

# Good and Bad Excuses for Unstructured Business Process Models

Volker Gruhn, Ralf Laue

University of Leipzig, Germany, Chair of Applied Telematics / e-Business\*  
{gruhn, laue}@ebus.informatik.uni-leipzig.de

## Abstract

*Building well-structured business process models (similar to writing well-structured control-flow blocks in a programming language) is often recommended in order to raise the quality and readability of such models. This article discusses patterns that are related to structured and unstructured business process models. One kind of these patterns deals with situations where some part of the model is modelled in an unstructured way. For these patterns, possible structured alternatives for improving the model are given. Another kind of patterns deals with situations where an unstructured model is the best way to express some situation.*

*We illustrate this by discussing one pattern of both types. Finally, we give an example of a pattern for which the decision whether an unstructured model should be changed into a well-structured one needs a closer look at the purpose of the model.*

## 1 Introduction

Business process analysts use graphical languages like event driven process chains (EPC), UML activity diagrams, BPMN or YAWL for drawing business process models. Unlike modern programming languages, these graphical languages do not force the modeller to build structured models. Instead, arbitrary jumps from one point in the model to another one are possible. It is well-known that such jumps (GOTO-statements) in a piece of software make the code more difficult to read, and the same holds for graphical models with GOTO-like jumps. Holl and Valentin[8] write that "the current unstructured style of business process modelling, which we can call spaghetti business process modelling, leads to similar problems as spaghetti coding".

Consequently, in one of our previous research papers, we have suggested complexity metrics that are related to the unstructuredness of business process models[6]. Such complexity metrics give the modeller a feedback about possible improvements in a model. If the unstructuredness metric is very high, the modeller should consider to re-structure the model.

From this approach, two pattern-related questions arise: At first, it is helpful to compile a set of patterns containing well-structured alternatives for situations that are often unnecessarily modelled in an unstructured way. Such a pattern catalogue can help a modeller to improve the model.

---

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

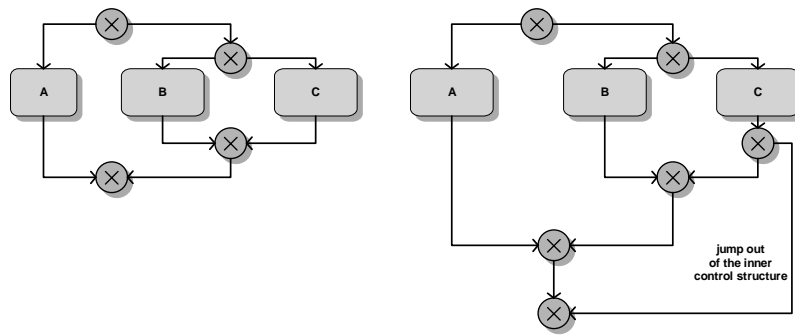


Figure 1. The left model has properly nested control flow blocks, the right one has not.

	sound[24]	weak sound[15]	relaxed sound[5]	lazy sound[17]
no deadlocks or livelocks	✓	✓	must hold for one possible execution only	
no active activities when an end activity is reached	✓	✓	must hold for one possible execution only	✓
no “dead parts” in the model	✓		✓	

Table 1. Soundness Properties

However, in some situations it is actually a good idea to use a modelling construct that is not well-structured. This is similar to the fact that exits from loop-structures in software code can make the understanding of the code easier, even if the code becomes unstructured[18]. We are interested in compiling such patterns as well. A program that computes the metrics for measuring (un-)structuredness of a model should know those rare situations when unstructured modelling should not be regarded as bad modelling. Consequently, this program should not complain about problems that actually are no problems at all.

We have started to collect patterns of both types, and we are currently expanding our pattern catalogue. In Sect. 5, we give a pattern for a situation where unstructured modelling is good modelling. This is followed by another pattern in Sect. 6 as an example for bad use of unstructured modelling. A third pattern in Sect. 7 shows that sometimes the decision whether an unstructured model is a good or a bad model needs a closer look at the purpose of the model.

