Reducing the Cognitive Complexity of 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

Current research on the quality of business process models mostly discusses the quality of a model mainly in terms of correctness of its behaviour. In this paper, we discuss another perspective: Because the main purpose of business process models is to be read and understood by humans, understandability is an important requirement. Some patterns for improving the understandability by reducing the cognitive load are given. By applying these patterns to a repository of almost 1000 models, we have been able to improve the comprehensibility of several models.

Keywords: cognitive complexity, business process models

1 Introduction

Business process models (BPM) are graphical representation of current or to-be business processes in an organisation. They play an important role not only in the field of business process management but also as an artifact in the analysis stage of the software development cycle. Often domain experts, business analysts and information system developers use graphical BPM when communicating with each others. The models help to bridge the well-known “language-barrier” between domain and IT-experts.

2 Business Process Modelling Languages

There are several visual languages for modelling business processes. The Business Process Modeling Notation (BPMN) is the most widespread language, for this reason we use it for the examples in this paper. However, the techniques described in this paper can also be used for other languages, because most BPM language share some basic constructs. In this section, we will shortly describe those basic constructs of the BPMN language. For more details about BPMN, the reader is referred to [4].

Events (something that happens during the lifetime of a business process) are represented with a circle. Although not formally required by the standard, every BPMN model should have at least a start event (depicting the fact that the process is instantiated) and an end event (depicting the fact that the process has been completed).

Activities (tasks that have to be performed) are represented with a rectangle with rounded corners. The flow of control (called sequence flow in BPMN terminology) between the activities is depicted by arcs. The direction of such an arc shows in which order the activities have to be performed. In Fig 2, Activity A has to be performed right after the start event and activity B has to be performed after the completion of activity A.

![Figure 2. Start event followed by a sequence of two activities](image)

Gateways can be used for forking and joining paths that have to be performed in parallel or (based on certain conditions) alternatively. There are two kinds of gateways: Splits have more than one outgoing arc, and joins have more than one incoming arc. Gateways are represented with a diamond shape. In BPMN, there are five different kinds of such gateways. In this paper, we restrict ourselves to those three kinds of gateways that can also be found in other important modelling languages (in particular, EPC and YAWL).

When used as a split, an exclusive gateway splits the sequence flow to exactly one of its outgoing branches. When used as a join, it awaits one incoming branch being completed before triggering the outgoing flow. This kind of gateways is depicted by a diamond shape with the × symbol inside. We will refer to this kind of gateways as XOR-gateways.

When splitting, a parallel gateway activates all outgoing branches; the activities on these branches are executed...
in parallel. When used as a join, a parallel gateway waits for all incoming branches to complete before triggering the outgoing flow. Parallel gateways are depicted by a diamond shape with the + symbol inside. We will refer to this kind of gateways as AND-gateways.

The inclusive gateway is something in-between the exclusive and the parallel gateway. When used as a split, some of the outgoing branches (but at least one) are activated. When merging, an inclusive gateway waits until all active incoming branches have been completed before triggering the outgoing flow. Inclusive gateways are depicted by a diamond shape with the ⃝ symbol inside. We will refer to this kind of gateways as OR-gateways.

Fig. 1 contains all mentioned kinds of gateways. The model shows a simplified business process in a bank. When a customer applies for a real-estate credit, the customer’s credit rating, the real estate construction documents and the land register record are checked. All these activities are done in parallel, therefore an AND-gateway is used in the model. As the result of those assessments, the application has either to be rejected or the contract is to be prepared. As only one of the activities "Reject Application" and "Prepare Contract" can take place, the XOR-gateway is used for forking the sequence flow. Finally, after the contract has been prepared, the process either can end or the bank might offer some additional products: a loan protection insurance and a residence insurance. Whether a loan protection insurance, a residence insurance or both are offered, has to be decided case-by-case. The OR-gateway shows that only one of the activities or both of them (in parallel) can take place.

