Verification of EPCs: Using Reduction Rules and Petri Nets

B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeek

Department of Technology Management, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{b.f.v.dongen, w.m.p.v.d.aalst, h.m.w.verbeek}@tm.tue.nl

Abstract. Designing business models is a complicated and error prone task. On the one hand, business models need to be intuitive and easy to understand. On the other hand, ambiguities may lead to different interpretations and false consensus. Moreover, to configure process-aware information systems (e.g., a workflow system), the business model needs to be transformed into an executable model. Event-driven Process Chains (EPCs), but also other informal languages, are intended as a language to support the transition from a business model to an executable model. Many researchers have assigned formal semantics to EPCs and are using these semantics for execution and verification. In this paper, we use a different tactic. We propose a two-step approach where first the informal model is reduced and then verified in an interactive manner. This approach acknowledges that some constructs are correct or incorrect no matter what interpretation is used and that the remaining constructs require human judgment to assess correctness. This paper presents a software tool that supports this two-step approach and thus allows for the verification of real-life EPCs as illustrated by two case studies.

1 Introduction

Nowadays, *process-aware information systems* such as Workflow Management (WFM) [4, 15] and Enterprise Resource Planning (ERP) [12] systems are used to support a wide range of operational business processes. These systems are often configured on the basis of a process model and therefore it is of the utmost importance that the process model is *correct*. Therefore, many researchers have worked on the verification of process definitions. Several tools and approaches have been developed for the workflow domain. The basis for most of the work in this area is typically the construction of mathematically sound and executable semantics for a specific modeling language. However, when looking at process modeling techniques, we see that very often, such semantics do not exist, or are too complex for a process designer to comprehend. Still, creating models in these languages is usually easy to do and the resulting models are understood by a broad audience. In this paper, we will focus on one of these modeling languages: *Event-driven Process Chains* (EPCs) [11, 12, 22]. EPCs are used in a large variety of systems, most notably SAP/R3, the Aris Toolset and Aris

