# Generating Natural Language Texts from Business Process Models

Henrik Leopold[1], Jan Mendling[2], and Artem Polyvyanyy[3]

[1] Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
henrik.leopold@wiwi.hu-berlin.de
[2] WU Vienna, Augasse 2-6, A-1090 Vienna, Austria
jan.mendling@wu.ac.at
[3] Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
artem.polyvyanyy@hpi.uni-potsdam.de

**Abstract.** Process Modeling is a widely used concept for understanding, documenting and also redesigning the operations of organizations. The validation and usage of process models is however affected by the fact that only business analysts fully understand them in detail. This is in particular a problem because they are typically not domain experts. In this paper, we investigate in how far the concept of verbalization can be adapted from object-role modeling to process models. To this end, we define an approach which automatically transforms BPMN process models into natural language texts and combines different techniques from linguistics and graph decomposition in a flexible and accurate manner. The evaluation of the technique is based on a prototypical implementation and involves a test set of 53 BPMN process models showing that natural language texts can be generated in a reliable fashion.

**Keywords:** Natural Language Generation, Verbalization, Business Process Models

## 1 Introduction

Business process modeling is nowadays an integral part of information systems engineering and of organizational design. Many organizations document their operations in an extensive way, often involving several dozen modelers and resulting in thousands of business process models [1]. The audience of these models is even bigger, ranging from well-trained system analysts and developers to casual staff members who are unexperienced in terms of modeling. Most of the latter lack confidence to interpret visual process diagrams. Clearly, there is a divide between the persons with modeling expertise and those without.

This problem of diverging skills is reflected by modeling methods which explicitly distinguish between modeling experts and domain experts. In this setting, *system analysts* have to formalize facts about a specific domain with which they are often not familiar. *Domain experts* have to provide details about this domain although they do not fully understand the models the analysts create.

Therefore, the validation of the models has to rely on a discourse in natural language. For data modeling, this concept is supported by Object-Role Modeling (ORM) and its predecessor NIAM [2,3]. A key feature of ORM is a direct mapping from models to natural language text called *verbalization*. This verbalization capability has been emphasized as a crucial advantage for the validation of models in a discourse between system analyst and domain expert [4].

The reason why a verbalization technique for process models is missing might be a result of several facts. First, the validation of process models in general is difficult. Domain experts are typically not familiar with fundamental process concepts such as concurrency, decision points or synchronization. In the same vein, process models are also more difficult to translate into natural language text. The non-sequential structure of a process model has to be serialized into sequential, yet execution-order preserving text. Furthermore, activity labels have to be parsed into fragments of verb-phrases, which have to be used for constructing sentences. Such a procedure has to take optionality of information (e.g. using passive voice if no actor is mentioned) explicitly into account. Finally, the overall text has to be structured in such a way that the reader can understand the process effectively.

In this paper, we address the challenge of automatic verbalization for process models. Our contribution is a technique that is able to generate natural language text from a process model, taking into account the issues of parsing text labels, sequentializing the structure of the process, as much as flexible sentence and text planning. We deem this technique to be beneficial not only for process model validation, but also for various scenarios where diagrams have to be translated into a corresponding piece of text. These include the generation of work instructions from process models, creating process handbooks, or publishing processes on the intranet to an audience that might have little experience with process modeling.

The paper is structured as follows. Section 2 introduces the background of our work. We illustrate the text generation problem by the help of an example and identify a list of challenges. Section 3 defines our generation technique. It addresses the problems of sequentializing the process model, parsing the labels and flexibly planning sentences and text structure. Section 4 provides an evaluation of this technique using a collection of process models from practice. Section 5 discusses related work before Section 6 concludes the paper.

## 2 Background

Our solution for generating natural language texts from process models builds on general process modeling concepts and natural language architectures. In this section, we introduce BPMN as the process modeling language upon which we define our approach. Furthermore, we provide an overview of natural language generation architectures and the challenges associated with generating text from process models.