3 Cognitive Complexity of Business Process Models

We can see from the example in Fig. 1 that split and join connectors do not have to occur pairwise. In [15], we have shown that unstructuredness in a BPM, i.e. splits that have not a matching join of the same type, is strongly correlated to semantic errors (for example deadlocks) in the model. This result is similar to previous findings on how the (un)structuredness of software impacts maintainability[23, 9].

However, “absence of semantic errors” (in terms of the soundness property defined for business process models[30, 28, 21, 7]) is not the only requirement for business process models: One of the main purposes of these models is to be used as a language to be used in a discussion between humans. In particular, the models can serve as a bridge between the stakeholders in a software development project. They are formal enough to serve the demands of software developers but easy enough to be understood by business experts as well.

With the aim to improve the understandability of BPM, we take advantage of results from psychology, in particular from the cognitive load theory. It has been shown that a reader understands visual models by identifying smaller submodels (which can be understood as a whole) and connecting those submodels later. [27, 8, 20].

When mentally connecting parts of a BPM, the necessary cognitive processing capacity corresponds to the number of control flows between those parts that the reader has to consider. The cognitive load can be reduced by eliminating unnecessary sequence flows from the mental model that the reader has to construct.

4 Finding and Removing Unnecessary OR-Gateways

Sarshar and Loos[25] found out in a study on the comprehensibility of BPM that OR-gateways where significantly less comprehended than AND- or XOR-gateways. One explanation for this result is that the reader has to consider more possible combinations of outgoing / incoming sequence flows.

Using the ideas from [5], we can ask for the number of mental states that a reader of a model has to build in order to understand the prerequisites for executing an activity that is placed after a join gateway with \( n \) incoming flows.

For an AND join gateway, this is rather easy: Always all incoming sequence flows of the join gateway have to be completed, i.e. there is only one possible combination.
For an XOR-join gateway with \( n \) incoming flows, exactly one of the \( n \) incoming flows has to be completed, i.e. there are \( n \) possible combinations. However, the semantics of an OR-join gateway with \( n \) incoming flows is that any of the \( 2^n - 1 \) possible combinations for which at least one incoming flow has been completed has to be considered. Among all gateways, the OR-gateway (inclusive join) leaves the most possibilities. Analogously, an OR-split is more difficult to comprehend than an AND- or XOR-split.

The cognitive load can be decreased by substituting OR-gateways whenever this is possible without changing the semantics of the model. Luckily, this is often the case: We remember that the semantics of an OR-join gateway is to wait for the completion of all active incoming branches. When an XOR-split gateways is paired by an OR-join gateway (which is not forbidden, because the semantics of most BPN languages does not imposes well-structuredness requirements on the model), exactly one of its outgoing sequence flows become active. After performing the activities along this sequence flow, the OR-join gateway (which waits for the completion of all active incoming branches) has to wait until this single active sequence flow reaches the join. This means, that an OR-join gateway can be replaced by a XOR-join gateway which describes the same semantics much better.

Similarly, when an AND-split gateway is paired with an OR-join gateway, the join gateway can (and should!) be replaced by the easier-to-understand AND-join gateway. The same is true when an OR-join gateway is used as an entry into a loop. The three cases described before are shown in Fig. 3. Please note that while the model fragments in 3 are rather trivial, they only serve as simple examples. The replacement can also be applied if instead of one activity A and B there are arbitrary complex structures along the sequence flows. In fact, the more complex the model fragment is, the more the reader can profit from replacing the OR-gateways.

5 Empty Sequence Flows

A situation that is common in a business process is when - based on a condition - some activities either have to be performed or can be skipped completely. Such a situation can easily be modelled by a pair of XOR-gateways as shown in Fig. 4.

