in every case is similar:

> Try to give a formal executable semantics for an informal model.

As a result, these approaches all fail to take the designer's knowledge into account.

[2] Note that the analysis technique presented in [20,21] is incorrect. As shown in [22,23] not all correct models can be reduced.

2.1.3 By design

The last category of verification methods is somewhat of a by-stander. Instead of doing verification of a model given in a specific language, it is also possible to give a language in such a way that the result is always correct. An example of such a modeling language is IBM MQSeries Workflow [3]. This language uses a specific structure for modeling, which will always lead to a correct and executable specification. However, modeling processes using this language requires advanced technical skills and the resulting model is usually far from intuitive.

2.2 Execution of informal models

It is interesting to note that verification is strongly related to the efficient execution of models. Especially the approaches presented in the previous paragraph, all rely on executable semantics of the process model under consideration. As an example, we mention YAWL models. YAWL models use an OR-join of which the intuitive idea is taken from EPCs. To obtain executable semantics for YAWL models, YAWL models are mapped onto reset nets to decide whether an OR-join is enabled or not in [27]. In the context of EPCs the possibility to provide executable semantics has been investigated in [25], where executable semantics are proven to exist for a large sub-class of all EPCs. In [26] an approach is presented to efficiently calculate the state space of an EPC, thereby providing executable semantics for the EPC. The authors mainly motivate this work from the viewpoint of simulation/execution although their approach can also be used for verification purposes. Because of the semantical problems in some EPCs [25] the algorithm does not always provide a result. Moreover, the authors also point out the need for "chain elimination" to reduce the state space of large models.

2.3 Conclusion

In this section, we have presented an overview of the literature on process model verification and related subjects. We have categorized the various verification methods in three main categories and pointed out why many of them are not used in practice. The main difference between our approach and existing literature is that we will not restrict an informal modeling language to fit our verification, nor will we give executable semantics of an informal model. Instead, we combine the best of existing literature and provide the designer with a tool to find possible problems in a specification. We do not aim at solving these problems. Instead, we assume the designer to be able to decide whether or not a specification is correct.

This paper builds on our earlier work of the authors on EPCs and reference modeling. In [28] we presented the idea of verifying SAP models and in [29] we presented the idea of using reduction rules for verification purposes. The main contributions of this paper are a detailed description of the verification approach, extended with the use of transition invariants and a detailed analysis of the SAP reference models.

3 Preliminaries

In this section, we introduce the basic concepts used in the verification process. We briefly introduce the languages used in this paper, i.e., EPCs and Petri nets. Furthermore, we introduce the notions of soundness and relaxed soundness and provide pointers to the Petri-net-based analysis techniques used in our approach.

3.1 Event-driven Process Chains

The concept of Event-driven Process Chains (EPCs) is to provide an intuitive modeling language to model business processes. EPCs were introduced by Keller, Nüttgens and Scheer in 1992 [4]. It is important to realize that the language is not intended to be a *formal* specification of a business process. Instead, it serves mainly as a means of communication.

An EPC consists of three main elements:

Functions, which are the basic building blocks. A function corresponds to an activity (task, process step) which needs to be executed. A function is drawn as a box with rounded corners.

Events, which describe the situation before and/or after a function is executed. Functions are linked by events. An event may correspond to the postcondition of one function and act as a precondition of another function. Events are drawn as hexagons.

Connectors, which can be used to connect functions and events. This way, the flow of control is specified. There are three types of connectors: $\wedge$ (AND), $\times$ (XOR) and $\vee$ (OR). Connectors are drawn as circles, showing the type in the center of the circle.

Combined, these elements define the flow of a business process as a chain of events.

Functions, events and connectors can be connected with edges in such a way that

(1) events have at most one incoming edge and at most one outgoing edge, but at least one incident edge (i.e. an incoming and/or an outgoing edge),
(2) functions have precisely one incoming edge and precisely one outgoing edge,
(3) connectors have either one incoming edge and multiple outgoing edges, or multiple incoming edges and one outgoing edge, and
(4) on every path, functions and events alternate (in between two functions, there has to be an event, and vice versa).

The EPC shown in Figure 2 contains 9 functions, 17 events, and 9 connectors (3 $\wedge$, 5 $\times$, 1 $\vee$). If an EPC is ready to be verified (the event in the upper left corner of Figure 2), a reduction rule has to be applied. Note that any non-trivial EPC has to contain functions. Therefore, it is safe to assume that initially some reduction rules can be applied. After we have applied the reduction rules, we have to make a choice. Either more reduction rules can be applied, or the EPC is trivial, or the EPC is not trivial. Depending on the situation at hand, we follow some path through the EPC, until we reach the end.

From the definition of an EPC it is clear that a process always starts when a certain event occurs. Such an event should be one of the initial events, i.e., one of the events without incoming edges (cf. "EPC is ready to be verified"). After the process is finished, the events that have not been dealt with yet should be final events, i.e., events without outgoing edges (cf. "EPC is correct and executable"). If this is the case, we call the EPC *correct*.

3.2 Petri nets

Petri nets are a formal language that can be used to specify processes. Since the language has a formal and executable semantics, processes modeled in terms of a Petri net can be executed by an information system. For an elaborate introduction to Petri nets, the reader is referred to [9,10,13]. For sake of completeness, we mention that the Petri nets we use in this paper correspond to a classic subclass of Petri nets, namely Place/Transition nets.

A Petri net consists of two modeling elements:

Transitions, which typically correspond to either an activity (cf. an EPC function) which needs to be executed, or to a "silent" step (cf. an EPC AND connector) that takes care of routing. A transition is drawn as a rectangle.

Places, which are used to define the preconditions and postconditions of transitions (cf. an EPC event). A place is drawn as a circle.

Transitions and places are connected through directed arcs in such a way that (i) places and transitions have at least one incident edge and (ii) in every

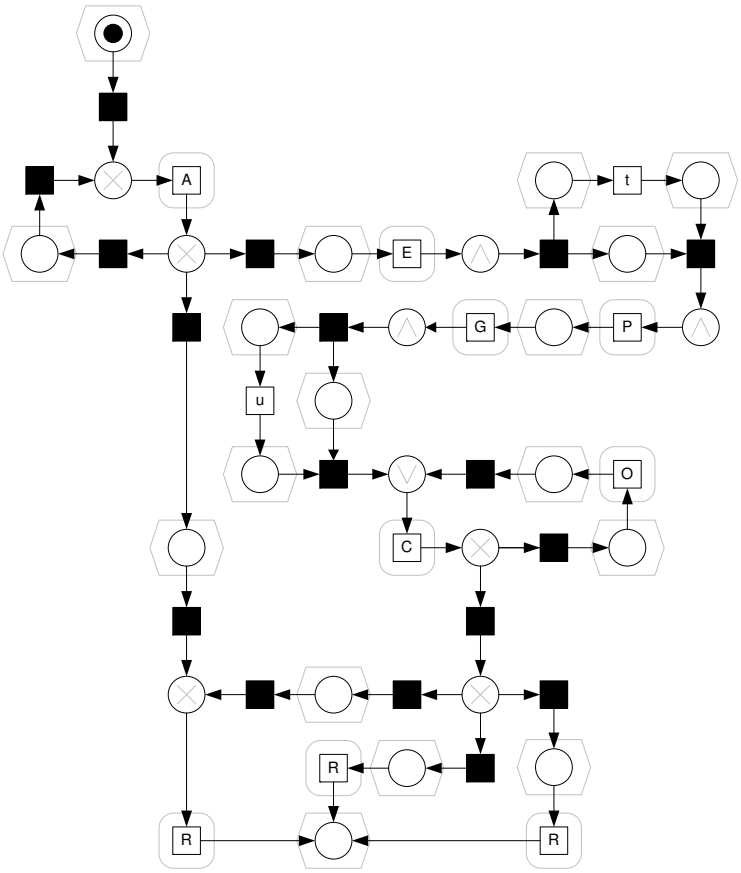

Fig. 3. A Petri net resembling the EPC shown in Figure 2

path, transitions and places alternate (no place is connected to a place and no transition is connected to a transition).

Figure 3 shows an example Petri net. This Petri net resembles the EPC shown in Figure 2, which is visualized in Figure 3. For ease of discussion, we have abbreviated the names of the EPC functions to A (Apply Reduction Rule), E (Initial Events are known), P (Transform to Petri net), G (Calculate state space, or Generate state space), C (Color the state space), O (Remove OR-transitions if allowed), and R (Check Result). Furthermore, as we explain later on, we have added transitions t and u, and have folded the three final events at the bottom of Figure 2 onto one place.

To denote the state of a process execution the concept of tokens is used. A token is placed inside a place to show that a certain condition holds. In Figure 3 one place is marked with one token. Each place can contain arbitrarily many of such tokens. If a transition execution occurs (or *fires*), one token is removed from each of the input places and one token is produced for each of the output places. Note that this restricts the behavior in such a way that a transition can only occur when there is at least one token in *each* of the input places. The distribution of tokens over the places is called a *state*, better know as

marking in Petri net jargon.

As indicated the Petri net in Figure 3 and the EPC in Figure 2 resemble each other. By playing the "token game" one can indeed verify that this is the case.

3.3 Analysis techniques

Petri nets can be used as executable specifications of business processes, i.e., their unambiguous semantics can be used to enact business processes. Moreover, it is possible to use a variety of analysis techniques answering different questions. In this subsection we discuss some of the classical questions and briefly describe two analysis techniques: state spaces and invariants.

3.3.1 State space

Given a marked Petri net, i.e., a Petri net and some initial marking, an interesting question is which markings can be reached. That is, what is the set of *reachable* markings (also called the state space) for that marked Petri net? Related to this is the question whether the state space is finite, i.e., whether the number of reachable markings is finite. If this is the case, the marked Petri net is called *bounded*. It has been proven that a marked Petri net is bounded if and only if for every place we can find some upper bound on the number of tokens inside that place. If this upper bound is 1, then the marked Petri net is called *safe*. A marked Petri net is called *live* if every transition can be fired from any reachable marking (possibly after some other transitions have been fired first). Although the state space of a bounded Petri net is finite, it can be too big for any computer to compute. Therefore, alternative analysis techniques with better complexity have been developed, including the so-called place and transition *invariants*.

3.3.2 Invariants

A *place invariant* assigns a weight to each place such that no firing of a transition changes the "weighted token sum", where the weighted token sum is defined as the sum of all tokens multiplied by the weights of the corresponding places. Note that place invariants are structural, i.e., they do not depend on the initial state. Place invariants correspond to conservation laws. A place invariant is called semi-positive if it does not assign negative weights to transitions.

Transition invariants are the dual of place invariants. A transition invariant assigns a weight to each transition such that if every transition fires the spec-

ified number of times, the initial state is restored, where negative weights correspond to "backward firing" of the transition. For our approach, we only consider *semi-positive transition invariants*. The net effect of executing every transition as many times as indicated by a semi-positive transition invariant is zero. Note that a transition invariant does not specify any order. Moreover, a transition invariant does not state that it is indeed possible to fire each transition as many times as indicated by the transition weights. This a direct consequence of the fact that invariants do not depend on a marking.

A base of invariants can be computed quite easily (polynomial time w.r.t. the number of places and transitions) [9–11]. However, the computation of a minimal set of semi-positive invariants is more complex (exponential time w.r.t. the number of places and transitions) [30].

3.4 Workflow nets, soundness, and relaxed soundness

In this paper, we mostly consider *workflow nets* (WF-nets). WF-nets are a subclass of Petri nets tailored towards workflow modeling and analysis. A WF-net has one source place (no incoming arcs, usually denoted i), one sink place (no outgoing arcs, usually denoted o), and all places and transitions are covered by the paths from i to o. Note that the Petri net shown in Figure 3 is a WF-net, because we added transitions t and u and have folded the three final events at the bottom of Figure 2 onto one place. Based on WF-nets, correctness notions such as soundness [2,6], generalized soundness [15] and relaxed soundness [7,8] have been defined.