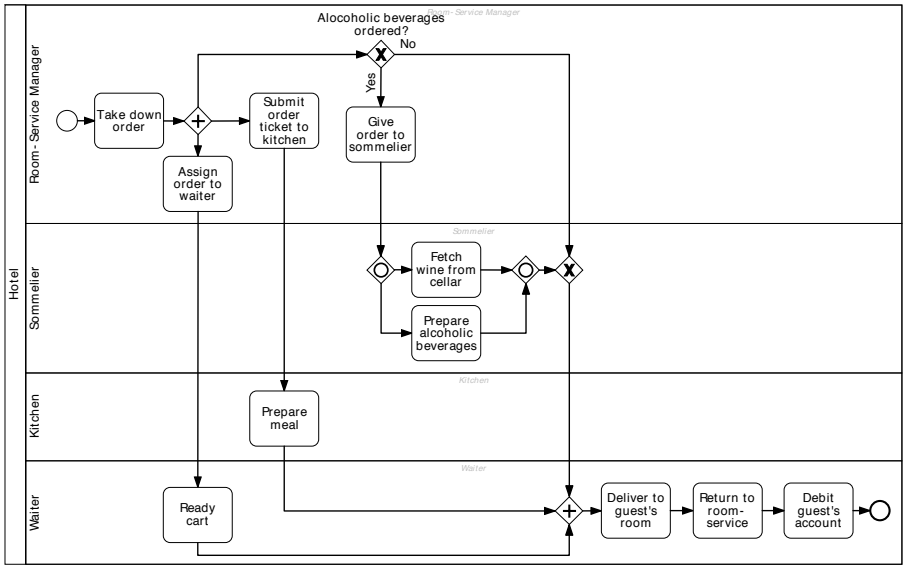



**Fig. 1.** Exemplary BPMN Process

### 2.1 Business Process Model and Notation

The Business Process Model and Notation (BPMN) is a process modeling standard providing a graphical notation for the specification of business processes. Recently, it was published in its current version 2.0 [5]. The symbol set of BPMN covers four types of elements: flow objects (activities, events, gateways), connecting objects (sequence and message flow, associations), swim lanes (pools and lanes) and artifacts (e.g. data objects and annotations).

Figure 1 shows an example of a business process in a hotel represented as a BPMN model. It describes how an order is handled. The process includes four roles and is hence subdivided into four lanes. The activities performed by the different actors are depicted as boxes with round corners, the diamond shaped gateways define the routing behavior. The plus in a gateway specifies a parallel execution, the circle an inclusive choice (one or more branches can be executed) and the cross an exclusive choice (only one branch can be executed). The process starts with the room-service manager taking an order. This triggers three streams of action: a meal is prepared in the kitchen, beverages may be prepared if required, and the delivery is arranged by the waiter.

### 2.2 Architectures of Natural Language Generation Systems

In general, there are different techniques available in order to translate process models into natural language text. Simple, also often called non-linguistic, approaches are based on canned text or templates-based systems. In the first case

some input data is directly mapped to a predefined string. For instance, a system translating weather data into natural language text could use the sentence *The weather will be nice today* for expressing a warm and sunny day. Slightly more advanced is the use of templates where at least some information is added to the predefined string. In this case, the template *Today there is a X% probability of rain* could be filled with the according rain probability derived from a data source. However, such approaches are not considered to be truly linguistic techniques as the manipulation is done at the character string level [6].

Linguistic, or *real* natural language generation approaches, use intermediate structures to obtain a deeper representation of the text. Such an intermediate structure usually specifies the main lexemes (abstract representations of words encapsulating inflectional aspects) for each sentence. Moreover, it carries additional information, defining for instance the tense or the modus of the verb. As pointed out by Reiter, many natural language generation systems take a three-step pipeline approach including the following stages [7]:

1. **Text Planning**: First, the information is determined which is communicated in the text. Furthermore, it is specified in which order this information will be conveyed.
2. **Sentence Planning**: Afterwards, specific words are chosen to express the information determined in the preceding phase. If applicable, messages are aggregated and pronouns are introduced in order to obtain variety.
3. **Surface Realization**: Finally, the messages are transformed into grammatically correct sentences.

Natural language generation systems have also been defined in a functional way [8]. Nevertheless, core of all these architectures is the usage of an intermediate structure for storing messages before they are transformed into natural language sentences. The advantage of this procedure is the significant gain in maintainability and flexibility. In a template-based system, each template must be manually modified if a change in the output text is required. In a linguistic-based approach, the output of the generation system can be altered by changing a parameter of the intermediate structure. For instance, the sentence *The weather will be nice today* can be easily transformed into *The weather is nice today* by adapting the tense feature of the main verb in the intermediate representation. Although templates and canned text have been critically discussed, they also have advantages [9]. Therefore, Reiter and Mellish propose a cost benefit analysis [10]. As a result, many natural language generation systems use hybrid approaches where linguistic techniques are combined with canned text and templates [11,12].