However, the same situation could be modelled using a pair of OR-gateways. In addition to the possibilities "the lower sequence flow (with activity A) is followed" and "the upper sequence flow (without any activity) is followed", the model would also allow a third one: "Both sequence flows are followed". However, as there are no activities on the upper sequence flow, this is in fact the same as "the lower sequence flow with activity A is followed". Once again, it makes sense to replace the OR-gateway by an XOR-gateway which expresses the desired situation more precisely.

A sequence flows without any activities that are started and later joined by AND-gateways is completely useless in a model. The meaning of such a model would be that the "empty arc" has to be followed in parallel to the activities on the other sequence flow(s) which does not make any sense at all. Surprisingly, such constructions can be found anyway in real-world models (see Sect. 8). If they can be identified, the useless arc (and - if there are no additional parallel sequence flows - the split and join gateways as well) can be deleted without affecting the semantics of the model.

Two less frequent patterns where a simplification of the model is possible are shown in Fig. 6 and Fig. 7. Replacing the OR-split in Fig. 6 makes it easier to comprehend the situation, because the number of mental states required to...
understand the behaviour is reduced from 3 to 2.

Fig. 7 is a kind of variant of the lower case in Fig. 4. One might suppose that such useless constructs do not appear in real-world models, but we have indeed found them in examples from our model repository (see Sect. 8).

Fig. 8 shows a situation where the same activity is located at the last position of all incoming paths of a join. In such a case, it can be moved outside the control block. Fig. 8 shows only one of many different cases: The gateways does not have to be AND-gateways, and to move one or more identical activities out of a control block is also possible if identical nodes directly follow a split node.

Note however, that a replacement is not possible if the "identical" activities are placed in different swimlanes of a BPMN diagram, because this means that they have to be performed by different organizational units.

### 6 Reducing the number of model elements

An obvious way to improve the understandability of a BPM is to reduce the number of its nodes. Many authors have pointed out that models with less nodes are easier to understand[3, 26, 19].

Fig. 8 shows a situation where the same activity is located at the last position of all incoming paths of a join. In such a case, it can be moved outside the control block. Fig. 8 shows only one of many different cases: The gateways does not have to be AND-gateways, and to move one or more identical activities out of a control block is also possible if identical nodes directly follow a split node.

As we cannot assume that split and join gateways are properly nested, we also have to define when we regard a split gateway to be matched by a join gateway. For this purpose, we define the clause $\text{match}(S, J)$ such that $S$ is a split gateway, $J$ is a join gateway and there are two sequence flow paths from $S$ to $J$ whose only common elements are $S$ and $J$.

Once we have these rules defined, we can express the patterns described in the previous sections with a few lines of code. For example, in order to find instances of the case in the second line of Fig. 3, we have to search for an OR-join gateway $J$ for which all split gateways with $\text{match}(S, J)$ are XOR-split gateways. Furthermore, we have to require that $J$ is not entry node into a loop and that $J$ cannot receive a sequence flow coming from another start event. These cases are illustrated in Fig. 9.

Using Prolog, the patterns can be specified very easily, and they can be found very fast in existing models. However, our pattern-based approach does not necessarily have to be used with Prolog or another logic-based language. We have already implemented a pattern-finding algorithm in the open source modelling tool $\text{bflow}^*$ using the languages $oAW Check$ and $\text{XTend}$ from the openArchitectureWare\(^1\) model management framework[14]. Currently, we are working on an implementation using the query language BPMN-Q[2].

---

\(^1\)http://www.eclipse.org/gmt/oaw/
Pattern                          | Problems found          |
--------------------------------|--------------------------|
XOR/AND split combined with OR-join (Fig. 3, first and second line) | 72 occurrences in 40 models |
OR-join as a loop entry (Fig. 3, third line)                        | 30 occurrences in 25 models |
OR split/join used for optional execution (Fig. 5, first line)      | 3 occurrences in 3 models |
useless empty flow from AND-split to AND-join (Fig. 5, second line)  | 28 occurrences in 19 models |
OR-split in the situation of Fig. 6                                 | 1 occurrence in 1 model   |
useless elements in the situation of Fig. 7                         | 2 occurrences in 2 models |

Table 1. Problems found

![Diagram](image)

Figure 9. We cannot replace the OR-join gateway in these cases.

