Business Process Modelling with Continuous Validation

Stefan Kühne[1], Heiko Kern[1], Volker Gruhn[2], and Ralf Laue[2]

[1] Business Information Systems, University of Leipzig
Johannisgasse 26, 04103 Leipzig, Germany
{kuehne|kern}@informatik.uni-leipzig.de

[2] Applied Telematics/e-Business**, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany
{gruhn|laue}@ebus.informatik.uni-leipzig.de

Abstract. In this paper, we demonstrate the prototype of a modelling tool that applies graph-based rules for identifying problems in business process models. The advantages of our approach are twofold. Firstly, it is not necessary to compute the complete state space of the model in order to find errors. Secondly, our technique can even be applied to incomplete business process models. Thus, the modeller can be supported by direct feedback during the model construction. This feedback does not only report problems, but it also identifies their reasons and makes suggestions for improvements.

1 Introduction

Validation of business process models has been studied for a long time. In a recent paper [1], Wynn et al. write that "process verification has matured to a level where it can be used in practice". Although this is good news, we argue that many of the current approaches do not yet support the business process modeller in an optimal way. The reason for this statement is that most validation methods are applied only after the model has already been completed. For example, all those methods which transform a business process model into an analyzable Petri-net have problems with incomplete models.

In this paper, we present a validation approach that gives the modeller an immediate feedback about modelling errors. A prototypical implementation of our approach has been integrated into a business process model editor. It locates not only "technical" errors (such as deadlocks in the control flow), but also parts of the model that can be regarded as "bad style". The modeller not only receives the information that the model has problems, but our tool also shows the locations of error causes in the visual representation and suggests how to fix the problems. A rule language allows the user to add own rules, for example, rules for checking company-wide style guidelines.

Similar to techniques such as continuous compilation and continuous testing that are integrated into modern software development systems, our approach –

** The Chair of Applied Telematics/e-Business is endowed by Deutsche Telekom AG.

which we call *continuous validation* – can help to detect and fix errors at a very early stage.

2 Basic Concepts and Definitions

2.1 Event-Driven Process Chains

There exist several languages for graphical business process modelling. In this paper, we use Event-Driven Process Chains (EPC) [2] to demonstrate our approach. However, the underlying principles can be applied to other languages such as BPMN [3] as well.

We would like to start with a semi-formal description based on the metamodel given in Fig. 1. EPC models are finite directed coherent graphs consisting of non-empty sets of nodes and arcs. Nodes are either functions (activities which need to be executed, depicted as rounded boxes), or events (representing pre- and postconditions of functions, depicted as hexagons) or connectors. Arcs between these elements represent the control flow. A function has exactly one incoming and exactly one outgoing arc. An event has at most one incoming and at most one outgoing arc. An event without incoming arcs is called *start event*, and an event without outgoing arcs is called *end event*.

The connectors are used to model parallel and alternative executions. There are two kinds of connectors – splits and joins. Splits have one incoming and at least two outgoing arcs, joins have at least two incoming arcs and one outgoing arc.