### 2.3 Challenges in Generating Text from Process Models

There are a number of challenges for the automatic generation of text from process models. We identified a list of challenges by analyzing the required generation steps and by investigating the respective literature on natural language generation systems. The challenges can be assigned to one of four different categories

**Table 1.** Challenges in Generating Text from Process Models

| Challenge | References |
|---|---|
| 1 Text Planning | |
| 1.1 Linguistic Information Extraction | [13,14] |
| 1.2 Model Linearization | [15,16] |
| 1.3 Text Structuring | [17] |
| 2 Sentence Planning | |
| 2.1 Lexicalization | [18] |
| 2.2 Message Refinement | [19,20] |
| 3 Surface Realization | [21,22] |
| 4 Flexibility | [7,23] |

including text planning, sentence planning, surface realization and flexibility. Table 1 provides an overview of these challenges and according references.

In the *text planning* phase we face three main challenges. First, we have to adequately infer given linguistic information from process model elements. For instance, the activity *Take down order* must be automatically split up into the action *take down* and the business object *order*. Without this separation, it would be unclear which of the two words defines the verb. Label analysis is further complicated by the shortness of process model labels and the ambiguity of the English language [24]. The second challenge is the linearization of the process model to a sequence of sentences. Process models rarely consist of a plain sequence of tasks, but also include concurrent branches and decision points. In addition to these tasks, it must be decided where techniques of text structuring and formatting such as paragraphs and bullet points should be applied.

The *sentence planning* phase entails the tasks of lexicalization and message refinement. The aspect of lexicalization refers to the mapping from BPMN constructs to specific words. It requires the integration of linguistic information extracted from the process model elements and of control structures as splits and joins in such a way that the process is described in an understandable manner. The aspect of message refinement refers to the construction of text. It includes the aggregation of messages, the introduction of referring expressions as pronouns and also the insertion of discourse markers such as *afterwards* and *subsequently*. In order to suitably consolidate sentences, the option of aggregation must first be identified and then decided where it can be applied to improve the text quality. The introduction of referring expressions requires the automatic recognition of entity types. For instance, the role *kitchen* must be referenced with *it* while the role *waiter* must be referenced with *he* or *she*. The insertion of discourse markers should further increase the readability and variability of the text. Hence, varying markers must be inserted at suitable positions.

In the context of the *surface realization*, the actual generation of a grammatically correct sentences is performed. This requires the determination of a suitable word order, the inflection of words, introduction of function words (e.g. articles) and also tasks such as punctuation and capitalization.

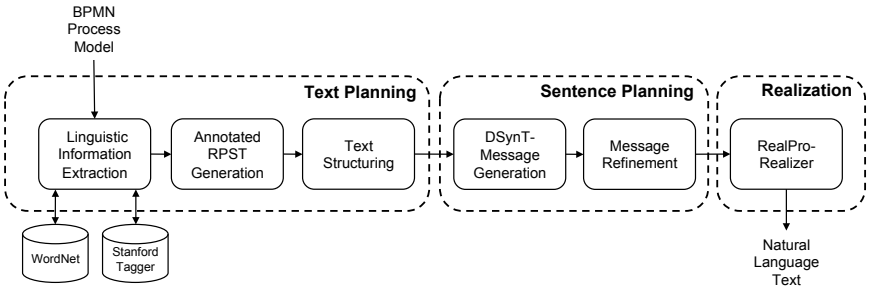



**Fig. 2.** Architecture of our NLG System

Besides the core natural language generation tasks, we consider *flexibility* to be an important feature. As we do not expect the input models to adhere to certain conventions, we have to deal with considerably differing characteristics of the input models. Different scenarios have to be covered, with varying implications for the output text. For instance, if a model uses lanes and thus provides a role description, the sentence can be presented in active voice (e.g. The clerk checks the application). If it is unknown who performs the considered task, the description must be presented in passive voice (The application is checked).

## 3 Text Generation Approach

This section defines our approach to text generation. Figure 2 gives an overview of the six major components building a pipeline architecture. The following subsections will introduce each component in detail.

### 3.1 Linguistic Information Extraction

The goal of this component is the adequate inference of linguistic information from all labeled process model elements. Thereby, it is important that we are able to deal with different types of linguistic representations, so-called label styles. Therefore, we extended prior work which extracted action and business object from activity labels and events [25,13,14]. We included gateways and arc labels in the extraction algorithm to cover all relevant process model elements. The Stanford Parser [26] and the lexical database WordNet [27] are used to recognize different patterns in gateway and arc labels which we identified in the context of a comprehensive analysis of industry process models. Based on the structural insights of process model labels, we can extract action, business object and possible modifying attributes from any process model label, independently from the actual linguistic presentation.