O. Pastor and J. Falcão e Cunha (Eds.): CAiSE 2005, LNCS 3520, pp. 372–386, 2005.

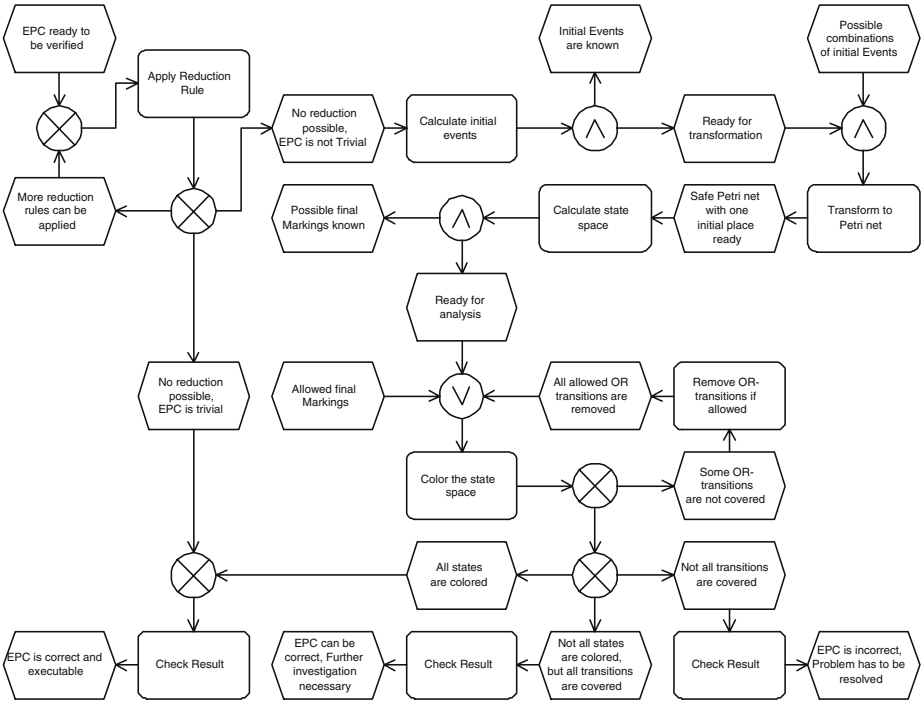

Fig. 1. EPC describing the EPC verification process

Process Performance Monitor (PPM). We will *not* provide "yet another formal semantics" for EPCs to be used as a basis for verification. Instead, we look at verification from a designer point of view. We help the designer to find structural conflicts, and give feedback about possible semantical problems. Furthermore, if there is a trivial executable semantics, we will also provide this information to the designer. However, in case of possible semantical problems, we leave the designer in charge and let him decide what to do.

Since this paper is about the verification of EPCs, we use an EPC to describe our approach. Figure 1 shows the details of the verification process in terms of an EPC. The process consists of two main parts. First we take the EPC that is defined by a process designer and, using simple reduction rules, we extract the possibly problematic area. Then we translate the result into a Petri net and use variants of existing Petri-net-based verification techniques to give feedback to the designer.

As we discussed before, we will look at verification from a designers perspective, instead of from a formal perspective. Therefore, we will look at a more relaxed correctness notion (similar to relaxed soundness [8, 7]). This process consists of multiple steps. First, we introduce fictive nodes to make an EPC with *one* initial event and *one* final event. Then, we use some reduction rules to eliminate the "easy" constructs for which we know that they are correct.

In Section 4 we will discuss these reduction rules. If the EPC under investigation reduces to the trivial EPC, it is correct. In the reduction step, all functions and events except initial and final event are removed. Furthermore, local choices and trivial synchronization constructs are eliminated. If the result of the reduction is not the trivial EPC, the next step is to translate the EPC into a Petri net in a rather simplistic way. In the last step we use the theory of workflow nets [2, 4] and its related concepts such as soundness [2] and relaxed soundness [8, 7]. The last step will provide the designer with one of the following three answers:

The Petri net is *sound*, which means that the original EPC is correct and no further reviewing is necessary.

The Petri net is *relaxed sound*, which means that some constructs need further reviewing of the designer.

The Petri net is *not relaxed sound*, which means that there are unrecoverable problems with the EPC. Corrections are necessary to create a correct EPC.

We have developed a tool for the analysis of EPCs using the approach depicted in Figure 1. The tool is implemented in the context of the ProM framework[1].

In the remainder of this paper we will first look at some related work in Section 2. Then, in Section 3 we introduce concepts like EPCs, Petri nets, soundness, and relaxed soundness. As mentioned before, Section 4 introduces a set of powerful but simple reduction rules for EPCs. In Section 5 we translate the EPC into a Petri net and discuss the verification process in more detail. In Section 6, we briefly describe two case studies: one involving the trade process within a large Dutch bank and the other involving the SAP R/3 reference models. Section 7 concludes the paper.

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

[1] See www.processmining.org for details.

2.1 Verification of Models with Formal Semantics

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, 17, 18]. Since Petri nets have a formal mathematical definition, they lend themselves to great 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 [2, 6, 8, 10, 23] 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 a formal semantics. On the contrary, the most widely used modeling techniques, such as UML and EPCs are merely an informal representation of a process. These modeling techniques therefore require a different approach to verification.

2.2 Verification of Informal Models

Modeling processes in a real-life situation is often done in a less formal language. People tend to understand informal models 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., [1, 3, 7, 14]. 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 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] and followed up by many authors including Van der Aalst et al. in [5] and Lin et al in [16], 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 makes informal models easy to use.

The tendency to capture informal elements by using smarter semantics is reflected by recent papers, cf. [3, 7, 13]. 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 all these authors have different approaches, the goal in every case is similar: *try to give a formal executable semantics for an informal model*.

2.3 Verification 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 if always correct. An example of such a modeling language is IBM MQSeries Workflow [15]. 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.