However, before starting with the patterns, in Sect. 2, we give a brief sketch of the modelling language Event Driven Process Chains that will be used for the patterns in this paper. In section 3, we discuss the terms structuredness and unstructuredness and introduce some desirable properties that a business process model should have.

## 2 Event Driven Process Chains

If we have to depict a business process model in this article, we use the notation of Event Driven Process Chains (EPC)[24]. To a large degree, the patterns are independent from the

chosen modelling language. The issues related to (un-)structuredness apply for the languages UML activity diagrams, BPMN and YAWL as well.

EPCs consist of functions (activities which need to be executed, depicted as rounded boxes), events (pre- and postconditions before / after a function is executed, depicted as hexagons) and connectors (which can split or join the flow of control between the elements). Arcs between these elements represent the control flow. The connectors are used to model parallel and alternative executions. There are two kinds of connectors: Splits have one incoming and at least two outgoing arcs, joins have at least two incoming arcs and one outgoing arc.

AND-connectors (depicted as  $\bigwedge$ ) are used to model parallel execution. When an AND-split is executed, the elements on all outgoing arcs have to be executed in parallel. The corresponding AND-join connector waits until all parallel control flows that have been started are finished.

XOR-connectors (depicted as  $\times$ ) can be used to model alternative execution: A XOR-split has multiple outgoing arcs, but only one of them will be processed. The corresponding XOR-join waits for the completion of the control flow on the selected arc.

Finally, OR-connectors (depicted as  $\bigvee$ ) are used to model parallel execution of one or more flows. An OR-split starts the processing of one or more of its outgoing arcs. The corresponding OR-join waits until all control flows that have been started by the OR-split are finished.

### 3 Model Properties

Informally spoken, a business process model is well-structured if the control flow blocks built by splits and joins are properly nested, i.e. every split is followed by a corresponding join of the same type. A formal Petri-net based definition is given in [23].

As an example, the left model in Fig. 1 shows a well-structured EPC. Its splits and joins are properly nested, i.e. the inner control structure (XOR-split/XOR-join) is contained completely within the outer control structure. In the right model, there is a jump out of the inner control block, and this jumps leads to a target “behind” the outer control structure. For this reason, we call the right model unstructured, while the left one is well-structured.

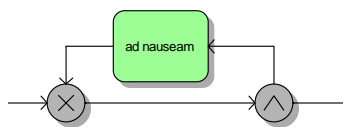


Figure 2. Livelock

A business process model should be free of deadlocks and livelocks. Informally spoken, a deadlock blocks the continuation of the process because two control flow paths cannot synchronise. An example is a model with a XOR split (which results in an exclusive choice of *only one* of several paths) that is followed by an AND join (which waits for the completion of *all* incoming paths.) A livelock prevents the process to come to an end because the execution runs into an infinite cycle; an example is shown in Fig. 2.

It can be shown that well-structured models are free of deadlocks [21]. While structured models *guarantee* to have these desirable properties, unstructured ones *can* have these properties.

Another reasonable requirement is that there are no “dead parts” in the model, i.e. there are no activities that cannot be reached from the start node.

Van der Aalst defined the soundness property for business process models. It is defined formally in terms of Petri-nets in [24]; informally spoken soundness means that:

1. There are no deadlocks and livelocks.
2. When an end activity is reached, no activities are or can become active (in terms of Petri nets this means that there are no tokens left on the net).
3. The model does not contain “dead parts”, i.e. all activities can be reached from the start node.

Other “soundness” definitions (whose requirements are less strict) are summarised in Tab. 1. Details can be found in the referenced literature. For all patterns suggested in this article, we give an information about their soundness properties.