Consider the gateway *Alcoholic Beverages Ordered?* from Figure 1. In this case the application of the Stanford Parser is impeded by the shortness of the label. It returns the tag result *Alcoholic/NNP Beverages/NNP Ordered/NNP ?/.*

indicating that all words are nouns. Hence, it neither recognizes the adjective nor the participle at the end. Being aware of regular labeling structures, we can use Wordnet to determine that the first word is an adjective and that the last word is a verb with the suffix *ed*. As a result we obtain an annotation record for this gateway containing the extracted information. Once this has been done for all labeled process model elements, the annotation records are handed over to the next module.

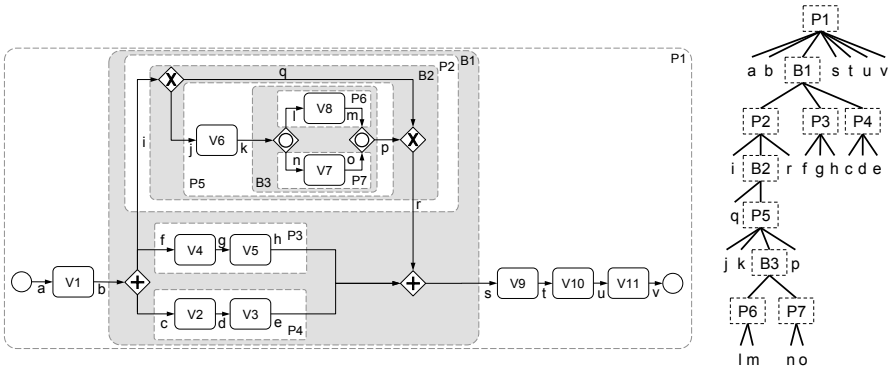
### 3.2 Annotated RPST Generation

The RPST Generation module derives a tree representation of the input model in order to provide a basis for describing the process step by step. In particular, we compute a Refined Process Structure Tree (RPST) which is a parse tree containing a hierarchy of subgraphs derived from the original model [15,16]. The RPST is based on the observation that every workflow graph can be decomposed into a hierarchy of logically independent subgraphs having a single entry and single exit. Such subgraphs with a single entry and a single exit are referred to as fragments. In a RPST any two of these fragments are either nested or disjoint. The resulting hierarchy can be shown as a tree where the root is the entire tree and the leaves are fragments with a single arc.

In total we may encounter four different fragment classes: trivial fragments (T), bonds (B), polygons (P) and rigids (R). Trivial fragments consist of two nodes connected with a single arc. A bond represents a set of fragments sharing two common nodes. In BPMN process models this generally applies for split and join gateways, including more complex split and join structures such as loops. Polygons capture sequences of other fragments. Hence, any sequence in a process model is reflected by an according polygon fragment. If a fragment cannot be assigned to one of the latter classes, it is categorized as a rigid. Although the original version of the RPST was based on graphs having only a single entry and exit point, the technique can be easily extended to compute a RPST for arbitrary process models. Figure 3 illustrates the concepts using an abstracted version of the hotel process and its corresponding RPST.

The RPST generation algorithm by [16] does not order the fragments with respect to the control flow. However, this can be easily accomplished. For each level in the RPST the order can be determined by arranging the fragments according to their appearance in the process model. Hence, the first level starts with the trivial fragment *a*, connecting the start event and vertex *V1*. Accordingly, the trivial fragment *b*, the bond *B1* and the trivial fragments *s*, *t*, *u* and *v* are following. If the order is not defined, for instance in case of parallel branches as within the bond *B1*, an objective criteria as the length of each path can be imposed which is conducive for text generation purposes.

In addition to the introduction of a suitable order, we annotate the RPST with the linguistic information from the extraction phase and with additional meta information. For instance, the vertex *V1* from the trivial fragment *a* is annotated with the action *take down*, the business object *order* and the role *room-service manager*. The bond *B1* is annotated with the action *order*, the




(a) Abstract Version of Example Process (b) RPST

**Fig. 3.** Abstract Version of Figure 1 and its RPST

business object *beverages* and the adjective *alcoholic*. Further, the bond as tagged as an XOR-gateway with *Yes/No*-arcs of the type *skip*. The latter aspect is derived from the fact that one branch is directly flowing into the join gateway and hence provides a possibility to skip the activities on the alternative branch.