3.4.1 Soundness

For our approach, the notions of soundness and relaxed soundness are highly relevant, therefore we describe these in more detail. Place i is the entry point for new cases (i.e., process instances), while place o is the exit point. Ideally, every case that enters the WF-net (by adding a token to place i) should exit it exactly once (by removing a token from place o) while leaving no references to that case behind in the WF-net (no tokens should be left behind). Furthermore, every part of the process should be viable, that is, every transition in the corresponding WF-net should be executable. Together these properties correspond to the notion of soundness [2,6], i.e., a WF-net is sound if and only if:

- From every marking reachable from $[i]$, the marking $[o]$ is reachable (completion is always possible).
- If in some marking M reachable from $[i]$ the place o is included in M, then $M = [o]$ (completion is always proper).

- Every transition is included in at least one firing sequence starting from $[i]$ (no dead transitions).

Note that $[i]$ denotes the state with a token in place i and $[o]$ denotes the state with a token in place o. The WF-net shown in Figure 3 is sound.

Some verification techniques require the addition of an extra transition connecting the sink place back to the source place. Such a *short-circuited* WF-net can be used to express soundness in terms of well-known Petri-net properties: A WF-net is sound if and only if its short-circuited net is live and bounded [31]. Liveness and boundedness are two well-known properties supported by a variety of analysis tools and techniques [9,10,13]. This transformation is based on the observation that an execution path that moves a token from place i to place o corresponds to a cyclic execution path in the short-circuited net: By executing the short-circuiting transition once, the token is back in place i. If we short-circuit the WF-net shown in Figure 3, the sink place is connected to the source place using a new transition (cf. Figure 4 for the short-circuited net of the Petri net shown in Figure 3).

3.4.2 Relaxed soundness

In some circumstances, the soundness property is too restrictive. Usually, a designer of a process knows that certain situations will not occur. As a result, certain execution paths in the corresponding WF-net should be considered impossible. Thus, certain reachable states should be considered unreachable. Note that in the verification process we are often forced to abstract from data, applications, and human behavior, and that it is typically impossible to model the behavior of humans and applications. However, by abstracting from these aspects, typically more execution paths become possible in the model. The notion of *relaxed soundness* [7,8] aims at dealing with this phenomenon. A WF-net is called relaxed sound if every transition can contribute to proper completion, i.e., for every transition there is at least one execution of the WF-net starting in state $[i]$ and ending in state $[o]$ which involves the execution of this transition. A WF-net is said to be relaxed sound if all transitions are relaxed sound.

As mentioned before, every case that enters a WF-net should exit it exactly once while leaving no references to that case behind in the WF-net (no tokens should be left behind). Thus, the ultimate goal of a WF-net is to move from place i to place o. The notion of relaxed soundness brings this goal down to the level of transitions: every transition should aid in moving a token from place i to place o. A transition that cannot aid in moving a token from place i to place o cannot help the WF-net in achieving its goal. Hence, such a transition has to be erroneous.

3.4.3 Transition invariants

By definition, every relaxed sound transition is covered by some path from the initial marking $[i]$ to the final marking $[o]$. As a result, *every relaxed sound transition is covered by some semi-positive transition invariant in the short-circuited net*. However, this does not work the other way around since transition invariants abstract from the state of the net. Therefore, it might be possible that the transitions covered by some transition invariant cannot be executed (because some tokens are required for a transition to fire, while they are only produced later). As a result, there may be a transition that is covered by some transition invariant in the short-circuited net, but that is not covered by any execution path from state $[i]$ to state $[o]$. Nevertheless, if we are unable to generate the state space in reasonable time (i.e., it is too large to be computed), then we can use transition invariants as an approximation. Note that for every execution path from state $[i]$ to state $[o]$ the short-circuiting transition needs only to be executed once to obtain a cyclic execution path. Furthermore, note that there may be cyclic execution paths present in the WF-net itself. For these two reasons, we restrict ourselves to transition invariants where the short-circuiting transition has either weight 0 (corresponds to a cycle in the WF-net itself) or 1 (corresponds to an execution path from $[i]$ to $[o]$).

3.4.4 Example

To summarize and illustrate the issues related to (relaxed) soundness, we use Figure 4. Figure 4 shows the short-circuited variant of the WF-net shown in Figure 3. The transition that short circuits the net is the one on the bottom, that transfers the token back to the initial place. Figure 4 also shows two (semi-positive) transition invariants. One invariant assigns weight 1 to all transitions labeled x (all others have weight 0). The second transition invariant assigns weight 1 to all transitions labeled y (again, all others have weight 0). It is not hard to show that all transitions in Figure 3 can be covered by semi-positive transition invariants, which suggests (but does not prove) that the WF-net is indeed relaxed sound.

3.5 Conclusion

In this section, we introduced EPCs, Petri nets, and relevant notions such as (relaxed) soundness and invariants. In the remainder of this paper, we show the process of EPC verification. The first step is made in Section 4, where we reduce the verification problem of a large EPC to that of a smaller (and possibly trivial) EPC. In Section 5, we present the whole approach and show

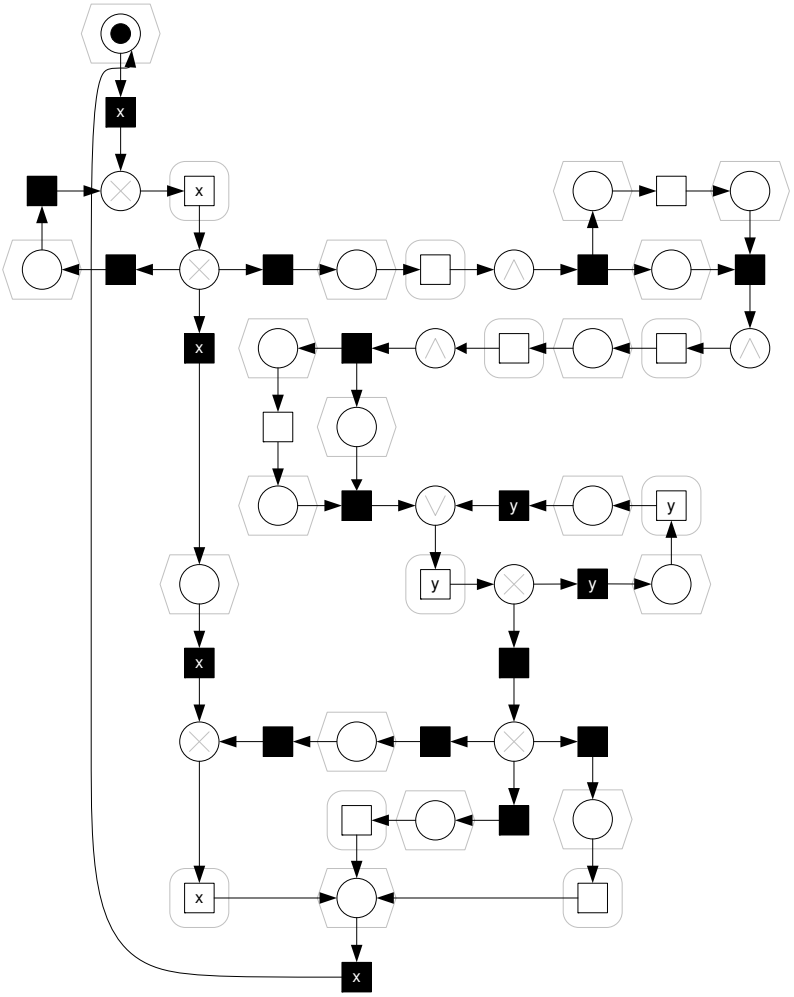

Fig. 4. Two possible transition invariants, denoted with x and y.

how we can use Petri-net-based techniques to further analyze the reduced EPC. Section 6 discusses the use of invariants if the state space is too large.

4 Reduction rules

EPCs can contain a large number of functions, events and connectors. However, for the verification of EPCs, not all of these elements are of interest. In particular, we are interested in the routing constructs that are used in the EPC, since that is where the errors can be. Furthermore, it is obvious that some constructs are trivially correct. For example, an arbitrary long sequence of events and functions is clearly correct and does not need to be considered in detail in the verification process. Moreover, if a split of some type is followed by a join of the same type, the exact semantics of the splits and joins is irrelevant as under any semantics this will be considered correct. In this section,

we introduce a set of reduction rules. These rules can be applied on any EPC in such a way that, if the EPC is correct before the reduction, then the result after reduction is correct and if the EPC is not correct before reduction, then the result after reduction is not correct, i.e. these rules are *correctness preserving*. However, we do not intend these rules to be complete. Instead, they merely help to speed up the verification process, by removing trivial parts before going to the more demanding steps in process (i.e., both in terms of computation time and human interpretation).

It is easily seen that the application of the reduction rules does not result in an EPC, since functions and events no longer alternate. However, for the process of verification, this is not a problem and we will refer to this reduced model as a *reduced EPC*.

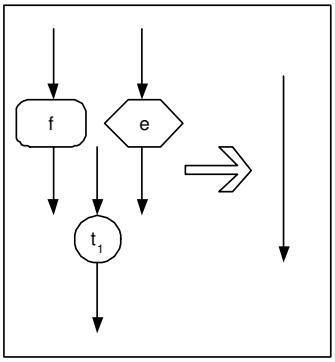

Fig. 5. Trivial construct.

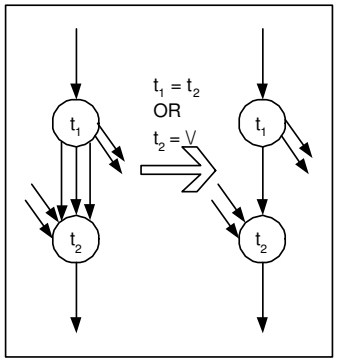

Fig. 6. Simple split/join.

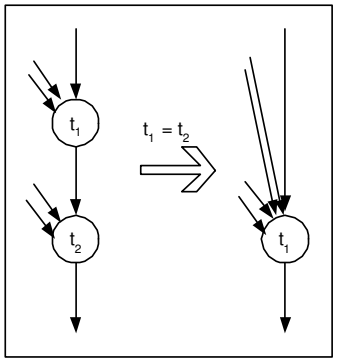

Fig. 7. Similar joins.

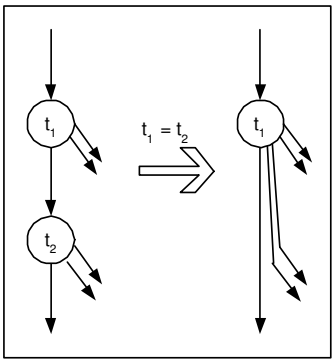

Fig. 8. Similar splits.

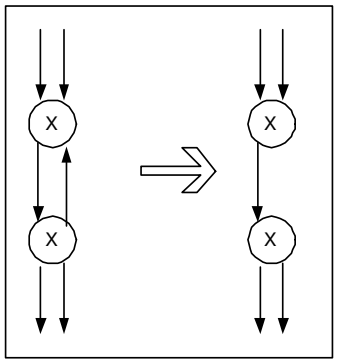

Fig. 9. XOR loop.

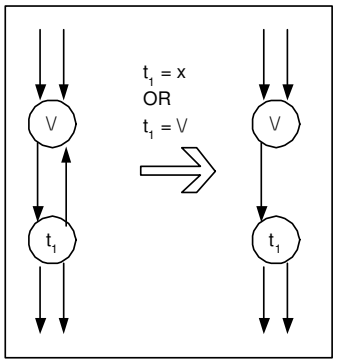

Fig. 10. Optional OR loop.