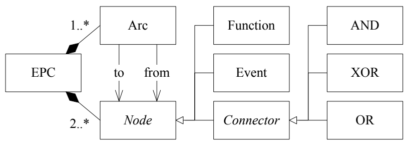


Fig. 1. EPC metamodel

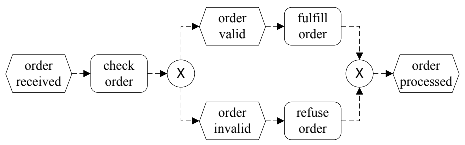


Fig. 2. A simple EPC

AND-connectors (depicted as $\textcircled{\scriptsize\wedge}$) are used to model parallel execution. When an AND-split is executed, the elements on all outgoing arcs have to be executed in parallel. An AND-join connector waits until all parallel control flows on its incoming arcs are finished. XOR-connectors (depicted as $\textcircled{\scriptsize X}$) can be used to model alternative execution: An XOR-split has multiple outgoing arcs, but only one of them will be processed. An XOR-join waits for the completion of the control flow of one of its incoming arcs. If flow arrives from more than one arc most semantic definitions regard it as a synchronisation error. No flow of control is forwarded in this case. OR-connectors (depicted as $\textcircled{\scriptsize\vee}$) are used to model parallel executions of one or more flows. An OR-split starts the processing of one or more of its outgoing arcs. This means, after an OR-split with n outgoing arcs, at least one of those arcs and at most all n arcs become active. An OR-join waits until all control flows that can reach it are finished.

Fig. 2 shows a very simple order process modelled as EPC. Note that it is unnecessary that splits and joins occur pairwise and form a well-structured model. Actually, the notation allows arbitrary combinations of connectors which is often the cause for modelling errors.

A state of an EPC is a binary marking of its elements, i.e. some elements of an EPC are marked as “active” by placing tokens on them. A state is a start state iff start events are active. A sequence of states is an execution of the business process model. A transition relation defines the semantics, i.e. the rules that define under which circumstances a state S_1 in this sequence is allowed to be followed by a subsequent state S_2. Several different definitions exist for transition relations; because of space restrictions we omit a detailed discussion. The interested reader is referred to [4, 5, 6].

2.2 Control Flow Errors

Van der Aalst [2] has defined *soundness* for EPCs which is the most important correctness criterion for business process models. This definition includes three required properties:

1. In every state that is reachable from a start state, there must be the possibility to reach a final state, i.e. a state without a subsequent state according to the transition relation *(option to complete)*.
2. If a state has no subsequent state (according to the transition relation that defines the precise semantics), then only end events must be marked in this state *(proper completion)*.
3. There is no element of the EPC that is never marked in any execution of the EPC *(no needless elements)*.

Violations of the soundness criterion usually indicate an error in the model. A typical example is a deadlock situation with an XOR-split which outgoing arcs are joined later by an AND-join. This example would lead to a violation of the second property: It is possible that no further progress in the execution of the EPC can be made, but the elements at the incoming arcs of the AND-join are still marked because the AND-join has to wait until *all* incoming arcs have been traversed.

3 Existing Verification Methods

The problem to overcome when verifying EPC models is that the modelling language has been introduced *without defining their semantics*. For this reason, the first step in a verification process is usually to transform the model into a

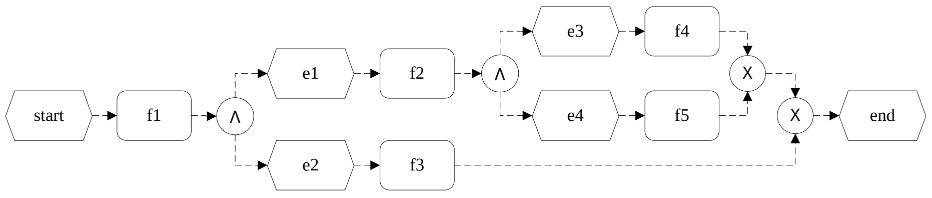


Fig. 3. Nested mismatched AND-splits and XOR-joins

formalism with well-defined semantics. In fact, the rules for such a transformation *define* the semantics of the model. Petri-nets are the natural choice for that purpose. They have been used by several authors [2, 7, 6][3].

Once an EPC model has been transformed into a Petri-net, the whole range of existing tools for analyzing Petri-nets is available. As shown in [9, 8, 5, 6], those tools can be used for checking the correctness of EPCs.