### 3.3 Text Structuring

To obtain a manageable and well readable text we use paragraphs and bullet points to structure the messages. We include editable parameters for defining the size of paragraphs. Once such a threshold is reached, the change of the performing role or an intermediate event is used to introduce a new paragraph. Similarly, we make use of bullet points. Every sequence longer than a definable parameter performed by the same role is presented as a bullet list. Further, the branches of any split having more than two outgoing arcs are presented as bullet points. In case of nested splits, the bullet points are indented accordingly in order to enable the reader to easily keep track of nested structures.

### 3.4 DSynt-Message Generation

The message generation component transforms the annotated RPST into a list of intermediate messages. This means that the linguistic information from the model is not directly mapped to the output text, but to a conceptual representation which still allows for diverse modifications. In particular, we store each sentence in a deep-syntactic tree (DSynT), which is a dependency representation introduced in the context of the Meaning Text Theory [28]. In a deep-syntactic tree each node is labeled with a semantically full lexeme, meaning that lexemes such as conjunctions or auxiliary verbs are excluded. Further, each lexeme carries grammatical meta information, so-called grammemes. Grammemes include voice and tense of verbs or number and definiteness of nouns. The advantages of

deep-syntactic trees are the rich but still manageable representation of sentences and the existence of off-the-shelf surface realizers which take deep-syntactic representations as input and directly transform it into a grammatically correct sentence.

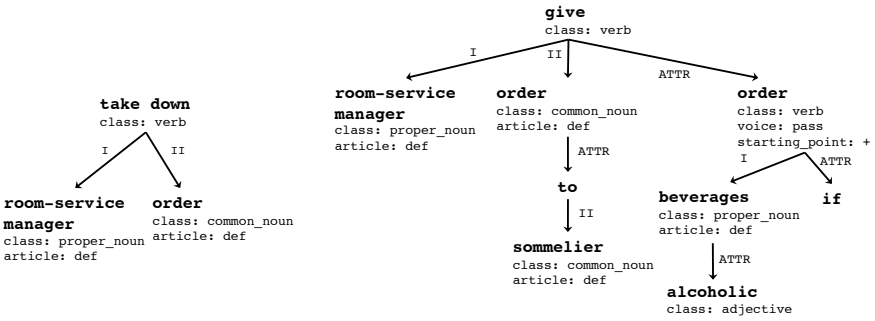
Based on this intermediate representation we developed an algorithm which recursively traverses the RPST and generates DSynT-based messages for trivial fragments and bonds. The following paragraphs introduce the main concepts how this mapping is conducted in our generation system.

As already pointed out, *trivial fragments* always consist of two activities with a simple connection. For the message generation we only consider the source activity, as the target activity is also included in the subsequent fragment as a source. Using the annotation from the RPST the considered activity can be directly mapped to a DSynT. For illustrating this procedure consider the first activity from the example process (Figure 1). Using the action *take down* as the main verb, the role *room-service manager* as subject and the business object *order* as object, we can derive the DSynT depicted in Figure 4(a) representing the sentence *The room-service manager takes down the order*. In case a considered activity is a sub process, we add a footnote to the sentence stating that this step is further explained in an extra model.

The transformation of bonds is more complex. For demonstrating the approach we examine the bond *B1* from our example process. This bond starts with an XOR-gateway labeled with *Alcoholic Beverages Ordered*. Based on the annotation we derive the condition clause *If alcoholic beverages are ordered*. This clause is then passed to the first activity of the *yes*-arc, where the condition and the main clause are combined. As a result, we obtain a DSynT representing the sentence *If alcoholic beverages are ordered the room-service manager gives the order to the sommelier.* The according DSynT is depicted in Figure 4(b). Similarly, we can accomplish the transformation of the join-gateway. The join-condition clause is then passed to the first activity after the bond (*Deliver to Guest's Room*) and incorporated accordingly. The procedure is used to handle bonds of different size and type. In case a process model does not provide any information about the condition of a split, we use predefined DSynT-templates for obtaining a suitable description. Depending on the requirements of the target group, these templates can be easily modified or extended. Within the bond, the recursive transformation algorithm is executed accordingly.