4.1 Trivial constructs

Figure 5 shows the reduction rules for trivial constructs. It shows that a function f, an event e or a connector with type t_1 with precisely one ingoing and one outgoing edge can be removed completely. As stated before, we are only interested in routing constructs. Functions, events, or connectors with only one

incoming and only one outgoing edge do not provide any routing information. Therefore, they can be removed while preserving correctness.

4.2 Simple splits/joins

Figure 6 shows the reduction rule for a split that is followed by a join connector. This rule can be applied if both connectors are of the same type (i.e. AND, OR or XOR), or if the join connector is of type OR. Again it is trivial to see that correctness is preserved.

4.3 Similar splits/joins

Figures 7 and 8 show the rules for two connectors of the same type that directly follow each other. These two connectors can then be merged into one connector of the same type. Note that syntactical restrictions of (reduced) EPCs do not allow for more than one edge between the first and the second connector, since connectors are either a split or a join and never both.

4.4 Loops

Finally, figures 9 and 10 show two very similar reduction rules that deal with loops. In these cases correctness preservation is less straightforward. For Figure 9 it is clear that removing the possibility to loop back is correctness preserving, because the "backward arc" does not introduce any new states. Figure 10 shows an optional rule. Unlike the others it is not correctness/incorrectness preserving in any situation like the first five rules. The rule assumes that the intended semantics is safe (i.e., no multiple activations of functions and no events that are marked multiple times). This implies that if t_1 is an OR-join either the backward arc is taken or any combination of the other arcs.

4.5 Example

Figure 11 shows the result of applying the reduction rules to the EPC of Figure 2. The resulting reduced EPC does not contain any functions, and only some of the connectors from the original EPC. We know that none of the reduction rules will make the reduced EPC incorrect if the original was correct, and they will not make the reduced EPC correct if the original was

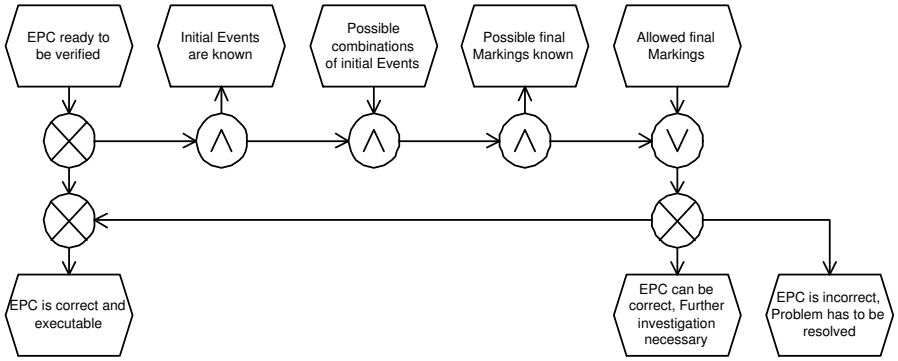

Fig. 11. Reduced EPC for the verification process.

incorrect. Therefore, we can now proceed with the verification process using this reduced EPC and the result can directly be translated back to the original EPC.

5 Verification of the reduced EPC

In the previous section, we introduced reduction rules for EPCs in such a way that we can use a reduced EPC for the verification process. In this section, we will translate the reduced EPC into a Petri net. This is also the part of the verification process where user interaction plays an important role. The user has to provide us with possible combinations of initial events. These combinations are then translated into initial markings of the Petri net. By calculating the state space, we can then provide the user with all possible combinations of final events that can happen. It is again up to the user to divide those into a set of desired and undesired combinations. Using this information we go into the final stage, where we use a simple coloring algorithm on the state space to decide whether the reduced EPC is correct. This is then translated back to the original EPC.

In this section we will use Figure 2 to describe the overall verification approach in more detail and focus on the second phase of the algorithm where the Petri-net mapping for the reduced EPC is used for deciding (relaxed) soundness. Note that the whole process shown in Figure 2 is implemented in the context of the ProM framework (cf. Section 7).

5.1 User interaction 1

As we stated before, the process of EPC verification relies on user interaction at two points. The first point is where the user has to specify which combinations of initial events can appear to initiate the process described by the EPC

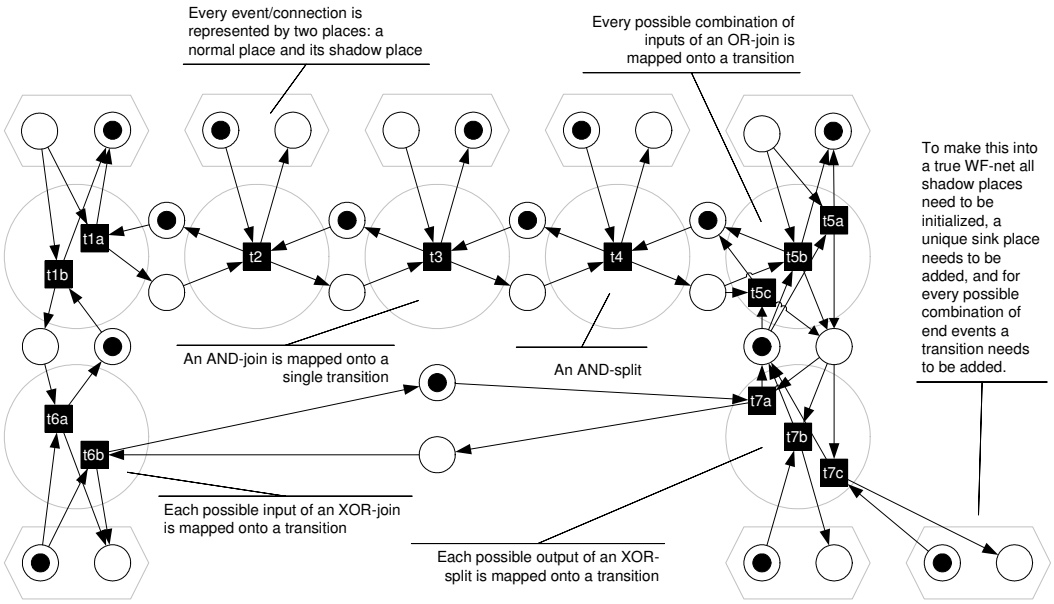

Fig. 12. Petri net translation of the reduced EPC.

(i.e., event "Possible combinations of initial Events" in Figure 2). Using this information from the user, we can calculate which initial markings are possible for the Petri net that we will build. If we consider the example from Figure 2, then there is only one combination of events that can start the process. This is the combination of the events "EPC ready to be verified", "Possible combinations of initial events" and "Allowed final markings". It has to be noted that the events "Possible combinations of initial events" and "Allowed final markings" can only appear as a consequence of some choice that was made in the model. However, these causalities are not expressed in the EPC, and therefore they cannot be known to the verification system. As can be seen in the procedure shown in Figure 2, we are now ready to transform the EPC into a Petri net.

5.2 Translation to Petri net

Many authors have described algorithms to translate EPCs to Petri nets. In this paper, we use a modified version of the translation proposed in [7,8]. The translation presented there gives a translation into normal Petri nets, whereas we use the same translation algorithm, but enforce the result to be a safe Petri net. The reasons for doing so are twofold. First, events in EPCs either have occurred or not. Therefore, it makes no sense to map an event onto a place that can contain multiple tokens. Second, the state space of a safe Petri net is more likely to be small, hence, it is more likely that we will be able to construct the state space. Furthermore, enforcing safeness is similar to the interpretation of EPCs in [25].

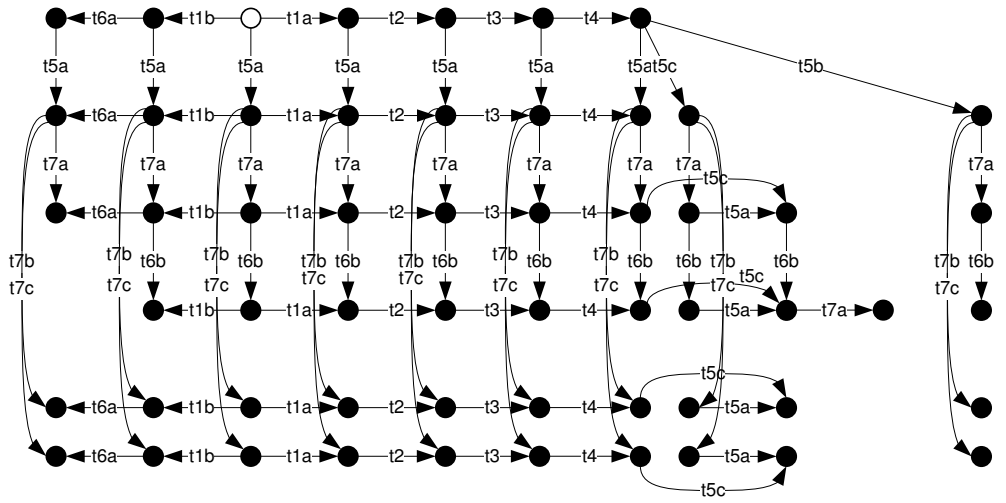

Fig. 13. State space of the Petri net as shown in Figure 12

To enforce safeness, we duplicate all places with arcs in the reversed direction, i.e., every "normal place" gets a "shadow place" that initially contains one token. Each time a token is produced for the normal place, a token is consumed from the shadow place. If the token on the normal place is consumed, a token is returned to the shadow place. Therefore, the sum of tokens in a normal place and it's shadow place is always 1. This way the resulting net is forced to be safe.

The result of the transformation process for our running example of Figure 2 is shown in Figure 12. Note that in the layout of the Petri net the reduced EPC from Figure 11 is visible. Also note that, by definition, the reduced EPC only contains events and connectors, and that the EPC cannot be reduced completely. Again each event is modelled by two places (i.e., a normal place and shadow place) and connectors are modelled by a small network of transitions. Note that this is not yet a WF-net. However, it can be used to generate the state space.

Using the combinations of initial events calculated in the previous step, we are ready for the state space generation.

5.3 State space generation

Figure 13 shows the state space of the Petri net as shown in Figure 12. For ease of reference, we have highlighted the initial state (the open dot).

As we discussed in Section 3.3, a state space of a bounded Petri net is always finite. Still, it may be too complex to be constructed (within reasonable time). However, given the fact that the Petri net results from a reduced EPC and

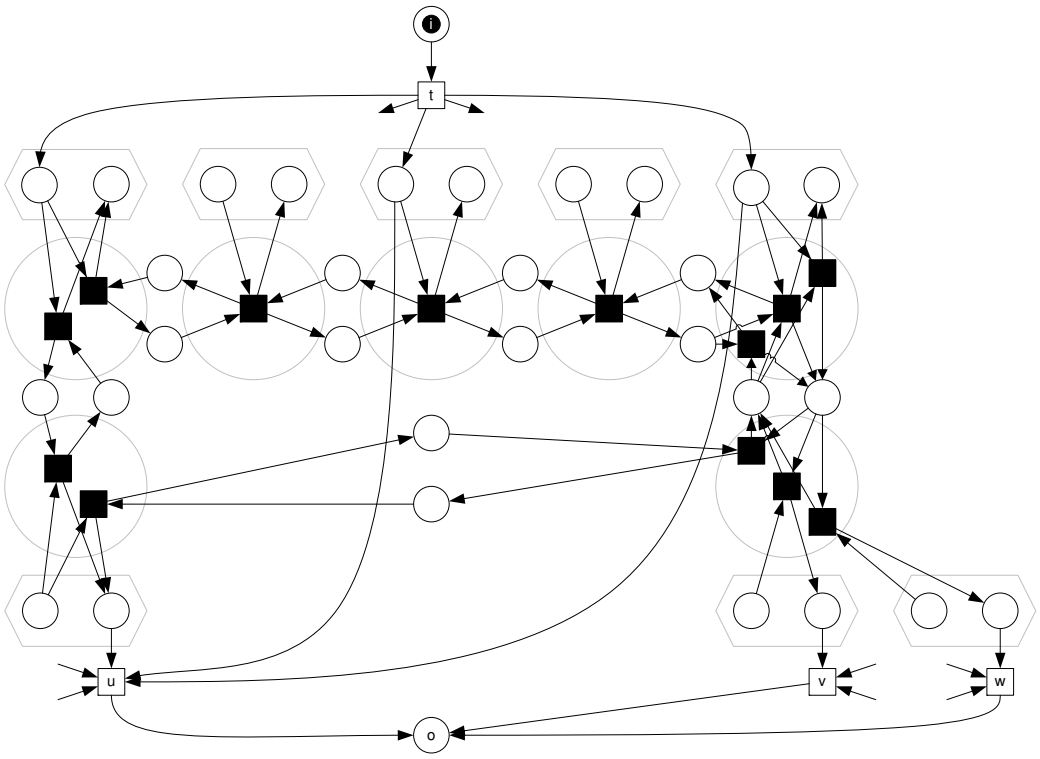

Fig. 14. The essence of the resulting WF-net

that the Petri net is safe by construction, we expect to be able to construct the state space for the vast majority of EPCs found in practice. Moreover, based on our experiences so far (with EPCs up to 100 functions and events), it seems possible to construct the state space without any problems for all EPCs encountered in practice (especially after reduction). Nevertheless, as discussed in Section 3.4, it is also possible to use transition invariants if the state space cannot be constructed. We discuss this in detail later.