## 4 Related Work

It has been shown that in most cases, unstructured models can be replaced by structured ones[9, 13, 7]. These approaches use automatic transformations. Such transformations do not necessarily have to result in models that are easy to comprehend by a human reader. However, the main purpose of business process models is to ease the discussion about a business process. This means that understandability is very important for these models and automatic transformations often do not work properly.

To transform arbitrary business process models into *readable and compact* BPEL code (which is well-structured by definition) is the aim of the tool WorkflowNet2BPEL4WS[11, 12]. The algorithm presented in [11] makes use of a component library which includes known modelling constructs and their mapping to structured BPEL code. What is called components in [11, 12] is very similar to what we call patterns in our paper. The focus of [11] lies on the transformation algorithm, not on the presentation of the the components (patterns).

Apart from this work, all other papers on patterns for business process modelling are on another abstraction level than ours. The focus of the well-known workflow control-flow patterns[1, 25] lies on evaluating the expressive power of workflow modelling languages. These patterns show which basic control-flow constructs (like “sequence”, “parallel branching” etc.) should be supported by workflow modelling languages and workflow management systems. This means that these patterns are on a lower level of abstraction. The same is true for business process patterns that deal with the data-flow[19], the resource perspective[20] or object/service interactions[2, 26, 16]. For an overview on these pattern catalogues, we recommend the web site [www.workflowpatterns.com](http://www.workflowpatterns.com). On the other side of the abstraction level scale, reference model catalogues like the the SAP R/3 reference model[4] or the MIT Process Handbook[14] contain customisable templates for domain-specific business processes as a whole.

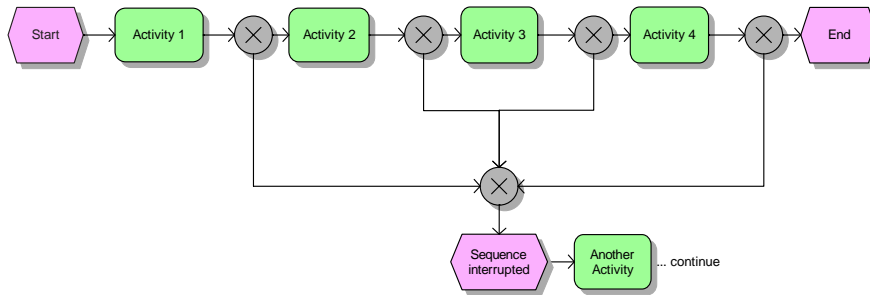


Figure 3. Pattern “Straight Road with Exits”

## 5 Pattern: Straight Road with Exits

**Intent:** A business process contains a number of activities that have to be performed sequentially. After certain activities, it should be possible to stop the execution of this sequence of activities and to continue the execution of the process with some other thread of control.

**Example:** Before approving a real-estate loan request, several documents (purchase contract, estimate of renovation costs etc.) have to be reviewed. If one document is not satisfying, an information is sent to the applicant, and no further documents need to be reviewed.

**Problem:** There are several possibilities to leave the sequence of activities. This means that there are several decision points. Using a well-structured model for such a situation can result in a model where control flow blocks are deeply nested, because every decision point adds one pair of split and join node. Such models can be hard to read[3].

**Solution:** see Fig. 3

**Category:** Good excuse for unstructured modelling

**Model Properties:**

well-structured	no
sound	✓
weak sound	✓
relaxed sound	✓
lazy sound	✓
single-entry-single-exit	no

**Discussion:** The model is not well-structured, because one join node corresponds to several split nodes. However, it is very easy to understand anyway.

In a situation where the exit from the sequential execution can be regarded as an exception, the modelling element “exception”, combined with the modelling element “interruptible activity region” should be used instead if the modelling language supports these modelling elements. This allows to “hide” the unstructuredness behind the exception. In a language like Event-Driven Process Chains that does not allow exceptions; the situation should always be modelled as shown in Fig. 3.