In this section, we have presented an overview of the literature on process model verification. We have categorized the various methods in three main categories and pointed out why many of them are not used in practice. The main difference between the technique presented in this paper and existing literature is that we will not restrict an informal modeling language to fit our verification, nor will we give an executable specification of an informal model. Instead, we combine the best of existing literature and provide a system 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. The result of our work will be a verification plug-in, implemented in the Process Mining (ProM) Framework, that is able to import EPCs defined in the Aris Toolset[2] and will provide the designer with feedback about possible problems.

3 Preliminaries

In this section, we introduce some basic concepts needed for the verification process. We introduce the modeling language of EPCs and Petri nets. Furthermore, we introduce the notion of soundness and relaxed soundness of Petri nets.

3.1 Event-Driven Process Chains

The concept of Event-driven Process Chains is to provide an intuitive modeling language to model business processes. They were introduced by Keller, Nüttgens and Scheer in 1992 [11]. It is important to realize that the language is not intended to be a *formal* specification of a business process.

[2] See www.ids-scheer.com for information about the ARIS toolset.

An EPC consists of three main elements. Combined, these elements define the flow of a business process as a chain of events. The elements used are:

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 position 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.

Functions, events and connectors can be connected with edges in such a way that (i) events have at most one incoming edge and at most one outgoing edge, but at least one incident edge (i.e. an incoming or an outgoing edge), (ii) functions have precisely one incoming edge and precisely one outgoing edge, (iii) connectors have either one incoming edge and multiple outgoing edges, or multiple incoming edges and one outgoing edge, and (iv) in every path, functions and events alternate (no two functions are connected and no two events are connected, not even when there are connectors in between.)

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 events without incoming edges. After the process is finished, the events that have not been dealt with yet should be events without outgoing edges. 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. In this paper, we use a variant of the classic Petri-net model, namely Place/Transition nets. For an elaborate introduction to Petri nets, the reader is referred to [9, 17, 18]. A Petri net consists of two modeling elements:

Transitions, which typically correspond to either an activity (task, process step) which needs to be executed, or to a "silent" step that takes care of routing.

Places, which are used to define the preconditions and postconditions of transitions. A transition can be *fired* (executed) if the precondition is satisfied. The result of such a firing will be that the postcondition holds.

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

To denote the state a process execution the concept of tokens is used. A token is placed inside a place to show that a certain condition holds. 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 in each of the output places. 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*, or a *marking*.

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 and one sink place and all transitions are on a path from source to sink. Based on WF-nets correctness notions such as soundness [2, 4], generalized soundness [10] and relaxed soundness [8, 7] have been defined.

3.3 State Space

Petri nets can be used as executable specifications of business processes. Whenever a Petri net is given, together with an initial marking it is possible to capture all possible behavior in a *state space*. The only caveat here is that the Petri net should be constructed in such a way that there is a maximum number of tokens that can appear in a place. This property is called *boundedness*, and a special case is when the maximum number of tokens in each place is one. In that case this is called *safeness*.

In this section, we introduced EPCs and Petri nets. 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 EPC. In Section 5, we use Petri nets and state spaces to decide whether the EPC is correct.

4 Reduction Rules

In general, 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 if a split of some type is followed by a join of the same type. 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 complex steps in terms of computation time.

It is easily seen that the applying 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*.