The text generation process builds on the creation of DSynTs with grammemes capturing all required meta information. This includes simple attributes like the word class, but also more sophisticated features as the tense or voice of verbs, the definiteness of nouns or the position of a conditional phrase (see attribute *starting_point* in Figure 4(b)). In order to obtain a high degree of flexibility, we implemented a rule system covering an extensive set of modeling scenarios. As a result, our generation algorithm can automatically decide on these features based on the given circumstances. Accordingly, the absence of a role description automatically leads to a passive sentence. In such a case, the first activity of the

example process would lead to a DSynT representing the sentence *The order is taken down.*




(a) Simple DSynT (b) DSynT with Conditional Sentence

**Fig. 4.** Deep-Syntactic Tree of Two Messages from Figure 1

### 3.5 Message Refinement

Within the message refinement component, we take care of message aggregation, referring expression generation and discourse marker insertion.

The need for *message aggregation* usually arises when the considered process contains long sequences. In such cases we make use of three aggregation techniques: role aggregation, business object aggregation and action aggregation. For instance, if two successive activities are performed by the same role, the messages are merged to a single sentence. Similarly, activities with equal actions or business objects are merged to one sentence, if they appear in two sequential activities.

If there are still adjacent messages with the same role after the aggregation, the role description in the second message is replaced with a *referring expression.* We use WordNet for replacing a role with a suitable personal pronoun. More specifically, we infer all hypernyms of the term associated with the considered role. As a result we obtain a set of more abstract words which semantically include the role description. If we e.g. look up the role *waiter* we will find the hypernym *person* indicating that this role can be replaced with *he* or *she.* By contrast, the set of hypernyms of *kitchen* only contains words like *artifact* or *physical entity* and no link to a human being. Hence, the role *kitchen* is reference with *it.*

For the discourse marker introduction, we identify messages appearing in a strict sequence. Using an extendible set of connectors, we randomly insert a suitable word. In this way, we obtain a well readable text with sufficient variety.

### 3.6 Surface Realization

As already pointed out earlier, the complexity of the surface realization task led to the development of publicly available realizers such as TG/2 [22] or RealPro [21]. Considering aspects as the manageability of the intermediate structure, license costs, generation speed and Java compatibility, we decided to utilize the DSynT-based realizer RealPro from CoGenTex. RealPro requires an XML-based DSynT as input and returns a grammatically correct sentence.

## 4 Evaluation

In this section, we demonstrate the capability of our approach to transform process models to texts. We implemented a prototype and tested it on a set of 53 BPMN models by evaluating the generated texts.

### 4.1 Prototypical Implementation

We used the approach as defined in the previous section for implementing a Java prototype. To this end, we confine the system with regard to two dimensions.

First, we reduced the amount of symbols covered by our system due to the extensiveness of the BPMN 2.0 symbol set. This decision was based on the observation that only a small set of elements is used in practice [29]. As a result, our prototype supports the transformation of the following elements: pools, lanes, activities, standard start and end events, message events, all gateway types, sequence and message flows and sub processes. However, as the capabilities of our technique are not associated with the coverage of individual symbols but with the conceptual approach, our technique can be easily extended.

Secondly, the current implementation supports structured processes and is not yet able to transform rigids. Nevertheless, there are strategies for obtaining structured processes from rigids [30]. These transformation algorithms can be utilized as a preprocessing step.

### 4.2 Evaluation Setup

To assess the capability of our technique we conducted an experiment. The overall goal of the experiment was to answer the following four questions:

1. Does the text contain an adequate sentence for each process model element?
2. Can models be adequately transformed?
3. Are the generated sentences grammatically correct?
4. Are the generated sentences stylistically favourable?

We designed a test collection of 53 BPMN process models, which cover diverse domains and mainly stem from academic training. Table 2 summarizes the characteristics of the test set. We asked three BPM domain experts with on average 4.3 years of process modeling experience to independently assess the 53

generated natural language texts with regard to the introduced questions. Any remark from one of the three evaluators was included in the final assessment. In addition to this qualitative dimension, we included characteristics such as the number of words per sentence to further evaluate the texts from a quantitative perspective.

**Table 2.** Test Set Characteristics

| **Property** | **Min** | **Max** | **Average** | **Total** |
|---|---|---|---|---|
| Number of Activities | 3 | 16 | 8.1 | 428 |
| Number of Events | 2 | 4 | 2.4 | 128 |
| Number of Gateways | 1 | 10 | 3.9 | 204 |

### 4.3 Results

From the evaluation experiment we learned that the first three of our assessment criteria are considered to be fulfilled. Our technique generated grammatically correct texts, which appropriately and completely describe the according process models. As an example consider the following text which was generated by our prototype and represents the process model from Figure 1. It illustrates the handling of labeled and unlabled gateways, nested structures and how referring expressions are introduced in sequences.