**Related Patterns:** The control-flow patterns Arbitrary Cycles and Cancel Case[25] are similar to “Straight Road with Exits”. The difference is as follows: In the Arbitrary Cycles pattern, some activities are *repeated* by cycling back in the process. In the Cancel Case pattern, the execution of the business case *is stopped* when the “exit” is reached. In the “Straight Road with Exits” pattern, *some other activity will be started* after the “exit”. The relation to the “All or Nothing” pattern is discussed in Sect. 6.

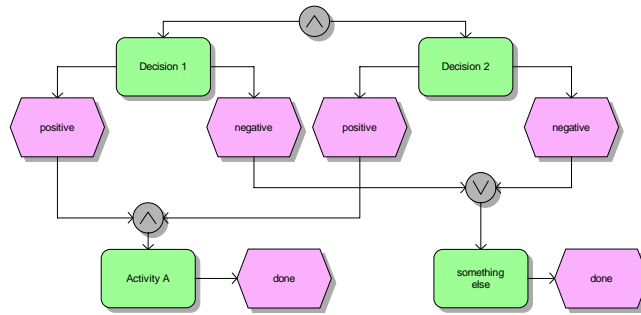


Figure 4. Pattern All Or Nothing

## 6 Pattern: All Or Nothing

**Intent:** A decision about whether some activity A should be performed is made by the parallel execution of a number of decision making activities. Only if all these activities come to a positive result, activity A will be performed. Otherwise, the process proceeds with some other activity.

**Example** (taken from [22]): The decision regarding the acceptance or the refusal of a customer order is made by the parallel execution of various sub-functions. The customer order is checked for technical feasibility and in addition, the customer's creditworthiness and the availability of the product are determined. One negative result, for example "Order is not technically feasible" or "Poor credit rating" leads to the rejection of the customer order.

**Problem:** No communication takes place between the performers of the decision making activities. For this reason, the decision how to proceed in the business process cannot take place before all of these activities have been performed.

**Solution:** see Fig. 4. In order to keep the example simple, this model shows two decision making activities only.

**Category:** Bad excuse for unstructured modelling

**Model Properties:**

well-structured	no
sound	no
weak sound	✓
relaxed sound	no
lazy sound	no

**Discussion:**

Such a model contains a deadlock at the AND-join. If at least one decision making activity come to a negative result, not all incoming arcs of the AND-join are activated and the AND-join cannot synchronise. In fact, this deadlock is used deliberately to block the flow of control in order to avoid the execution of activity A.

However, besides of this (rather theoretical) deadlock problem, there are also severe practical issues. We can distinguish two cases: Either each of the decision making activities can be performed very fast or the execution of some of these activities need a considerable amount of time. If each of the decision making activities can be performed very fast, there is actually no need to perform them in parallel. This means that the business case can easily be modelled with the pattern "Straight Road with Exits" (see Section 5).

On the other hand, if the execution of some of the decision making activities needs a consid-

erable amount of time (which means that these activities are expensive from a business management point of view), it is not desirable that once one decision making activity made a negative decision any of the other decision making activity still have to be performed. Even only one negative decision leads to the conclusion that activity A should not be performed, regardless of the other decisions.

There are two possible solutions to deal with these requirements: Either the decision making activities are not performed in parallel (which means that we can use the pattern “Straight Road with Exits” as described in Sect. 5) or every negative decision of one decision making activity should lead to the cancellation of all other decision making activities. Unfortunately, the modelling language EPC that is used in this paper does not support the modelling concept of cancellation, but other modelling languages like UML activity diagrams, BPMN or YAWL do. Another possibility for replacing the model fragment shown in this pattern is to use Exceptions. This is discussed in the following subsection that shows a variant of the pattern.