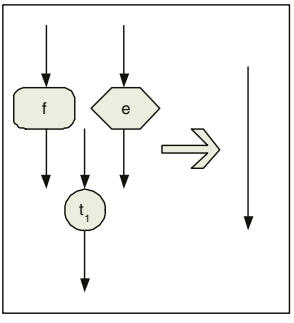

Fig. 2. Trivial construct

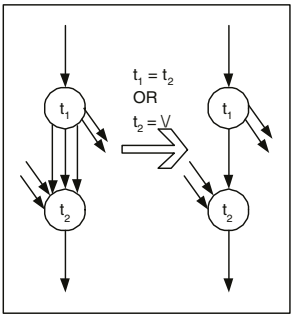

Fig. 3. Simple split/join

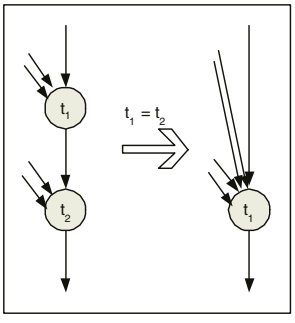

Fig. 4. Similar joins

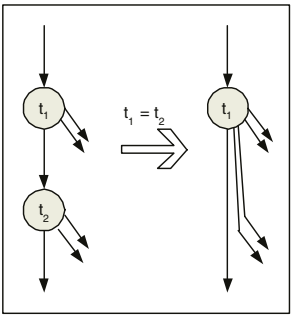

Fig. 5. Similar splits

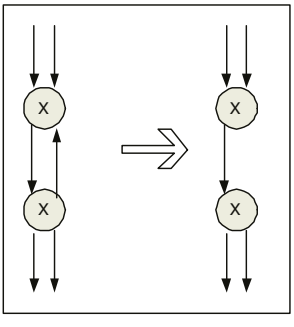

Fig. 6. XOR loop

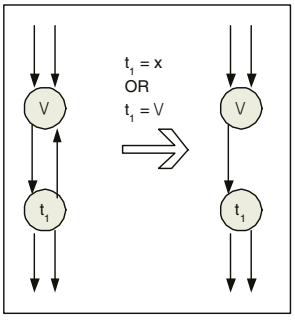

Fig. 7. optional OR loop

Figure 2 shows the reduction rule 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 and 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.

Figure 3 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.

Figures 4 and 5 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 then one edge between the first and the second connector, since connectors are either a split or a join and never both.

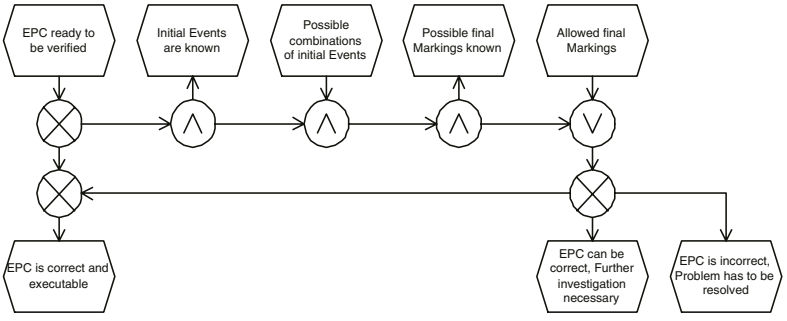

Fig. 8. Reduced EPC for the verification process

Finally, figure 6 and 7 show two very similar reduction rules that deal with loops. In these cases correctness preservation is less straightforward. For Figure 6 it is clear that removing the possibility to loop back is correctness preserving, because the "backward arc" does not introduce any new states. Figure 7 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 time). This implies that if t_1 is an OR-join either the backward arc is taken or any combination of the other arcs.

Figure 8 shows the result of applying reduction rules to the EPC of Figure 1. 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 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 safe Petri net (i.e. a Petri net where a place contains at most one token). 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.

The whole process of verification described in this section is implemented in our the ProM framework. This tool interacts with the Aris toolset, which is widely used in industry for modeling business processes.

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. 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 1, 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 1, we are now ready to transform the EPC into a Petri net.

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 [8, 7]. The translation presented there gives a translation into normal Petri nets, whereas we use the same translation algorithm, but assume the result to be a safe Petri net, or elementary net. In terms of an EPC, this corresponds to ruling out the situation where an event can occur more then once before it is dealt with. Converting an elementary net into a Petri net again is a trivial step, since it only requires the duplication of all places. The choice for elementary nets is motivated by the idea that an EPC should clearly reflect its behavior from its design. When one event is allows to appear again, before it is dealt with by some function, this does not hold any more. The result of the transformation process is shown in Figure 9. Note that in the layout of the Petri net the reduced EPC from Figure 8 is visible.