The Petri-net based approach works very well in practice, but has two disadvantages. First, often the analysis result covers only the information whether the model contains errors, without giving feedback about the reason for an error. Even if the verification tool "translates" the counterexample back to the EPC model, it is likely that information will be lost. For example, Fig. 3 obviously contains two synchronization problems – one in the outer AND-XOR control block and another one in the inner AND-XOR control block. However, the synchronization error in the inner control block makes the execution always blocks, and there will never be an execution where both incoming arcs of the rightmost XOR-join are enabled. For this reason, a dynamic analysis tool that explores the state space will be unable to report the problem of the outer AND-XOR control block. The second disadvantage is that it is often impossible to locate errors in models that are not yet completed (e.g., EPCs containing several subgraphs which are not yet connected with each other).

Another well-studied method for the validation of EPCs and similar models is the application of reduction rules (see e.g. [10]). The idea of the reduction approach is to delete repeatedly sections from an EPC which are well-structured (for example a control flow block where an AND-split is matched by an AND-join) and are thus trivially correct. If an EPC can be reduced to a single node in this way, it is correct. Otherwise, no answer about its correctness can be given. That is, the answer to the question "Are there any problems with the model?" is either "No" or "Don't know", which is far from the desired "No" or "Yes, and the problems are as follows...". However, recent work by Mendling [6] has made a fruitful contribution. By considering typical error situations in the reduction rules, Mendlings approach allows us to answer our question about errors in the model with "No", "Yes, and the problems are..." or "Don't know". In the latter case, Petri-net-based methods can still be used to come at least to a "Yes or No"

[3] The given references are not exhaustive. A more detailed categorization of related work can be found in [8] and [6].

answer. We considered the errors that can be found by reduction rules in [6] as a starting point for our own rules to locate errors (described in Sect. 5).

Our work has also been influenced by the approach described in [11]. This approach adapts ideas from Petri-net theory (like the concept of handles [12, 13]) to EPCs. In [11], the authors locate causal relationships between parts of an EPC. These relationships (called *causal footprints* in [11]) are relations like “after element X has been processed, at least one of the elements in the set $\{Y, Z\}$ has to be processed”. Using this method, error patterns can be detected and the reasons for errors can be identified. Moreover, the method works on incomplete models as well. Unfortunately, in the form as described in [11], the approach has not yet matured for practical use. A minor reason is that it does not work with EPCs containing OR-connectors (but it would not be too much effort to expand the method such that OR connectors could be considered). But more important is that the computation of relationships among elements is far too slow because too many relations have to be considered.

Another promising heuristic approach has been described in [14]. It uses decomposition of workflow graphs into single-entry/single-exit fragments and can quickly classify some fragments as sound or unsound. A validation in [14] shows that the approach does well for the majority of a sample of industrial business processes.

4 Immediate Validation Feedback in Business Process Modelling

4.1 Validation Approach

From our point of view, a sophisticated validation support of business processes gives immediate and continuous feedback to business analysts about weaknesses and inconsistencies in possibly incomplete models. The established modelling process with sequential modelling, validation and evolution stages should be shortened as far as possible to a modelling process with integrated validation support. A analogy is a programming environment with continuous syntax checking which is in our case extended to semantic and even pragmatic issues.

Our intended validation support is based on three principles. (1) For adaptability and extensibility reasons the validation rules should be expressed in a modular and human-readable manner. For this purpose we propose declarative validation rules which enable the expression of additional error patterns and modelling idioms by adding new rules without effecting existing ones. (2) To avoid disadvantages of non-localised error messages, the validation strategy should work on input models in a native way. This can be realised by referring to model elements in validation rules or by delegating calculations of model properties to helper functions that navigate on models. (3) A rather soft requirement is that the validation solution is seamlessly integrated into the modelling tool of choice with the ability to annotate error causes and to suggest possible improvements.

In the area of business process modelling the instantiation of these principles results in process-specific validation rules and helper functions. Since EPC models as well as other business process modelling languages represent graphs, the required helper functions provide graph calculation functionalities such as “there is a path between two elements” or “every path between two elements goes through another element”. Furthermore, basic element-related operations, such as “this element is a start event” or “replace an element by another element” and set-related operations, such as “all start events of an EPC” or “intersection of all successors of an element and all predecessors of another element” are typically referenced in validation rules.