**Variante:** Several paths of control are processed in parallel. In every of these paths, an exception can lead to the abortion of the execution of the business case. This is quite the same situation as shown in Fig. 4. The only difference is that the negative decision is now called “Exception”. In a modelling language that supports exception handling (like UML activity diagrams or BPMN), the best way to deal with this situation is to make use of exceptions. Care has to be taken to define the interruptible activity region correctly. In the original pattern, using exceptions for dealing with negative decisions that can be regarded as a “normal” flow of control can be misleading. If a negative decision is not really an exception but just a normal choice not to perform activity A but to proceed with some other activity, the reader of the model could be misguided when the modelling concept “Exception” is used.

**Related Patterns:** As discussed above, this pattern is related to the the “Straight Road with Exits” pattern. It is also related to the control flow patterns Cancel Activity and Cancel Case[25]. Cancel Activity can be used in order to prevent unnecessary work when one decision making activity already has been come to a negative result. If no “cleanup” or logging is necessary, one negative decision can even trigger the cancelling of the whole business case.

## 7 Pattern: Redo Something Instead of Everything

**Intent:** To achieve some goal, several activities are executed in parallel. After their completion, a final activity “Check Quality” decides whether the result is admissible or not. In case of a negative result, one of the activities that have been executed in parallel must be repeated.

**Example:** Two months before the start of a new academic year at a university, the enrolment process starts. At the same time, the course schedule is planned. The deadline for both activities is one week before the start of the academic year. Because the course schedule has been planned without knowing how many students will enrol, it is very likely that some courses need to be re-scheduled, for example because they need to take place in a larger lecture room. For this reason, the course schedule has to be compared to the number of students after the enrolment process. If the course schedule does not fit the actual enrolment numbers, the activity “build course schedule” needs to be repeated.

**Problem:** In order to avoid redundancies in the model, it is not desirable that the activity which might be repeated is modeled more than once.

**Solution:** see Fig. 5.

**Category:** *Can* be a good excuse for unstructured modelling (but often it is not)

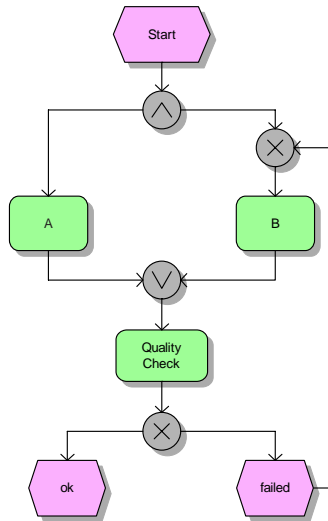


Figure 5. Pattern “Redo Something Instead of Everything”

**Model Properties:**

well-structured	no
sound	✓
weak sound	✓
relaxed sound	✓
lazy sound	✓

Another property that is worth mentioning is that the model’s semantics is unclear if we use the non-local semantics definition by Kindler[10]. However, the discussion of such semantical issues is out of the scope of this paper, the interested reader can find details in [10]. We assume the following semantics that should comply with the intuition of everyone who reads the model: The parallel activities are always executed, and if both have been finished, the quality check takes place. If it fails, one of the activities will be re-executed again and again as long as the quality check does not come to a positive result.

**Discussion:**

The model is unstructured, because the AND-split is not matched by an AND-join. Instead, the parallel activities are joined by an OR-join. This does not “look nicely”, but is modeled correctly: If only one activity needs to be re-executed, there must not be an AND-join that would have to synchronise *both* flows of control. The modeller could avoid the unstructuredness by using a model as shown in the leftmost model in Fig. 6. The advantage of the original model in Fig. 5 is that Activity B needs to be modeled only once. For this reason, the pattern shown in Fig. 5 **can** be a good excuse for the unstructuredness.