5.4 User interaction 2

Now that we have calculated the state space, we are able to provide the user with details about the possible outcomes of the process. In our example, there are many different outcomes that were not intended to be there. The reason for this is the informal definition of the OR-connector in the process. From this paper it will become clear that for the OR-connector you either have both events "Ready for analysis" and "Allowed final markings", or you have "All allowed OR transitions are removed" (cf. Figure 3, where we already took this into account). However, from the description of the EPC, this is not entirely clear. Therefore, we require the user to select those possible outcomes (set of final events) that correspond to correct executions of the process. In this case,

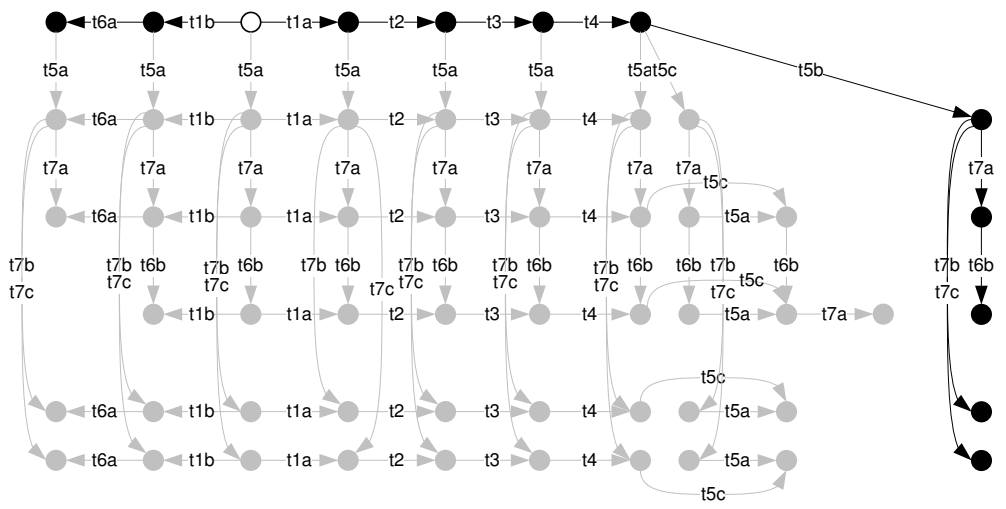

Fig. 15. Colored state space of the Petri net as shown in Figure 12

the user selects the three final states in the rightmost column of the state space.

Using this information about the allowed final markings, we can transform the Petri net generated earlier into a WF-net, i.e., a Petri net with a single source place and a single sink place. This implies that we can apply all the notions and results presented in Section 3.4. Figure 14 shows the essence of this WF-net: transitions t, u, v, and w have been added. For sake of readability, we restricted ourselves to the most relevant arcs to and/or from these transitions, i.e., we have not drawn all arcs from and/or to these transitions. Transition t also has to initialize all shadow places, while transitions u, v, and w also have to remove all remaining tokens from these shadow places. Note that transition u needs to remove the tokens corresponding to the initial events "Possible combinations of initial events" and "Allowed final markings", as these two initial events are not needed if the EPC reduces to the trivial EPC.

5.5 The decision process

Finally, we have all the ingredients we need to decide whether the EPC is correct. We have a state space, of which we know all initial states and all allowed final states. The first step toward the final decision is to color everything from the state space that appears on a path from a initial state to one of the allowed final states. The colored part of the state space then describes all the behavior that the user allows. Then, we look for all transitions that do not have a colored edge in the state space. We call those transitions "not covered". Figure 15 shows the colored state space, where the black parts represent the allowed behavior.

In principle, transitions that are not covered show that there is possibly incorrect behavior. Translating this back to an EPC would result in saying that a certain connector is used incorrectly. This is indeed the case for connectors of type XOR and AND. However, for connectors of type OR, we need to perform an additional step. When people use connectors of type OR, they do not necessarily want all the possible behavior to appear. As mentioned earlier, the designer knows that the OR-connector in the example EPC will either use the event "All allowed OR transitions are removed" or will synchronize and use the events "Ready for analysis" and "Allowed final Markings". As a result, the user can tell that the transitions t5a and t5c cannot occur in practice, and they should be removed from the Petri net. After all such transitions have been removed from the Petri net, the state space is then recalculated without the need for user interaction. Again, the coloring process is repeated and finally, we can provide the final answer:

The EPC is correct (i.e., the corresponding WF-net after reduction is *sound*). This is the case if the entire state space is colored. If the EPC is correct, then it is always possible to execute the process without ending up in some undesired state.

The EPC can be correct (i.e., the corresponding WF-net after reduction is *relaxed sound but not sound*). This is the case if the state space is not entirely colored, but all transitions are covered. This result tells the designer that the EPC can be executed, but special care has to be taken to make sure that an execution does not end up in some undesired state. Consider for example Figure 16, where the choices following functions A and B need to be synchronized to get proper execution paths. However, this synchronization is not made explicit in the model.

The EPC is incorrect (i.e., the corresponding WF-net after reduction *not relaxed sound*). This is the case when not all transitions are covered. Basically this means that there is some part of the EPC that cannot be executed without running into some undesired behavior. Consider for example Figure 17, where an obvious modeling mistake is depicted.

Note that in this final phase we heavily rely on the correctness criteria defined for WF-nets (cf. Section 3.4).

Despite the fact that the Petri net is safe, the state space can still be too complex to construct, especially if many parallel branches exist in the reduced EPC. Fortunately, we can use transition invariants as shown in Section 3.4 to detect errors. In the following section we briefly discuss the way that transition invariants can be deployed.

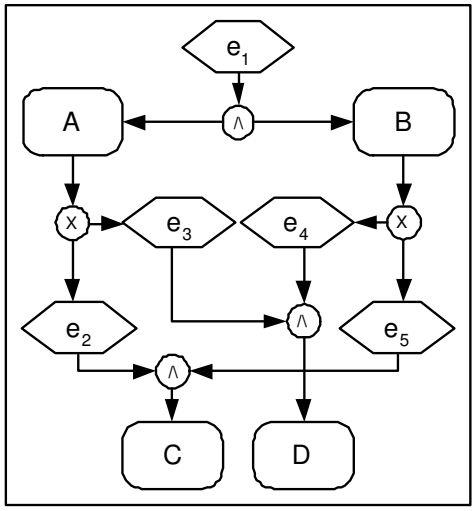

Fig. 16. An EPC with choice synchronization.

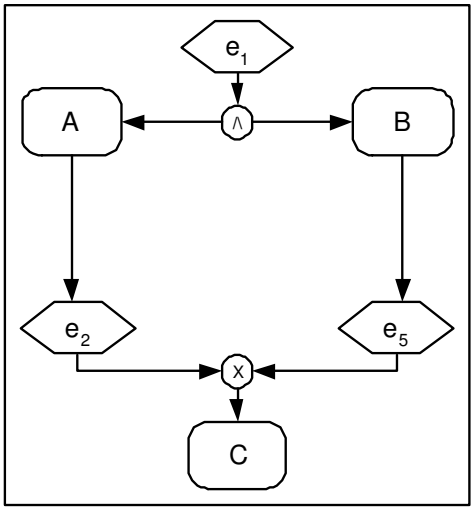

Fig. 17. An EPC with erroneous routing.

5.6 Conclusion

In this section, we have presented a step by step algorithm for the verification of EPCs. We have shown that we need user interaction on two levels, and that the resulting answer does not need to be conclusive. There is a "grey area" where the EPC can be executed correctly, but can also run into problems (i.e., relaxed soundness). This gray area is not a weakness of the verification process! Instead, it shows the difference between a conceptual modeling language such as EPCs and an executable specification in terms of a Petri net. The EPC language is an informal language aiming at discussions about the process and not intended as an executable specification. However, by making the implicit assumptions explicit, it is possible to derive an executable specification from the EPC. This way the model can be used as a starting point for the configuration of enterprise systems (e.g., in the form of an executable workflow model).

6 Using transition invariants

For well-structured EPCs (e.g., a 1-to-1 correspondence between splits and joins) the approach discussed in previous section is very effective. The reduction rules immediately remove all the complexity. However, for more complex "spaghetti-like" diagrams the reduced EPC may still be large and exhibit a lot of parallelism. Note that for an EPC with 10 functions in parallel there are at least 10! = 3628800 possible states (as there are 10! possible sequences of these functions), and that for an OR-split/join construct with 10 functions to

choose from, there are $2^{10} - 1 = 1023$ possible sets of functions that can all be executed in parallel. Hence, if the reduction rules fail to reduce the EPC sufficiently, the construction of the state space may simply take too much time. To address this problem we return to the observations made in Section 3.4.

If the state space is too complex to construct, we make use of so-called transition invariants. *Transition invariants do not require the state space to be constructed, and it is straightforward to show that a transition that cannot be covered by any of the transition invariants, can also not be covered by relaxed soundness.* Recall that every relaxed sound transition is covered by some path from the initial marking $[i]$ to the final marking $[o]$. As a result, every relaxed sound transition is covered by some transition invariant in the short-circuited net. For the analysis it suffices to focus on transition invariants where the short-circuiting transition has either weight 0 or 1. The former corresponds to a cycle in the WF-net itself (cf. the transition invariant labelled y in Figure 4), the latter to an execution path from $[i]$ to $[o]$ (cf. the transition invariant labelled x in Figure 4). Using standard algorithms [30] it is possible to calculate a set of minimal semi-positive transition invariants, and to select the relevant ones in a second step.

As discussed in Section 3.4, it might be possible that the transitions covered by some transition invariant cannot be executed (because some tokens are lacking). As a result, there may be a transition that is covered by some transition invariant in the short-circuited net, although it is not covered by any execution path from state $[i]$ to state $[o]$. As an example, consider the reduced EPC as shown in Figure 18.

The thick path shown in this figure corresponds to a transition invariant in the corresponding short-circuited Petri net. As a result, the two AND-connectors in the middle are covered by transition invariants. However, these connectors are not covered by relaxed soundness (as there is no executable sequence that uses these connectors, because the top connector would block). This example shows that if all transitions are covered by invariants, this is not a sufficient condition for relaxed soundness.

Examples such as the reduced EPC in Figure 18 are rare. In most cases invariants and relaxed soundness will yield identical results. Moreover, the designer should realize that all errors discovered using invariants are indeed errors in the EPC, but that some errors may remain undetected. As a result, we feel that our approach can also support the designer of an EPC even if the state space of that EPC is too complex to construct.