4.2 Implementation Strategy

Our approach is designed as an extension of the community-driven open-source EPC modelling tool *bflow*[4]. It, therefore, relies technically on the Eclipse Modeling Framework (EMF)[5] and the Graphical Modeling Framework (GMF)[6]. The implementation of our intended validation support requires a declarative validation language. Moreover, a model-to-model transformation language is required to express queries on models and possible error-resolving solutions. In the technical space EMF, openArchitectureWare (oAW)[7] and Epsilon[8] represent model management frameworks that provide these functionalities in an integrated manner. We chose oAW as implementation technology because of its build-in GMF integration.

Validation rules in our approach are expressed in the oAW Check language (see e.g. Listing 1). A declarative rule is introduced by a context specifying a metamodel element which instances are validated. The set of model elements can optionally be restricted by an if-clause. The ERROR keyword signals that a violated validation rule represents an error and specifies a corresponding advice. WARNING and INFO provide alternative error categories. The Boolean expression after the colon provides a validation assertion which holds for valid models.

Listing 1. A check rule

```
// Connectors are either splits or joins
context epc::Connector
    if this.isJoin()
    ERROR "Connector is a split and \
    a join as well. ..." :
    !this.isSplit();
```

Listing 2. An XTend function

```
// Is a connector a join-connector
Boolean isJoin(Element e) :
    epc::Connector.isInstance(e)
    && e.incomingArcs().size > 1;
```

[4] http://www.bflow.org/
[5] http://www.eclipse.org/modeling/emf/
[6] http://www.eclipse.org/modeling/gmf/
[7] http://www.eclipse.org/gmt/oaw/
[8] http://www.eclipse.org/gmt/epsilon/

The example given in Listing 1 assures that AND-, OR- or XOR-joins are not splits. The restriction expression as well as the assertion expression refer to separately defined oAW XTend functions. XTend is a functional programming language designed for model-to-model transformations. In our case XTend is used to express queries on the input model and to modify models. The definition of the *isJoin()* function used in Listing 1 is shown in Listing 2. Re-use is supported by the function concatenation operator "." and polymorphism.

While Check specifications work on EMF models, the oAW GMF adapter enables the execution of Check rules from a GMF editor. For this purpose the EPC-specific GMF plug-in representing the modelling editor has to be extended accordingly. Violated rules create corresponding markers that are assigned to the visual representation of model elements. In this way the modeller gets localised feedback. Furthermore, error messages may provide hints and improvement suggestions.

The build-in oAW GMF adapter supports the validation in read-only transactions. To enable model modifications in error cases with obvious improvements we changed this behaviour to read-write-transactions. A model modification that should be triggered by a validation rule can be expressed by a restriction expression matching the error pattern with a falsified assertion expression triggering the model modification (see Listing 3). XTend enables native Java calls which provides a technical interface for implementing user interactions in the case of non-obvious or alternative change suggestions.

Listing 3. A validation rule which triggers model modifications

```
// All elements are named
context epc::Node
    if (this.name == null)
    WARNING "Name was set to a default value" :
    (this.setDefaultName() -> false);
```

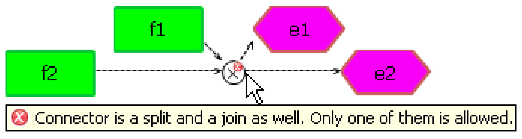
The mentioned technologies are independent of the EPC modelling language represented by the bflow tool and may be used to validate other model types. The contribution for the validation of BPM models results from a set of Check rules validating common classes of errors (discussed in Sect. 5). These rules rely on a limited set of performance optimised XTend functions.

The underlying principles of our approach are not restricted to the mentioned technologies. They can be applied to other EPC modelling tools, e.g. ARIS Design Platform with its macro language by IDS Scheer[9].