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

State Space Generation. As we stated in Section 3.3, it is possible to calculate the entire state space for a Petri net, if it is bounded, and the Petri net is not too large. In our case, the Petri net is likely to be of limited size, since we used the reduction rules to get a model that is as small as possible. Furthermore, the Petri net contains at most one token in each place. Therefore we are likely to be able to calculate the state space.

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 in the informal definition of the OR-connector in the process. From this paper it will become clear that you either have both events

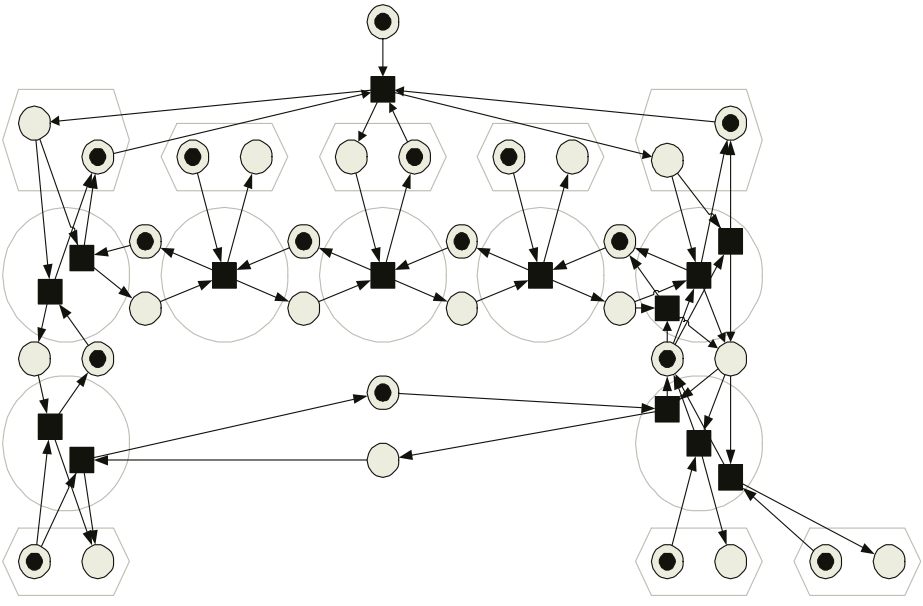

Fig. 9. Petri net translation of the reduced EPC

"Ready for analysis" and "Allowed final markings", or you have "All allowed OR transitions are removed". However, from the description of the EPC, this is not clear. Therefore, we require the user to select those possible outcomes that correspond to correct executions of the process.

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 reduced state space. We call those transitions "not covered".

In principle, transitions that are not covered show that there is possible 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. For example an OR split on two functions A and B can be used to express that you want to execute either A, or A and B, but never just B. In the verification process, this needs to be taken into account. If for example the transition that goes only to B is not covered, then it can safely be removed. However, this can only be done if the transition

to A and B is covered. This check is performed for all transitions that belong to OR connectors and are not covered. Some of them will be 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, when we know all the transitions that are covered, we can provide the final answer.

There are three possible answers, namely:

The EPC is correct. 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. 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 result.

The EPC is incorrect. 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.

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 is not "black or white". Instead, there is a gray area where the EPC can be executed correctly, but can also run into problems. This gray area is not a flaw 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 should be used to talk about the process and not as an executable specification. However, it is possible to derive such an executable specification from the EPC.

6 Two Case Studies

When developing methods for the verification of informal modeling languages, such as EPCs, there is a need to show applicability in real life. Therefore, we tested our approach in two different settings. The first case study was conducted within the Trade Department of a large Dutch bank.[3] There we applied our approach and tool on a trade execution process. We were not primarily interested in the outcome of the algorithm, i.e., whether the EPC analyzed was correct or not, but whether the consultants that modeled the EPC would understand the concepts described in this paper, and whether they would be able to use the tools we developed. The second case study was not conducted within an external organization. Instead we used our tool for the verification of some SAP reference models present in SAP R/3 and Aris for MySAP. Also in the second case study, we found some interesting problems.