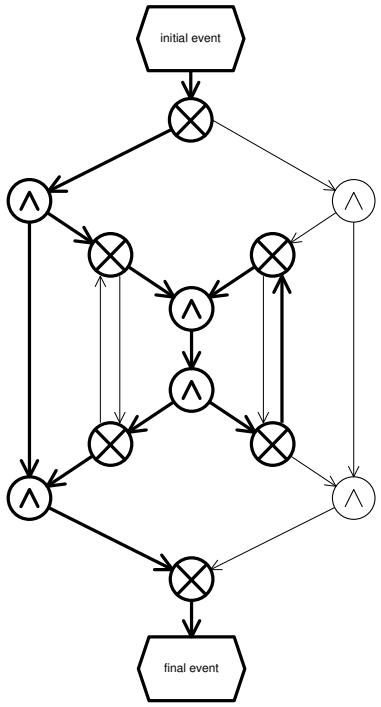

Fig. 18. Reduced EPC: invariants vs. relaxed soundness.

7 Tool

The approach described in Section 5 has been implemented as an analysis plug-in in the *ProM framework* [32]. This EPC verification plug-in fully supports the approach described in Section 5 and depicted in Figure 2: It automatically reduces the provided EPC, maps the reduced EPC onto a Petri net, constructs the state space, and does a state space analysis. Besides showing the reduced EPC it can also show the corresponding Petri net and state space. Possible problem areas in the state space are highlighted to make analysis easier for the user.

7.1 The ProM framework

As mentioned, the approach is implemented as an analysis plug-in in ProM framework. The ProM framework provides a "plugable" framework for *process mining* [33] and has been developed by the authors and other people from Eindhoven University of Technology and many other institutes [3] . Process mining aims at extracting knowledge from event logs (e.g., transaction logs in an ERP system or audit trails in a WFM system) and is closely related to Busi-

[3] Development of ProM was sponsored by the Dutch institutes NWO, STW, Beta and EIT.

ness Activity Monitoring (BAM) and Business Process Intelligence (BPI). The framework is flexible with respect to the input and output formats, and is also open enough to allow for the easy reuse of code during the implementation of new process mining ideas. Within the framework, several process modeling languages are supported, e.g., Petri nets, EPCs, heuristics nets, YAWL models. The framework also allows for conversions between different formats. For example, it is possible to load an EPC from the ARIS toolset, translate it to a Petri net for analysis purposes, and then convert the Petri net to a YAWL model that can be loaded into the YAWL workflow engine for enactment. In this context it is interesting to mention the multi-phase mining plug-in [34]. This plug-in can discover EPCs based on event logs without any explicit process information (i.e., on the basis of "raw data"). The resulting EPC models can be loaded in the ARIS toolset. Moreover, the logs and intermediate results can be exchanged with the ARIS Process Performance Monitor (ARIS PPM). Although the initial focus of the ProM framework was on process mining (in particular process discovery, i.e., constructing process models based on event logs), over time several analysis plug-ins have been developed. For example, the main part of the functionality of Woflan [16] has been embedded into a ProM analysis plug-in.

The ProM framework and related plug-ins (including the EPC verification plug-in) can be downloaded from `www.processmining.org`.

7.2 A real-life example

To show the functionality of the EPC verification plug-in we use a real-life example. The process is from the Trade Department of a large Dutch bank. (We cannot disclose the name of the bank for reasons of confidentiality.) We applied our approach and tool to several processes within this bank, including the trade execution process. First, we imported the EPC into the ProM framework using another plug-in: the ARIS graph format import plug-in, and we selected the trade support process. Figure 19 shows the result. Next, we started the EPC verification plug-in from the "analysis" menu in the menu bar. The left-hand side of Figure 20 shows the result.

The plug-in informed us that there was only one initial event. Therefore, we selected as possible initial event sets the set containing only this initial event, and continued. Behind the scenes, the plug-in now mapped the EPC onto a marked Petri net, constructed the state space for that Petri net, and computed the possible final states for that Petri net. Finally, the plug-in showed these final states. Figure 21 shows the result.

At this point, we had to decide which final states were desired (keep) and

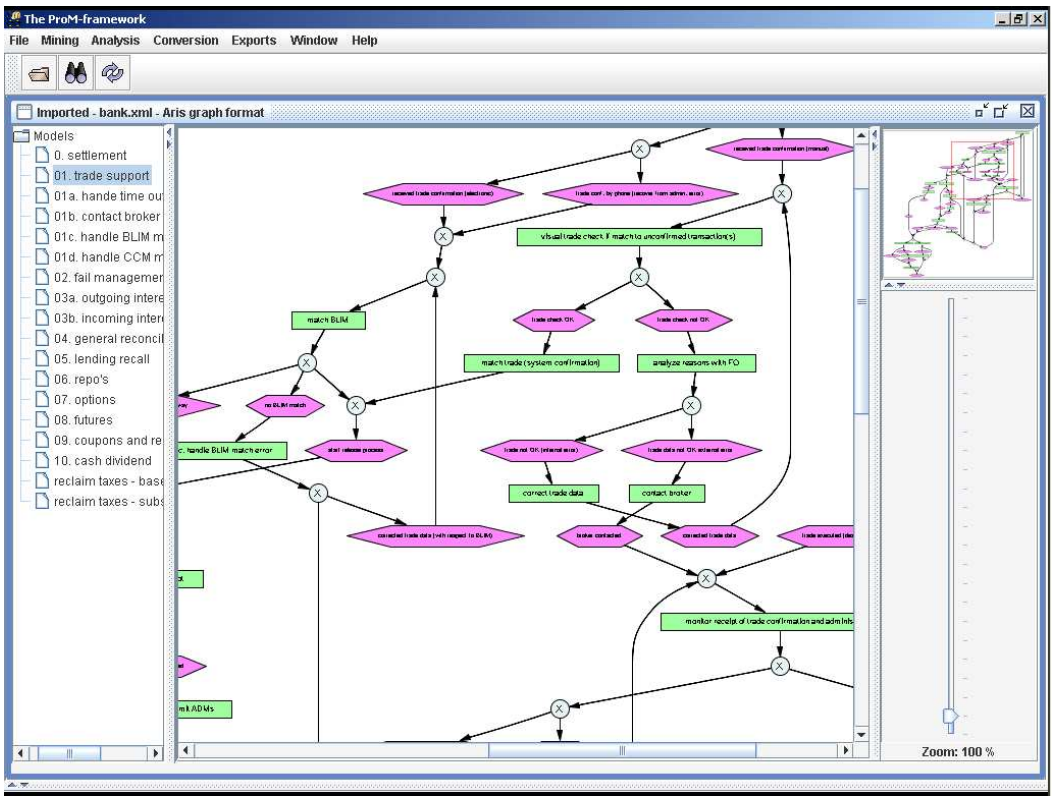

Fig. 19. The trade process imported in the ProM framework.

which were undesired (ignore). Note that the plug-in already had categorized the final states based on the fact whether the final states included non-final events: if a final state includes a non-final event, then the plug-in proposes to ignore that final event, since this indicates a process that terminated while some events have not been dealt with. We simply accepted the categorization as proposed by the plug-in, and continued. This resulted in the following message: "The EPC can be correct, but allows for undesired behavior".

At this point in time, the process owner at the bank informed us that two of the kept proposed final states were in fact undesired. As a result, we moved these two states to the undesired list. This immediately lead to the following message: "The EPC contains structural errors".

As a result, the identified problem parts were highlighted in ProM. These highlighted parts are shown in Figure 22. From this information, we deduced that the EPC contained a problem w.r.t. synchronization and choice: Two parallel branches were started, but from one of these branches it was possible to jump back and to start both branches again. As a result, the other branch could be started over and over again.

Thus, using the EPC verification plug-in, we were able to identify the problem areas, and from that we were able to conclude that the model contained errors

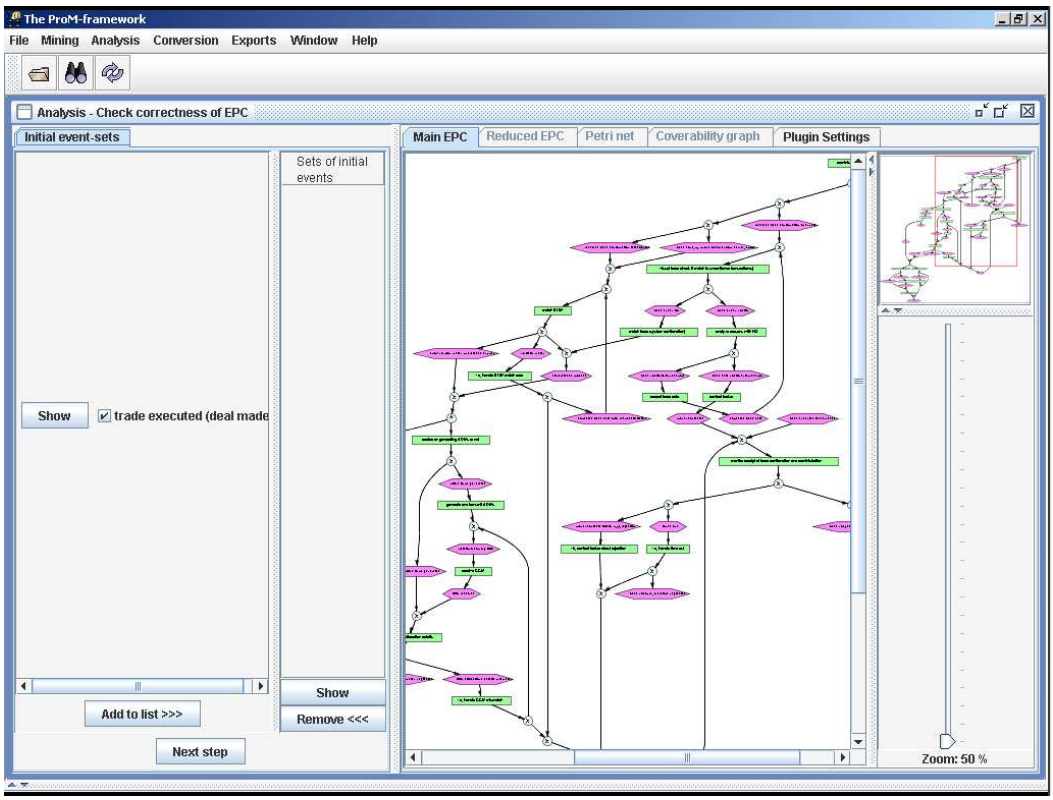

Fig. 20. The trade process after starting the EPC verification plug-in.

that should be corrected. Using this information, we could correct the trade execution EPC. As a result of this test on their trade execution process, the process owners of the EPC decided to start using the plug-in by themselves in the future.

After presenting our verification approach and the EPC verification plug-in, we now focus on the verification of the reference models present in SAP R/3 and ARIS for MySAP.