However, when using this pattern, the modeller should be sure that the answer to all of the following questions is “yes”:

1. Should the activities really run in parallel?
2. Is it really correct to repeat only one of the parallel activities? (Otherwise, if all activities should be repeated, the correct model should look like the middle model in Fig. 6.)
3. Does the activity “Check Quality” really has to wait for the completion of all parallel paths? (Otherwise, the modeller should prefer the structured model shown in the rightmost

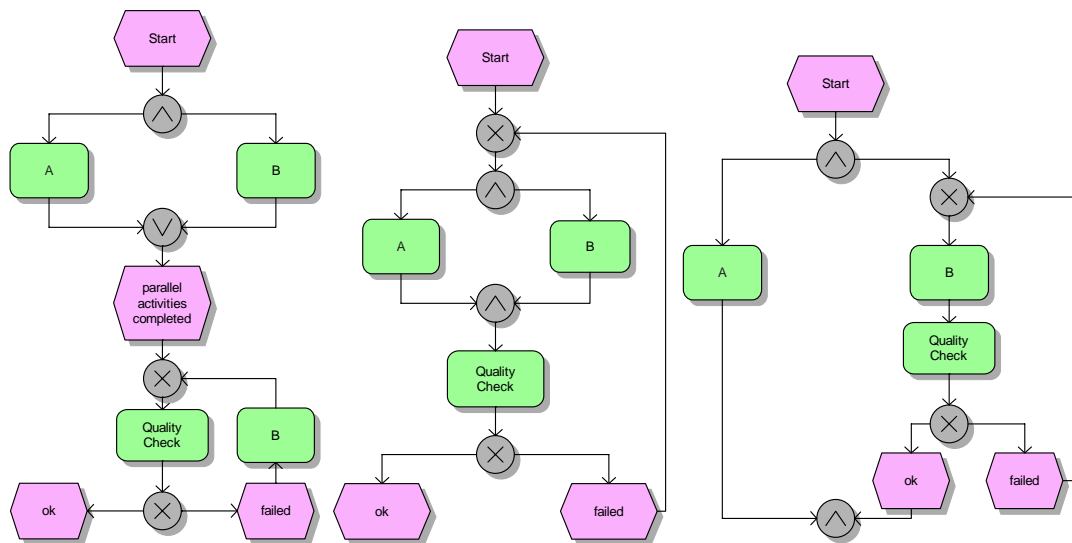


Figure 6. Alternative Solutions

model in Fig. 6.)

4. Is it really appropriate to *repeat* one activity when the quality check fails or should some other activity be performed instead? (In our example given above, one may argue that this is the case: Instead of building a completely new course schedule, most likely the existing one should be changed. It depends on the purpose of the model whether this should be regarded as being a different activity.) If the answer to question 4 is “no”, the problem should be modelled in a way similar to the left model in Fig. 6, but instead of repeating activity B, a “repair activity” B’ should be performed.

In our analysis of real-world models that used this pattern, we found that *in most cases* not all of the above questions have to be answered with “yes”. Therefore, these models should be changed as discussed above.

The implication from the discussion is that when the pattern occurs in a model, the modeller should answer the above questions in order to decide whether the model meets his intention. If all answers are “yes”, it’s fine. However, often one answer will be “no”, and it is advisable to change the model.

**Related Patterns:** This pattern is related to the Arbitrary Cycles[25] control flow pattern.

## 8 Conclusion

The three patterns presented in this paper show examples for three different situations we have to deal with when we have to review an existing unstructured business process model:

1. There can be a good reason for the unstructuredness. A program that computes complexity metrics for business process models should consider this when computing the complexity of the model. (example: *Straight Road with Exits*)
2. The unstructured part of the model should be considered as being wrong (or at least bad). It should be replaced by another alternative if the modelling language supports this. (example: *All Or Nothing*)

3. The unstructuredness might indicate that the model does not comply with the real-world business process. A review of models containing such patterns is advisable. Often, the result of the review will be that the model should be changed. (example: *Redo Something Instead of Everything*)

We are currently analysing a collection of business process models from various sources in order to build a comprehensive pattern catalogue. Currently, we focus on the control-flow perspective, other aspects like data flow or resource utilisation could be the subject of future work.

We would like to invite the reader to participate in building the pattern catalogue. Contributions and comments are welcome to `laue@ebus.informatik.uni-leipzig.de`.