*The process begins, when the Room-Service Manager takes down an order. Then, the process is split into 3 parallel branches:*

- *In case alcoholic beverages are ordered, the Room-Service Manager gives the order to the sommelier. Afterwards, one or more of the following paths is executed:*
  - *The Sommelier fetches the wine from the cellar.*
  - *The Sommelier prepares the alcoholic beverages.*
- *The Room-Service Manager assigns the order to the waiter. Subsequently, the Waiter readies the cart.*
- *The Room-Service Manager submits the order ticket to the kitchen. Then, the Kitchen prepares the meal.*

*Once all 3 branches were executed, the Waiter delivers to the guest's room. Afterwards, he returns to the room-service. Finally, the Waiter debits the guest's account.*

One remark of the evaluators was that some models describe the processes in a very general manner. As an example, consider the transformation of the OR-join in the generated text. However, this results from the fact that a process model does not provide further information on the split condition and the performed activities. Accordingly, the lack of information in the model always translates to a lack of information in the text.

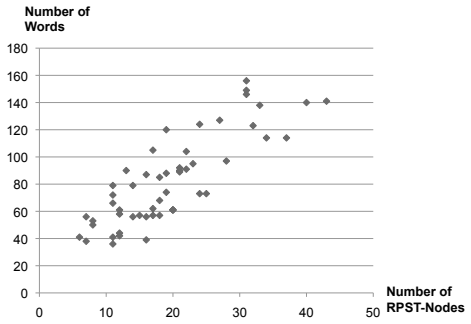



**Fig. 5.** Correlation between Model and Text Size

With respect to the fourth criterion, in total five sentences were considered to be stylistically flawed. In particular, the evaluators criticized sentences where definite articles were unnecessarily inserted. Consider the activity labels *Assign negative points* and *Ship motorcycles to customer*. The first label is transformed into the sentence *The teacher assigns the negative points*. Obviously, the use of the definite article *the* before *negative points* is not appropriate. Although it is not a grammatical mistake, humans would most likely leave out this article. In the second case, the activity label is transformed into *The vendor ships the motorcycles to the customer*. Here the usage of the definite article before *customer* is necessary in order to obtain a naturally appealing text. These examples highlight that the introduction of articles is a context-sensitive task and has to be further investigated in the future. However, due to the small amount of these cases, we still consider our technique to produce text of suitable quality.

The quality of the generated texts is also supported by different quantitative characteristics. As originally pointed out in [31], the sentence length is an important factor for the overall text comprehension. With an average sentence length of 9.4 words, the generated text can be considered to be easily understandable. Further factors for text comprehension are the size of the text and its paragraphs. Figure 5 illustrates the correlation between text and model size, where the latter aspect is represented by the number of RPST nodes resulting from the model. Although some variation is caused be deeper explanations for bonds, the correlation is approximately linear. As a result, also more complex models are converted into texts of manageable size. In addition, the average size of the generated paragraphs of 3.1 sentences indicates a proper organization of the presented text.

## 5 Related Work

Techniques for Natural Language Generation have been around for many years and were applied in many different scenarios. Some examples are the generation of weather forecasts [23] or the creation of reports on computer systems [32].

Research related to text generation for conceptual models can be divided into three categories: text generation systems, usage of natural language for formal specifications and comprehension of text versus models.

There are a few applications of text generation for conceptual modelling. The ModelExplainer generates natural language descriptions of object models [33] and the GeNLangUML system transforms UML class diagrams into text specifications [34]. None of these approaches tackles the specifics of process models. Our approach complements these systems by introducing a flexible technique which is able to handle different aspects of control flow.

Natural language is also used for formally specifying business semantics. For instance, the OMG standard Semantics of Business Vocabulary and Business Rules (SBVR) uses natural language for specifying complex business rules [42]. Our technique contrasts this approach by providing a rather informal complementation to a model while the natural language in SBVR represents a formal specification itself.

The merits of verbalization for model comprehension and validation are discussed from different perspectives. The need for natural language feedback in the validation of conceptual models is addressed in [35,36] by proposing natural language generation as an appropriate solution. This is in line with the approach of ORM and NIAM in terms of verbalization [2,3]. The advantages of text and diagram have been intensively debated in the area of visual programming languages (see [37] for an overview). Today, the consensus is rather that text plus diagram provides better comprehension than any of the two in isolation [38]. The most important reference for this insight is provided by the Cognitive Theory of Multi Media Learning introduced by Mayer [39]. In a series of experiments Mayer demonstrated the value of the combination of text and a graphical representation.