8 Verification of the SAP reference models

The application of the verification approach presented in Section 5 is based on a basic assumption: It assumes that the designer of a model has a good understanding of the actual business process that was modeled, and that he knows which combinations of events may actually initiate the process in real life. Typically, reference models are used by consultants that do indeed have a good understanding of the process under consideration. Besides, they know under what circumstances processes can start, and which outcomes of the execution are desired and which are not. Therefore, our approach seems to be well suited for the verification of the SAP reference models. Before presenting

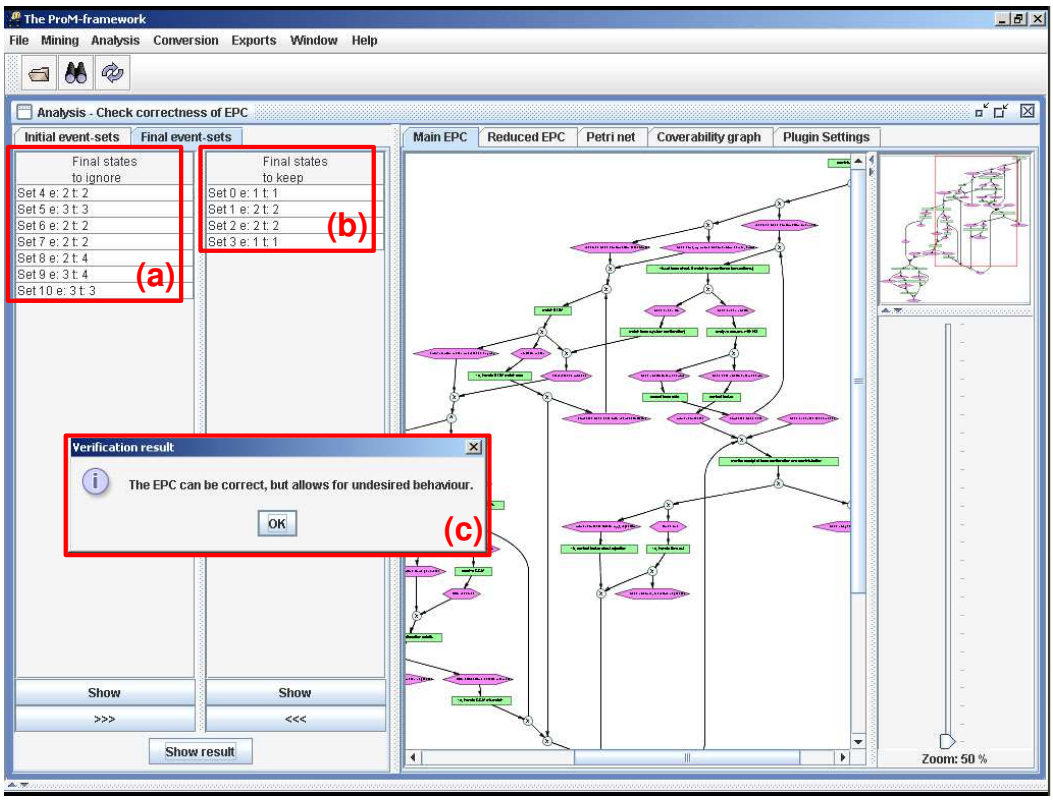

Fig. 21. The trade process with its undesired (a) and desired (b) final states and the verification result (c).

the verification of the SAP reference models we first provide some background information on reference models in general and the SAP reference models in particular.

8.1 SAP R/3 Reference models

Reference models [35–43] are generic conceptual models that formalize recommended practices for a certain domain [37,38]. Reference models accelerate the modeling process by providing a repository of potentially relevant business processes and structures. With the increased popularity of business modeling, a wide and quite heterogeneous range of purposes can motivate the use and development of reference models. These purposes include software development, software selection, configuration of enterprise systems, workflow management, documentation and improvement of business processes, education, user training, auditing, certification, benchmarking, and knowledge management [40].

Literature (e.g., [39,40]) suggests that we can distinguish two types of reference models: industry models and application models. Industry reference models are generally higher level models and they aim to streamline the design of

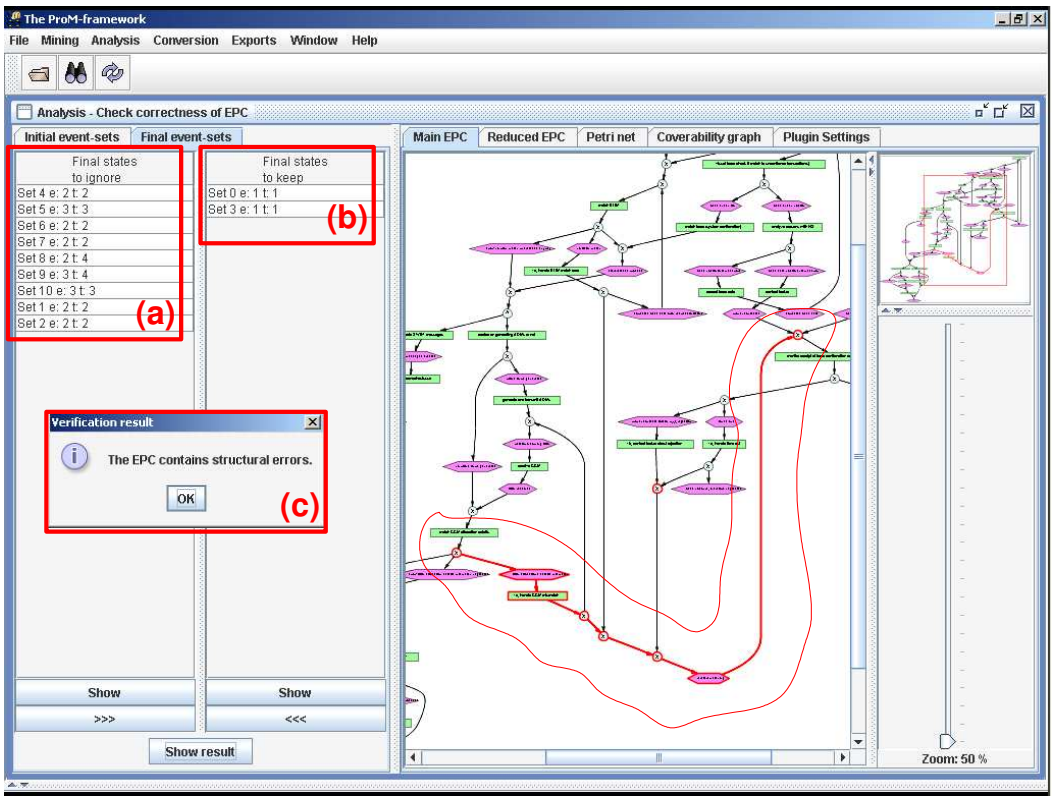

Fig. 22. The trade process with updated undesired (a) and desired (b) final states, the new verification result (c) and highlighted part of the process.

enterprise-individual (particular) models by providing a generic solution. Application reference models describe the structure and functionality of business applications including enterprise systems. In these cases, a reference model can be interpreted as a structured semi-formal description of a particular application. This application can then be seen as an existing off-the-shelf-solution that supports the functionality and structure described in the reference model.

Rosemann and Van der Aalst explain in [40] that application reference models tend to be more complex than industry reference models. They explain that the SAP reference model is one of the most comprehensive models [36]. Its data model includes more than 4000 entity types and the reference process models cover more than 1000 business processes and inter-organizational business scenarios. EPCs have been used for the design of the reference process models in the ARIS for MySAP database that we consider in this paper. EPCs also became the core modelling language in the Architecture of Integrated Information Systems (ARIS) [41].

In the ARIS for MySAP reference database, there are hundreds of EPCs that can be used in many different situations, from “asset accounting” to “procurement” and “treasury”. Since we cannot discuss all these models here, we first focus on one of the modules that can be considered to be a representative

subset of all reference models, namely "procurement". This is a set of some 40 EPCs, all in the area of procurement. They describe processes for

(1) internal procurement,
(2) pipeline processing,
(3) procurement of materials and external services,
(4) procurement on a consignment basis,
(5) procurement via subcontracting,
(6) return deliveries, and
(7) source administration.

After considering the "procurement" module we discuss some more general observations based on an analysis of the other top-level reference models in the ARIS for MySAP database.

8.2 Procurement module

As indicated in the previous subsection, we focus on the procurement module of the ARIS for MySAP reference model database, since it can be seen as a representative subset of all reference models. The procurement module contains several sub-modules and we analyzed all the models from these modules using the approach presented in Section 5. Surprisingly, already in the first model (Internal Procurement) there were structural errors. In Figure 23, we show a screenshot of the EPC verification plug-in while analyzing the Internal Procurement EPC. It clearly shows that an *AND*-split is later joined by an *XOR*-join. Recall Figure 17, where we have shown that this is incorrectly modeled. As a result, if this model would *not* be repaired, payments could be made for goods that were never received. Obviously, this is not desirable. In this case, the problem can easily be repaired. If the *XOR*-join at the bottom is changed into an *AND*-join, the model is correct: The EPC cannot be reduced to the trivial EPC but the corresponding WF-net is sound.

The results of our analysis of the whole procurement module are presented in Table 1, which contains three columns. The first column shows the name of the module. The second contains the verification result. We use "I" for incorrect models (i.e., the corresponding Petri net is not relaxed sound), "S" for syntactically correct models (i.e., soundness can be decided by just applying the reduction rules) , and "C" for semantically correct ones (i.e., sound or relaxed sound). The final column gives short explanation of the error found.

In addition, we applied the analysis using transition invariants. From which, we again were able to conclude that the processes "Internal Procurement" and "Procurement via Subcontracting" were incorrect, i.e., the use of invariants (rather than constructing the state space) also allowed us to find errors. This

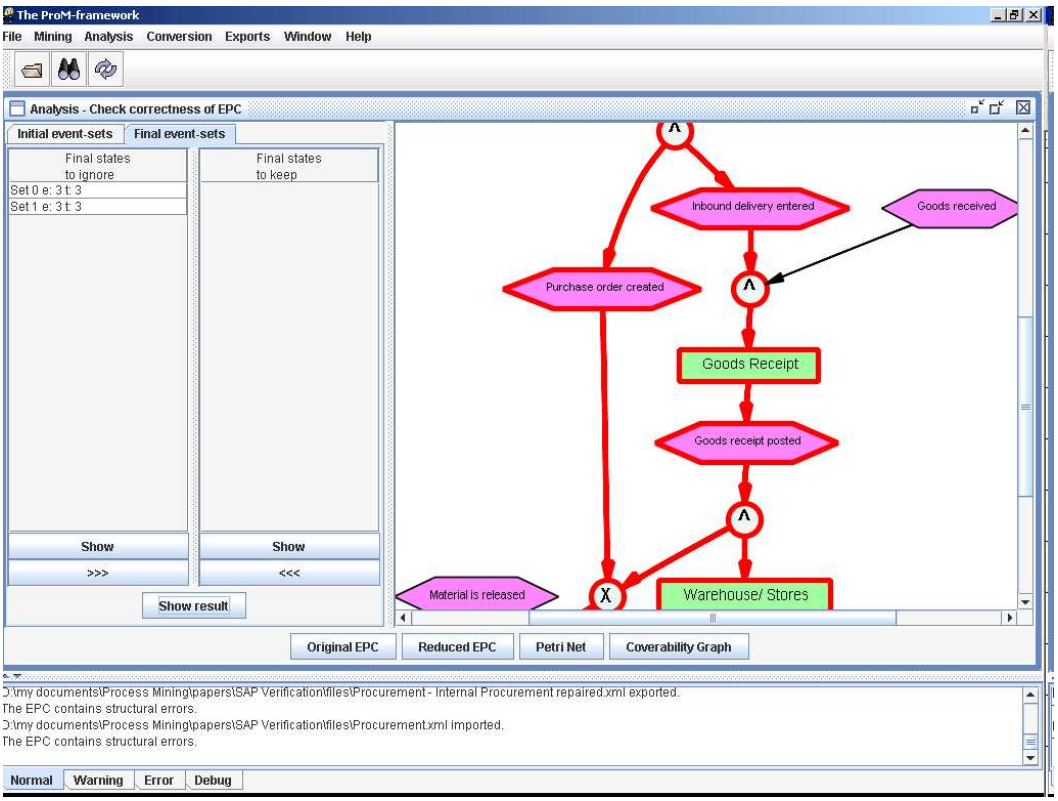

Fig. 23. Fragment of the "Internal Procurement" EPC showing a structural error.

strengthens our belief that examples as shown in Figure 18 are rare, and that, in general, the technique with transition invariants is applicable in practice. Using this technique, we were also able to conclude that an OR-join in the "Outline Purchase Agreements" (from the Module "Source Administration") could be replaced by an AND-join without changing the allowed behaviour of the process.