## References

- [1] W. Aalst, A. van der, B. Hofstede, and A. Kiepuszewski. Advanced workflow patterns. In O. E. en P. Scheuermann, editor, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer-Verlag, Berlin, 2000.
- [2] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Service interaction patterns. In *Business Process Management*, pages 302–318, 2005.
- [3] J. Cruz-Lemus, M. Genero, M. Piattini, and A. Toval. Investigating the nesting level of composite states in uml statechart diagrams. In *9th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2005)*, 2005.
- [4] T. A. Curran and A. Ladd. *SAP R/3 business blueprint (2nd ed.): Understanding enterprise supply chain management*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
- [5] J. Dehnert and P. Rittgen. Relaxed soundness of business processes. In *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, pages 157–170, London, UK, 2001. Springer-Verlag.
- [6] V. Gruhn and R. Laue. Complexity metrics for business process models. In *9th International Conference on Business Information Systems (BIS 2006)*, Klagenfurt, Austria, 2006.
- [7] R. Hauser, M. Friess, J. M. Kuster, and J. Vanhatalo. Combining analysis of unstructured workflows with transformation to structured workflows. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 129–140, Washington, DC, USA, 2006. IEEE Computer Society.
- [8] A. Holl and G. Valentin. Structured business process modeling (SBPM). In *Information Systems Research in Scandinavia (IRIS 27) (CD-ROM)*, 2004.
- [9] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *Conference on Advanced Information Systems Engineering*, pages 431–445, 2000.
- [10] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In *Business Process Management*, pages 82–97, 2004.
- [11] K. B. Lassen and W. M. van der Aalst. Translating workflow nets to BPEL, booktitle = BETA Working Paper Series, WP 145, Eindhoven University of Technology, Eindhoven, year = 2005,.
- [12] K. B. Lassen and W. M. van der Aalst. Workflownet2bpe14ws: A tool for translating unstructured workflow processes to readable BPEL. In *BETA Working Paper Series, WP 167, Eindhoven University of Technology, Eindhoven, 2006*.
- [13] R. Liu and A. Kumar. An analysis and taxonomy of unstructured workflows. In *Business Process Management*, pages 268–284, 2005.
- [14] T. W. Malone. *Organizing Business Knowledge*. The MIT Press, 2003.
- [15] A. Martens. On Compatibility of Web Services. *Petri Net Newsletter*, 65:12–20, 2003.

- [16] M. Paludo, R. Burnett, and E. Jamhour. Patterns leveraging analysis reuse of business processes. In *ICSR-6: Proceedings of the 6th International Conference on Software Reuse*, pages 353–368, London, UK, 2000. Springer-Verlag.
- [17] F. Puhmann and M. Weske. Investigations on soundness regarding lazy activities. In *Business Process Management*, pages 145–160, 2006.
- [18] E. S. Roberts. Loop exits and structured programming: reopening the debate. In *SIGCSE '95: Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education*, pages 268–272, New York, NY, USA, 1995. ACM Press.
- [19] N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In *Conceptual Modeling - ER 2005, 24th International Conference on Conceptual Modeling*, pages 353–368, 2005.
- [20] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005*, pages 216–232, 2005.
- [21] W. Sadiq and M. E. Orłowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, June 2000.
- [22] O. Thomas, O. Adam, and P. Loos. Using reference models for business process improvement: A fuzzy paradigm approach. In *9th International Conference on Business Information Systems (BIS 2006), Klagenfurt, Austria*, pages 47–57, 2006.
- [23] W. M. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [24] W. M. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology*, 41(10):639–650, 1999.
- [25] W. M. van der Aalst, A. Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3), 2003.
- [26] J. yoon Jung, W. Hur, S.-H. Kang, and H. Kim. Business process choreography for B2B collaboration. *IEEE Internet Computing*, 08(1):37–45, 2004.