8 Empirical Results

We searched for the patterns defined above in a repository of 984 models that have been collected from 130 sources. These sources can be categorized as follows:

- 531 models from the SAP R/3 reference model, a widespread business reference model
- 112 models from 31 bachelor and diploma theses
- 25 models from 7 PhD theseses
- 13 models from 2 technical manuals
- 82 models from 48 published scientific papers
- 12 models from 6 university lecture notes
- 4 models from sample solutions to university examination questions
- 88 models from 11 real-world projects
- 88 models from 7 textbooks
- 29 models from 14 other sources

Among the models in our repository, there is a great variation in size of the models, purpose of modelling, business domain and experience of the modellers. For this reason, we think that the models represent a reasonable good sample of real-world BPM.

The models in our repository have been created using the BPM language Event Driven Process Chains (EPC)[29]. The main reason for this was the availability of a large number of models in this notation. However, the application of patterns for minimizing the cognitive load would be the same for models in other languages like the BPMN notation that has been used in this paper, because most languages share the main elements and in particular the three types of gateways. Because of subtle differences among the modelling languages\(^2\), the number of redundant activities as shown in Fig. 8 that can be found in EPC models is not representative for other modelling languages, we have therefore not included this class of problems in the reporting of pattern instances found.

The distribution of patterns found in the models is shown in Tab. 1. Altogether, our analysis found 136 suggestions for reducing the cognitive complexity in 73 models. An interesting finding was that from the 443 OR-joins in the models, no less than 106 could be replaced by an AND- or XOR-join which not only models the given situation more precisely but also reduces the cognitive load when someone has to read the model.

9 Related Work

There are several good guidelines on how to create BPM that are correct and understandable [3, 19, 1] as well as useful compilations of common patterns for bad modelling[13, 26].

However, there is no much related work on actually identifying understandability problems in existing models. As already mentioned, most work on formal analysis of business process models focus on the correct behaviour of the model.

Remarkable exceptions of tools that find certain kinds of comprehensibility problems are the model checking approach by Rump [24] and the Petri-net based approach used in YAWL [32] which both can find OR-joins that can be replaced by more meaningful AND- or XOR-joins. Because

\(^2\)EPCs require the state, i.e. some kind of postcondition, after the execution of each task to be modelled explicitly which lead to a somewhat different structure of the models.
both approaches require the full statespace of the models to be explored in order to identify problems, they can be too time-consuming for running in background at modelling time when used with a complex BPM.

ArgoUML [22] is a modelling tool that helps the user to improve UML models by applying design critics. This approach has several similarities to ours, but most of the design critics currently implemented in ArgoUML are just checking the correctness of a model on a syntactical level.

In another direction of research, complexity metrics are used to measure the complexity of a model. Here, complexity means how difficult a model is to understand, to test and to maintain. A large complexity can be regarded as a recommendation to simplify the model without directly pointing to the problematic modelling elements. An overview on metrics for BPM complexity can be found in [10, 6, 17].

Even if there are some first attempts to study the relationship between those complexity metrics and the understandability of models[18, 31], there are still many unanswered questions[16].

10 Conclusions

In our analysis, we found that in a considerable part of real-world models, an automatic search for the patterns given in this paper can lead to an improvement of the comprehensibility by reducing the cognitive load. Comprehensibility problems which can lead to potentially costly misunderstanding can be prevented by replacing the modelling elements that are unnecessarily difficult to comprehend. The pattern finding algorithm can be implemented into a modelling tool such that it runs continuously in background and alerts the modeller as soon as a problem occurs[14]. It would also not be difficult to implement a auto-correction feature.

Future directions of research include the inclusion of more modelling elements in order to reduce the complexity of situations where cancelation of processes and exceptions in the control flow are involved[12].

References