8.3 Further analysis of the SAP reference models

From the previous section it seems that we can conclude that most errors are made in the higher level models. Using this as a guide, we tried to find problems in the entire set of reference models. In fact, in the high level models, it is not hard to find these mistakes. These high level models are usually more complex then the lower level models (i.e. they contain more functions, events and connectors). Therefore, errors are more likely to be introduced there. Instead of giving a detailed list of all the errors we have found, we would like to mention three observations that we made during this guided model selection.

Table 1
Results for the procurement module

Module name	Result	Implication of the problem
Internal Procurement	I	Payments can be done for goods never received.
↪ Goods Receipt	C	
↪ Invoice Verification	C	
↪ Purchase Requisition	C	
↪ Purchasing	C	
↪ Warehouse stores	C	
Pipeline Processing	C	
↪ Invoice Verification	C	
↪ Pipeline Withdrawal	C	
Materials and External Services	I	An invoice can be paid for ordered goods (not services) that have not yet been delivered.
↪ Goods Receipt	C	
↪ Invoice Verification	C	
↪ Purchase Requisition	C	
↪ Purchasing	C	
↪ Service Entry Sheet	C	
↪ Transportation	C	
↪ Warehouse/Stores	C	
Procurement on a Consignment basis	C	
↪ Goods Receipt	C	
↪ Invoice Verification	C	
↪ Purchase Requisition	C	
↪ Purchasing	C	
↪ Warehouse/Stores	C	
Procurement via Subcontracting	I	An invoice that is received twice will be paid twice.
↪ Goods Receipt	C	
↪ Invoice Verification	C	
↪ Provision of Components	C	
↪ Purchase Requisition	C	
↪ Purchasing	C	
↪ Transportation	C	
↪ Warehouse/Stores	S	When materials are simultaneously placed into the stock and removed from it, erroneous behavior occurs. Operational procedures should avoid this.
Return Deliveries	C	
↪ Invoice Verification	C	
↪ Outbound Shipments	C	
↪ Quality Notification	C	
↪ Shipping	C	
↪ Warehouse	C	
Source Administration	C	
↪ Outline Purchase Agreements	C	Redundant objects are present.
↪ RFQ/Quotation	C	

8.3.1 Mixed process

The first observation is that the errors are not always simple mistakes like an *XOR* connector that should be *AND* connector. Surprisingly, some models have logical errors that transcend the level of a single connector and indicate that the process is fundamentally flawed. When making a process model, it is important to be absolutely clear what the process is about once it is instantiated (i.e., what is the definition of the "case" being handled in the EPC). It is important not to mix different processes into one diagram. A nice illustration of this problem is shown in Figure 24. This EPC is taken from the "Recruitment" module. The left-hand side of the process is about a vacant position that is created by event "Need to recruit has arisen". The right-hand

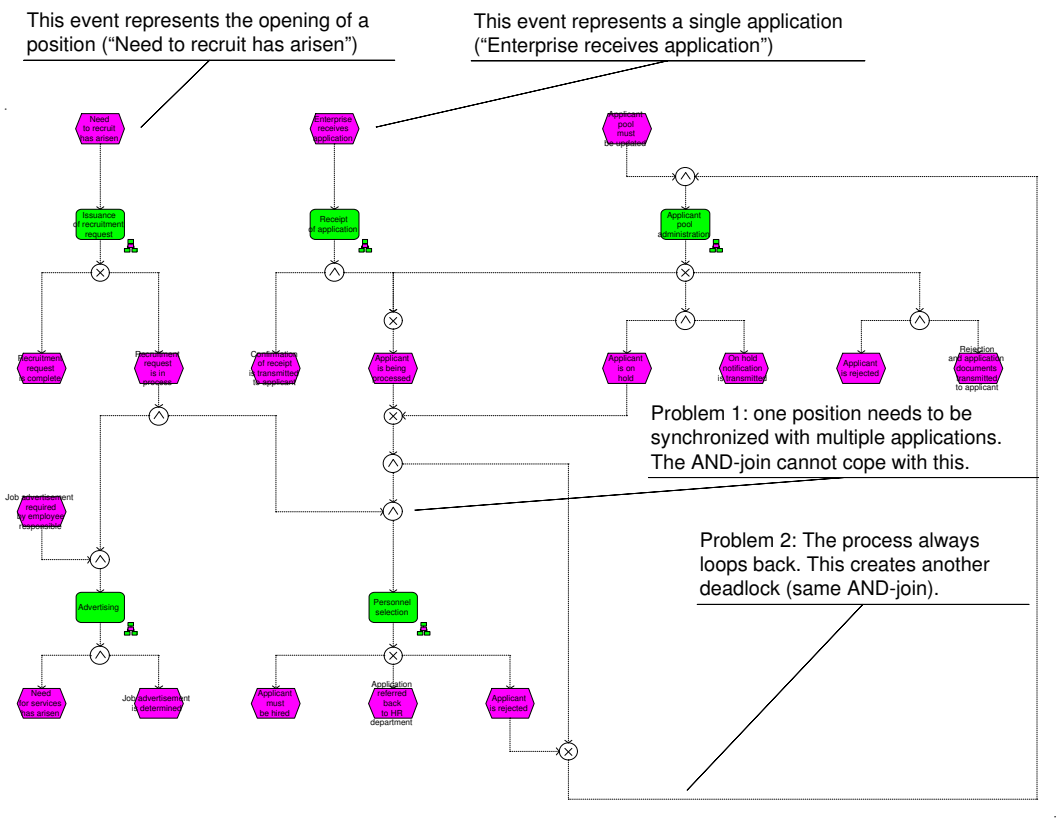

Fig. 24. The “Recruitment” EPC with fundamental flaws.

side however, is about individual applications. This part is triggered by the event “Enterprise receives application”. Note that there may be many applications for a single position. This makes it unclear whether the process is about filling the vacant position or about dealing with the applications. This leads to all kinds of problems. Figure 24 highlights two of these problems. The first problem is that most applications (all except the first one) deadlock because of the *AND*-join connector that requires a position for each application. The second problem is of a similar nature. Looping back will create a deadlock at the same location. The only way to resolve this problem is to split the EPC in two separate EPCs: one for the vacant position and one for dealing with applications.

8.3.2 Inconsistent models

The second observation is that often one particular initial event is applied in several (sub)models. Take, for example, the event “Deliveries need to be planned”. This event occurs in 15 different EPCs! Every time it occurs, it is joined with the event “Delivery is relevant for shipment”. However, in some models this is done via an *XOR*-join, and in some models via an *AND*-join. In Figure 25, we show these two events, used in the “Consignment Processing” module, where they are joined by an *XOR*-join. However, in Figure 26, we show

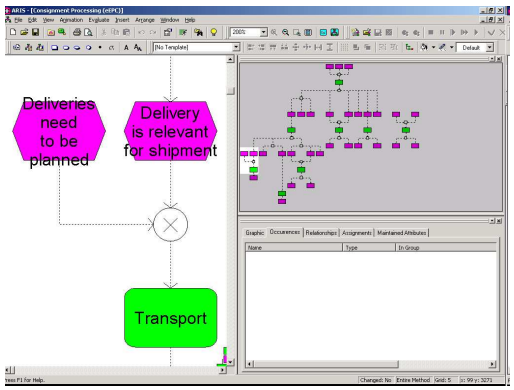

Fig. 25. Fragment of an EPC in the "Consignment Processing" module.

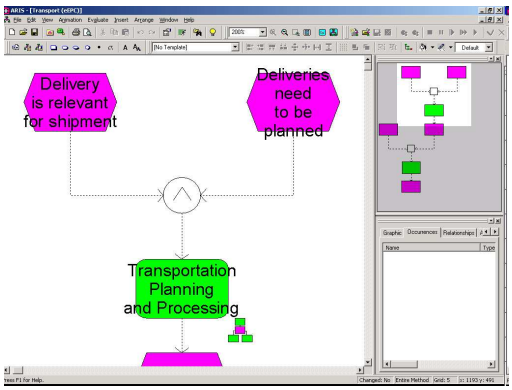

Fig. 26. Fragment of an EPC in the "Transport" module.

the same two events in an *AND*-join configuration. Since these two events are always followed by something that refers to transportation, it seems that they should always appear in an *AND*-join configuration. However, only a designer with deep knowledge of the process that is modeled can decide if that is the case.

8.3.3 Re-use

The third observation, that shows a common problem, is the effect of re-use. Typically, many different organizations have very similar processes. Therefore, when building reference models, it is a good idea to use one model to create another one. The new model is then changed in such a way that it fits the needs of the new organization better. Figure 27 shows a screenshot of the ARIS toolset, showing two models, namely "Q-notification with Complaint Against Vendor" on top and "Internal Quality Notification" below. These two models are exactly alike, except that in the top-model, a vendor's complaint score can be updated. Here, one of the models has been correctly re-used to create the other. In Figure 28, two models are shown for which the re-use was performed incorrectly. The model on the left hand side represents the handling of a "Service Order" and on the right hand side it represents the handling of a "Maintenance Order". They are very similar, except that the latter does not make a distinction between maintenance at a customer site and at an internal site. Both models however, contain the same mistake: If services are to be entered, the rightmost event called "Services are to be Entered" occurs. However, when that is the case, due to the *XOR*-split in front of it, the function "Overall Completion Confirmation" will never be able to execute. Solving this problem requires a good understanding of the modelled situation since many correct solutions are possible. Since both models have the same problem, the process designer should have detected this while he derived the second model from the first, thus giving him the opportunity to correct both mistakes.

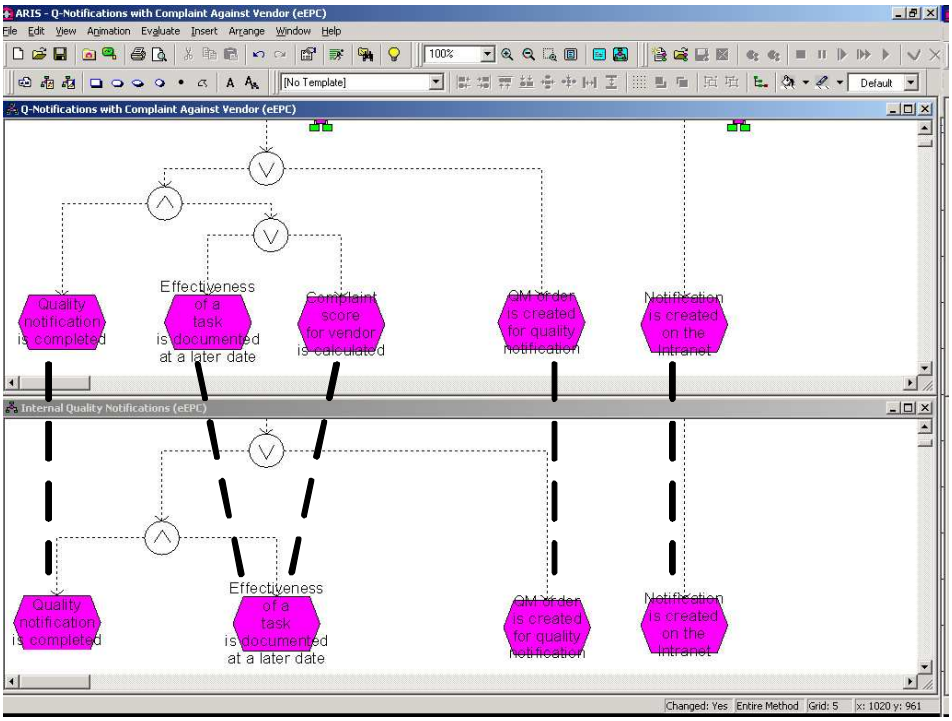

Fig. 27. Re-use of a correct model.