5 Examples

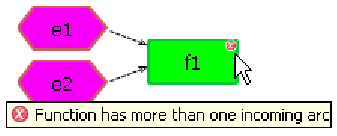
In this chapter, we discuss typical classes of modelling errors that are identified by checking rules included in our tool.

[9] http://www.ids-scheer.com



Connector is a split and a join as well. Only one of them is allowed.

(a)



Function has more than one incoming arc

(b)

Fig. 4. Connector with several incoming and outgoing arcs (a), function with several incoming arcs (b)

5.1 Syntax Errors

A less complicated task is to check an EPC (or to be precise: a construction that is supposed to be one) for syntactical correctness. Not all syntactical requirements are included into the EMF meta-model. Some constraints like non-existence of cycles made from connectors only require a few separate rules [15]. Writing syntax checking rules is in no way difficult [16] and is supported in our approach.

Fig. 4(a) shows a syntax error which violates the Check rule given in Listing 1. Another common syntax error is that an event or a function has more than one incoming or outgoing arcs (see e.g. Fig. 4(b)). Such errors are not uncommon: We found 14 of them in the 604 models of the SAP reference model.

5.2 Connector Mismatch

Informally spoken, deadlocks and synchronisation errors in an EPC result from a mismatch between the type of a split and the type of the corresponding join. If an XOR-split starts only one of two possible flows of control which are joined later by an AND-join waiting for both flows being completed, this will always result in a deadlock.

In an EPC that is not well-structured, the term of a "corresponding" join needs to be defined first. We do this as follows:

Definition 1 *A split s is matched by a join j (symbol: $match(s, j)$) iff there exist two directed paths from s to j which only common elements are s and j.*

If there are no "entries into" or "exits from" the control structure that starts with an XOR-split s and ends with and AND-join j, the process will always have an error regardless of the elements "between" s and j. The terms entries and exits are defined as follows:

Definition 2 *Let s be a split and j be a join with $match(s, j)$. We say that there are no entries and no exits between s and j, $seseMatch(s, j)$, if the following conditions hold:*

1. *Every path from s to an end event must contain j.*
2. *Every path from a start event to j must contain s.*
3. *Every path from s to s must contain j.*
4. *Every path from j to j must contain s.*

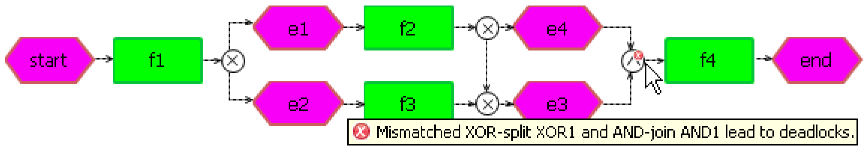
The above definitions allow us to find a certain type of control flow errors where the type of the split differs from the type of the join. From the 178 errors found in [6], 44 fell into this category.

We have similar rules for finding errors that occur in control flow blocks with entries and exits and for finding errors that occur in iterations (circles) in an EPC. Due to space restrictions, we do not describe them in detail.

Fig. 5 shows an obvious error consisting of a mismatched XOR-split (the left one)/AND-join combination. The two XOR-connectors inside the fragment illustrate that the rule given in Listing 4 also finds errors in non well-structured models.

Listing 4. Mismatched XOR-split and AND-join

```
// XOR−AND−Mismatch
context iepc::Connector if (this.isAndJoin())
    ERROR "Mismatched XOR−split ..."
    this.allPredessesors().notExists(p| p.isXorSplit() && p.seseMatch(this));
```



Mismatched XOR-split XOR1 and AND-join AND1 lead to deadlocks.

Fig. 5. Mismatched XOR-split and AND-join

For an XOR-split/OR-join combination, the situation is different: The OR-join waits for all incoming flows to be completed. If there is only one such flow (as the result of the choice at the XOR-split), the execution will continue without problems. For this reason, most verification approaches (a remarkable exception is [5]) do not complain about such combinations. Our rules produce a warning, because it is a good advice to the modeller to change the OR-join into an XOR-join in order to avoid misunderstandings.