[3] We cannot disclose the name of the bank.

6.1 Verification of Trade Execution Process in a Dutch Bank

Within the bank, business consultants made large EPCs modeling the trade execution process. They used the Aris toolset for modeling their business processes. They approached us to verify these processes. We applied the approach and the conclusion of the ProM tool was that the EPC *could* be correct. In other words, there existed possible executions that were not desirable. Using our tool, the consultants were able to identify the problem area's and from that they concluded that the model was correct and that the intended behavior would not lead to undesirable outcomes. Performing this test on their trade execution process made them decide to keep on using the ProM tool in the future. Figure 10 shows the trade execution process in our ProM tool.

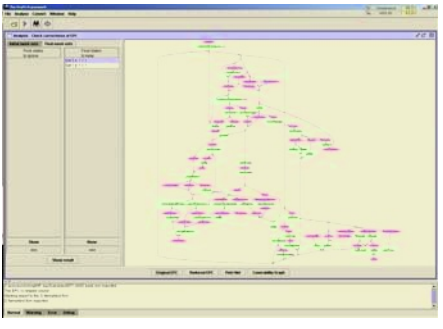

Fig. 10. ProM showing the trade process

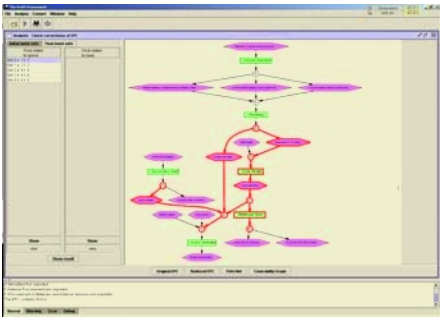

Fig. 11. ProM showing the SAP process

6.2 Verification of SAP Reference Models

The SAP reference models are widely used in industry as a starting point for the configuration of SAP implementations. Of course, one would expect all these reference models to be correct, or at least to be possibly correct. Surprisingly, many reference models contained unrecoverable errors, and they would (if applied directly in industry) definitely lead to undesired behavior of the SAP system. In Figure 11 we show our tool highlighting the problem area of one of the SAP reference models. The model shown here is the "Procurement of Materials and External Services" process, where a mistake was made in one of the connectors, since it was modelled as a XOR-join instead of an AND-join.

The two case studies highlight the applicability of the approach. Unfortunately, we cannot elaborate on them because a detailed discussion would make the paper too long.

7 Conclusion

In this paper, we have presented an algorithm for the verification of EPCs. In contrast to many authors, we do not assume EPCs to be an executable specifica-

tion of a process, nor do we translate the EPC into one. In order to still be able to say something about the correctness of EPCs, we developed an interactive way of verifying EPCs. In this interactive process, we assume the user to have deeper knowledge of the EPC and we assume the user to be able to interpret the results. Besides that, we acknowledge the fact that EPCs are conceptual models and therefore our result cannot be expressed in a binary way. An EPC obviously is incorrect, if some part of it will always lead to undesired behavior, and it is correct if no part will ever lead to undesired behavior. However, there is a gray area in between those two extremes, where the EPC does allow for undesired behavior on the level of the model. This however, does not mean that there is no way of deriving an executable specification using the EPC as a basis.

References

1. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.
2. 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.
3. 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.
4. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
5. 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.
6. 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.
7. 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.
8. 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.
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. 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.

11. 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.
12. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
13. 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.
14. 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.
15. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
16. 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.
17. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
18. 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.
19. 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.
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. A.W. Scheer. *Business Process Engineering, Reference Models for Industrial Enterprises*. Springer-Verlag, Berlin, 1994.
23. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer-Verlag, Berlin, 2000.