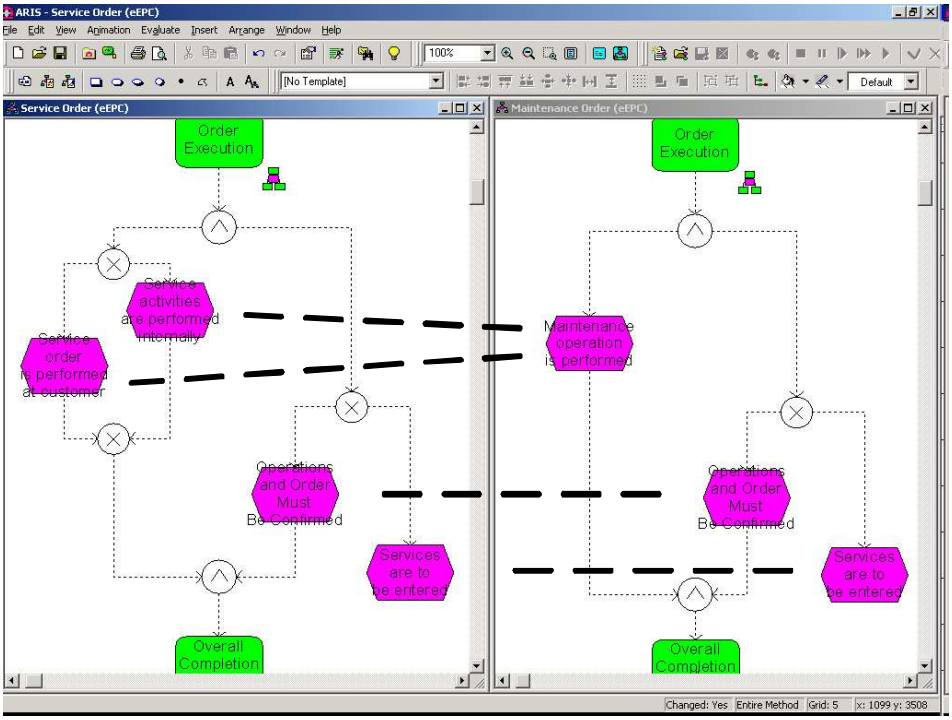

Fig. 28. Re-use of an incorrect model.

8.3.4 Conclusion

The three observations discussed above show that based on a detailed analysis of the EPC reference models we discovered some general problems that should be addressed urgently. It is difficult to take the reference models seriously if they are not 100 percent correct. Note that the reference models are supposed to represent "best practices". Currently, this is clearly not the case. Note that the ARIS toolset offers the so-called "ARIS Semantic Check". This involves the checking of rules such as: [4]

- *Path begins with a start event.* This rule checks whether all start objects are of the Event type.
- *Path ends with an event or function.* This rule checks whether all end objects are of the Event or Function type.
- *Function or rule after a joining rule.* This rule checks whether the successor of a joining rule is of the Event or Function type.
- *Event after splitting rule.* This rule checks whether the successors of a splitting rule are of the Event type.
- *No OR or XOR after a single event.* This rule checks whether there is no opening OR or XOR rule (distributor) after events.

Unfortunately, these checks are purely syntactic and will not identify any of the errors mentioned. Therefore, ARIS clearly offers too little verification support and the support that is offered is definitely not aiming at the semantic level.

To conclude this section we would like to mention that we could analyze all EPCs using the approach presented in Section 5 without the use of transition invariants, i.e., the state spaces of the models where rather small.

9 Concluding remarks

In this paper, we presented a new approach to verify EPC models using reduction techniques, state space analysis, and transition invariants. The reduction rules do not depend on specific interpretations of the various modeling elements (e.g., the OR-join connector). Therefore, the reduced EPC can be used for verification purposes without loosing any information relevant for the correctness of the model. Many models can be reduced to a trivial EPC, i.e., no state-space analysis and transition invariants are needed to assess their correctness. In case the model cannot be reduced, we can either do a state-space analysis or use transition invariants. The state-space analysis provides detailed diagnostics, but for large and complex models this may be challenging from a

[4] Note that this text is taking directly from the "Semantic Check" report generated by ARIS.

computational point of view. Using transition invariants it is possible to detect errors but, in theory, some errors may remain undetected.

The verification approach has been implemented as a plug-in in the ProM framework and is freely available. Moreover, ProM can also exchange process definitions with the ARIS toolset and also allows for a variety of model transformations (e.g., from EPCs to Petri nets to YAWL models).

We have applied the verification approach to real-life process models, e.g., the trading processes in a large Dutch bank. In this paper, we reported on the verification of the SAP R/3 reference models. We have used the ARIS for MySAP reference database, where hundreds of EPCs are defined that cover different areas (from "asset accounting" and "recruitement" to "procurement" and "treasury"). Although the "semantic check" of ARIS is unable to find errors, we have been able to locate several errors. Some errors seem to be simple mistakes like an XOR-join that should be an AND-join. However, other errors are more serious and reveal that some models are fundamentally flawed and that the consistency among models leaves much to be desired. These examples show that the quality of the SAP reference models should be improved. Moreover, our detailed analysis of the SAP reference models shows the applicability of our approach and the ProM framework.

References

[1] G. Keller and T. Teufel. SAP R/3 Process Oriented Implementation. Addison-Wesley, Reading MA, 1998.

[2] W.M.P. van der Aalst and K.M. van Hee. Workflow Management: Models, Methods, and Systems. MIT press, Cambridge, MA, 2002.

[3] F. Leymann and D. Roller. Production Workflow: Concepts and Techniques. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.

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

[5] A.W. Scheer. Business Process Engineering, Reference Models for Industrial Enterprises. Springer-Verlag, Berlin, 1994.

[6] W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, Business Process Management: Models, Techniques, and Empirical Studies, volume 1806 of Lecture Notes in Computer Science, pages 161–183. Springer-Verlag, Berlin, 2000.

[7] J. Dehnert and P. Rittgen. Relaxed Soundness of Business Processes. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), volume 2068 of Lecture Notes in Computer Science, pages 157–170. Springer-Verlag, Berlin, 2001.

[8] J. Dehnert and W.M.P. van der Aalst. Bridging the Gap Between Business Models and Workflow Specifications. International Journal of Cooperative Information Systems, 13(3):289–332, 2004.

[9] J. Desel and J. Esparza. Free Choice Petri Nets, volume 40 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1995.

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

[11] M. Silva, E. Teruel, and J.M. Colom. Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems. In W. Reisig and G. Rozenberg, editors, Lectures on Petri Nets I: Basic Models, volume 1491 of Lecture Notes in Computer Science, pages 309–373. Springer-Verlag, Berlin, 1998.

[12] W. Sadiq and M.E. Orlowska. Modeling and verification of workflow graphs. Technical Report No. 386, Department of Computer Science, The University of Queensland, Australia, 1996.

[13] W. Reisig and G. Rozenberg, editors. Lectures on Petri Nets I: Basic Models, volume 1491 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1998.

[14] W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of Workflow Task Structures: A Petri-net-based Approach. Information Systems, 25(1):43–69, 2000.

[15] K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, Application and Theory of Petri Nets 2003, volume 2679 of Lecture Notes in Computer Science, pages 335–354. Springer-Verlag, Berlin, 2003.

[16] 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.

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

[18] 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.

[19] 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 1998, volume 1420 of Lecture Notes in Computer Science, pages 286–305. Springer-Verlag, Berlin, 1998.

[20] W. Sadiq and M.E. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In M. Jarke and A. Oberweis, editors, Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE '99), volume 1626 of Lecture Notes in Computer Science, pages 195–209. Springer-Verlag, Berlin, 1999.

[21] W. Sadiq and M.E. Orlowska. Analyzing Process Models using Graph Reduction Techniques. Information Systems, 25(2):117–134, 2000.

[22] W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An Alternative Way to Analyze Workflow Graphs. In A. Banks-Pidduck, J. Mylopoulos, C.C. Woo, and M.T. Ozsu, editors, Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02), volume 2348 of Lecture Notes in Computer Science, pages 535–552. Springer-Verlag, Berlin, 2002.

[23] H. Lin, Z. Zhao, H. Li, and Z. Chen. A Novel Graph Reduction Algorithm to Identify Structural Conflicts. In Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-35). IEEE Computer Society Press, 2002.

[24] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In J. Desel, B. Pernici, and M. Weske, editors, International Conference on Business Process Management (BPM 2004), volume 3080 of Lecture Notes in Computer Science, pages 82–97. Springer-Verlag, Berlin, 2004.

[25] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. Data and Knowledge Engineering, 56(1):23–40, 2006.

[26] N. Cuntz, J. Freiheit, and E. Kindler. On the Semantics of EPCs: Faster Calculation for EPCs with Small State Spaces. In M. Nüttgens and F.J. Rump, editors, Proceedings of Fourth Workshop on Event-Driven Process Chains (WI-EPK 2005), pages 7–23, Hamburg, Germany, December 2005. Gesellschaft für Informatik, Bonn.

[27] M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In G. Ciardo and P. Darondeau, editors, Applications and Theory of Petri Nets 2005, volume 3536 of Lecture Notes in Computer Science, pages 423–443. Springer-Verlag, Berlin, 2005.

[28] B.F. van Dongen and M.H. Jansen-Vullers. Verification of SAP reference models. In Business Process Management 2005, volume 3649 of Lecture Notes in Computer Science, pages 464–469. Springer-Verlag, Berlin, 2005.

[29] B.F. van Dongen, H.M.W. Verbeek, and W.M.P. van der Aalst. Verification of EPCs: Using reduction rules and Petri nets. In Conference on Advanced Information Systems Engineering (CAiSE 2005), volume 3520 of Lecture Notes in Computer Science, pages 372–386. Springer-Verlag, Berlin, 2005.

[30] J.M. Colom and M. Silva. Convex geometry and semiflows in P/T nets, A comparative study of algorithms for computation of minimal P-semiflows. In G. Rozenberg, editor, Advances in Petri Nets 1990, volume 483 of Lecture Notes in Computer Science, pages 79–112. Springer-Verlag, Berlin, 1990.

[31] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997, volume 1248 of Lecture Notes in Computer Science, pages 407–426. Springer-Verlag, Berlin, 1997.

[32] B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, Application and Theory of Petri Nets 2005, volume 3536 of Lecture Notes in Computer Science, pages 444–454. Springer-Verlag, Berlin, 2005.

[33] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering, 47(2):237–267, 2003.

[34] B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, International Conference on Conceptual Modeling (ER 2004), volume 3288 of Lecture Notes in Computer Science, pages 362–376. Springer-Verlag, Berlin, 2004.

[35] P. Bernus. Generalised Enterprise Reference Architecture and Methodology, Version 1.6.3. IFIPIFAC Task Force on Architectures for Enterprise Integration, March 1999.

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

[37] P. Fettke and P. Loos. Classification of Reference Models - a methodology and its application. Information Systems and e-Business Management, 1(1):35–53, 2003.

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

[39] M. Rosemann. Application Reference Models and Building Blocks for Management and Control (ERP Systems). In P. Bernus, L. Nemes, and G. Schmidt, editors, Handbook on Enterprise Architecture, pages 596–616. Springer-Verlag, Berlin, 2003.

[40] M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. QUT Technical report, FIT-TR-2003-05, Queensland University of Technology, Brisbane, 2003. (Accepted for publication in *Information Systems*.).

[41] A.W. Scheer. ARIS: Business Process Modelling. Springer-Verlag, Berlin, 2000.

[42] L. Silverston. The Data Model Resource Book, Volume 1, A Library of Universal Data Models for all Enterprises. John Wiley and Sons, New York, revised edition, 2001.

[43] L. Silverston. The Data Model Resource Book, Volume 2, A Library of Data Models for Specific Industries. John Wiley and Sons, New York, revised edition, 2001.