For EPCs that occur in practice, the checks discussed in this section can be performed fast; running them repeatedly as a background task is not a problem.

5.3 Synchronisation Problem in AND-join

The most common type of error found in [6] (102 errors out of 178 found) falls into a category that is depicted in Fig. 6: If the upper outgoing path at the XOR-split is processed, a deadlock will be produced at the AND-join.[10] Fig. 6 shows how our tool signals the problem. Listing 5 shows the corresponding Check rule for detecting such problems.

[10] In practice, we would not regard all such models as erroneous, but this discussion is outside the scope of this paper.

Listing 5. Check rule for (X)OR-split caused synchronisation problems in AND-joins

```
context epc::Connector if this.isAndJoin() // An AND−Join J
  ERROR ”AND−split might not get control” :
  // has no S with S is connected to J and
  this.allPredessesors().notExists(S|
    // S is an XOR−split or an OR−split
    (S.isXorSplit() || S.isOrSplit())
    // and S has a successor SSucc not conntected to J
    && S.successors().exists(SSucc|
      SSucc != this && SSucc.isNotConnectedTo(this))
    // and J has a predecessor JPred not connected from S
    && this.precessors().exists(JPred|
      JPred != S && S.isNotConnectedTo(JPred)) );
```

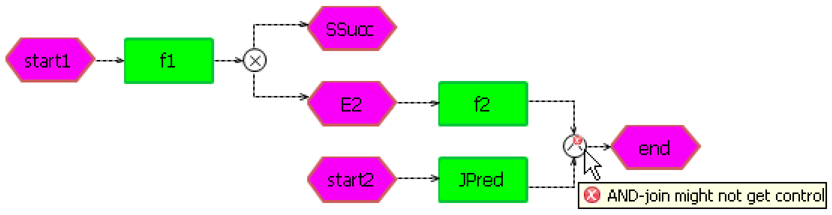


Fig. 6. Synchronisation problem in an AND-join

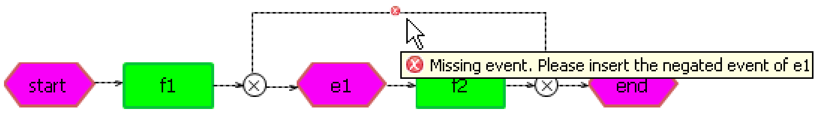
5.4 Company-Wide Style Rules

The rule set in our tool is open for adding new rules. If, for instance, a company wants to transform the EPC model later directly into executable BPEL code which enforces well-structuredness, it could be a requirement to use certain company-wide style rules for EPCs in order to disallow unstructured EPC models.

Such domain-specific adaptations can simply be realised in our tool by extending the predefined rule set. An example for such a rule is given in Fig. 7. The intention of the fragment between the two XOR-connectors is that function f2 is executed only if event e1 occurs which corresponds to a single if-expression in a programming language. However, because of a missing condition the arc connecting the two XOR-connectors might also be regarded as path that can always be executed. To avoid ambiguities a distinguishing event is desirable. With a rather simple check rule this pattern can found and the suggestion to insert a negation of event e1 can be created.

6 Validation

We used the appendix of Jan Mendlings PhD thesis [6] for a first validation of our checking rules. It contains the results from validating 604 EPCs from the



Missing event. Please insert the negated event of e1

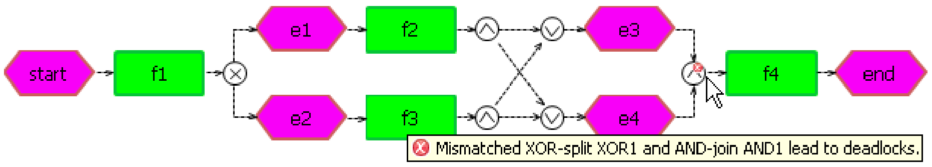
Fig. 7. Missing event between an XOR-split and XOR-join with a suggestion for improvement

SAP reference model. By using reduction rules, 178 error patterns have been found in 90 models. Our tool detected 176 of these errors (and several other errors that remained undetected by reduction rules). The remaining 2 errors are loops that start with an OR-join instead of an XOR-join. We regard the fact that this results in a deadlock with the semantics from [6] rather as a shortcoming of the semantics definition in [6].

For 57 models, the reduction rules in [6] did not reveal an error. Mendling used a state space exploring algorithm for judging about the soundness of these 57 models. 36 have been proved to be unsound, and our tool located one or more errors in *all* of these models. The remaining 21 models have shown to be sound, and for those models our tool produced only one alert.

7 Conclusions and Directions for Future Work

With the error patterns discussed in Sect. 5, we can already identify the vast majority of control flow problems in an EPC. However, the presented validation approach basically has a heuristic nature. The checks in the editor are not meant to be a complete validation. The EPC depicted in Fig. 8 shows two weaknesses of our approach. Although the model represents a sound EPC, the rules discussed so far produce a false error message for the AND-join. Furthermore our rules do not create hints according to the OR-joins which might be replaced by two XOR-joins. We deliberately did not try to include rules for such exotic cases.



Mismatched XOR-split XOR1 and AND-join AND1 lead to deadlocks.

Fig. 8. Sound EPC for which our tool would still report a problem

An evaluation based on the set of EPCs given in the SAP reference model and other EPC repositories shows that a limited set of rules already detects the vast majority of common errors. We see the main advantage of our approach in the fact that information about possible problems in a model is immediately reported to the modeller, even before the model has been completed. These alerts provide suggestions for improvement and are given in a way that does not force the modeller's attention away from the modelling task.

It will be a direction of future research to deal with patterns where a model change could result in more *readable* models – even for EPCs where the original model did not have deadlocks and similar control-flow problems [17].

References

1. Wynn, M.T., Verbeek, H., van der Aalst, W.M., Edmond, D.: Business Process Verification - Finally a Reality! Business Process Management Journal **to appear**
2. van der Aalst, W.M.: Formalization and verification of event-driven process chains. Information & Software Technology **41**(10) (1999) 639–650
3. Business Process Management Initiative: Business Process Modeling Notation. Technical report, BPMI.org (2004)
4. Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Business Process Management. (2004) 82–97
5. Wynn, M.T.: Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins. PhD thesis, Queensland University of Technology, Brisbane (2006)
6. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (2007)
7. van Dongen, B.F., van der Aalst, W.M., Verbeek, H.M.W.: Verification of EPCs: Using Reduction Rules and Petri Nets. In: CAiSE. (2005) 372–386
8. van Dongen, B.F., Jansen-Vullers, M., Verbeek, H.M.W., van der Aalst, W.M.: Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. Comput. Ind. **58**(6) (2007) 578–601
9. Langner, P., Schneider, C., Wehler, J.: Relating Event-driven Process Chains to Boolean Petri Nets. Report (9707) (December 1997)
10. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Information Systems **25(2)** (June 2000) 117–134
11. van Dongen, B., Mendling, J., van der Aalst, W.: Structural Patterns for Soundness of Business Process Models. EDOC (2006) 116–128
12. Esparza, J., Silva, M.: Circuits, handles, bridges and nets. In: Applications and Theory of Petri Nets. (1989) 210–242
13. van der Aalst, W.M.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
14. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. Service-Oriented Computing – ICSOC 2007 (Jan 2007) 43–55
15. Nüttgens, M., Rump, F.J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen. (2002) 64–77
16. Gruhn, V., Laue, R.: Checking Properties of Business Process Models with Logic Programming. In: MSVVEIS 2007, INSTICC Press (2007) 84–93
17. Gruhn, V., Laue, R.: Good and Bad Excuses for Unstructured Business Process Models. In: Proceedings of 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007). (2007)