## 6 Conclusion

In this paper, we presented an automatic approach for generating natural language texts from BPMN process models. This approach combines natural language analysis, graph decomposition techniques and a linguistic framework for flexible text generation. The approach has been implemented as a prototype. An evaluation of 53 BPMN process models shows that our technique is capable of reliably generating grammatically and stylistically appropriate texts, which adequately describe the considered process models.

Although the current results are very promising, our technique still requires further empirical tests. Most importantly, we plan to demonstrate that the generated text really helps users to understand process models. We assume that especially non-frequently used model elements such as attached events or complex gateways might be unclear to model readers. Further, we think that particularly people who are not very familiar with BPMN can benefit from an additional text. We aim to address these assumptions in a series of experiments. Moreover, we also plan to consider additional use cases for our technique such as the generation of test scenarios or personalized work instructions.

In addition to these points, the approach should be further generalized. We intend to include the whole BPMN symbol set and to cover text generation for rigids that cannot be structured. This could be done, for instance, by determining preconditions for specific unstructured sequences of activities as proposed in [40].

Beyond that, we plan to extend the current prototype towards a comprehensive technique for process model validation. By consulting the generated texts, also users who are not confident with interpreting process models can decide about the validity of a model. They should then have an interface to edit the generated text. A reverse generation procedure as proposed in [41] could then be used to apply the changes in the model.

## References


1. Rosemann, M.: Potential Pitfalls of Process Modeling: Part A. Business Process Management Journal **12**(2) (2006) 249–254
2. Verheijen, G., van Bekkum, J.: NIAM, an information analysis method. In: IFIP WG8.1 Conf. on Comparative Review of Inf. System Method. (1982) 537–590
3. Nijssen, G., Halpin, T.: Conceptual Schema and Relational Database Design: a fact oriented approach. Prentice-Hall, Inc. (1989)
4. Frederiks, P., Weide, T.: Information modeling: The process and the required competencies of its participants. Data & Knowledge Engineering **58**(1) (2006) 4–20
5. OMG: Business process model and notation (bpmn) version 2.0. (2011)
6. Reiter, E.: Nlg vs. templates. In: In Proceedings of the Fifth European Workshop on Natural Language Generation. (1995) 95–106
7. Reiter, E., Dale, R.: Building applied natural language generation systems. Nat. Lang. Eng. **3** (March 1997) 57–87
8. Cahill, l. et al.: In search of a reference architecture for nlg systems. (1999) 77–85
9. van Deemter, K., Krahmer, E., Theune, M.: Real versus Template-Based Natural Language Generation: A False Opposition? Comput. Linguistics **31**(1) (2005) 15–24
10. Reiter, E., Mellish, C.: Optimizing the costs and benefits of natural language generation. In: IJCAI. (1993) 1164–1171
11. Galley, M., Fosler-Lussier, E., Potamianos, A.: Hybrid natural language generation for spoken dialogue systems (2001)
12. Reiter, E., Mellish, C., Levine, J., Bridge, S.: Automatic generation of on-line documentation in the idas project. In: ANLP. (1992) 64–71
13. Leopold, H., Smirnov, S., Mendling, J.: Recognising Activity Labeling Styles in Business Process Models. EMISA **6**(1) (2011) 16–29
14. Leopold, H., Mendling, J., Reijers, H.: On the Automatic Labeling of Process Models. In: CAiSE 2011. Volume 6741 of LNCS., Springer (2011) 512–520
15. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data & Knowledge Engineering **68**(9) (2009) 793 – 818
16. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: Web Services and Formal Methods. Volume 6551 of LNCS. Springer (2011) 25–41
17. Meteer, M.W.: Expressibility and the Problem of Efficient Text Planning. St. Martin's Press, Inc., New York, NY, USA (1992)
18. Stede, M.: Lexicalization in natural language generation: A survey. Artificial Intelligence Review **8** (1994) 309–336

19. Dalianis, H.: Aggregation in natural language generation. Computational Intelligence **15**(4) (1999) 384–414
20. Kibble, R., Power, R.: An integrated framework for text planning and pronominalisation. In: Natural language generation, ACL (2000) 77–84
21. Lavoie, B., Rambow, O.: A fast and portable realizer for text generation systems. In: Applied natural language processing, ACL (1997) 265–268
22. Busemann, S.: Best-first surface realization. Interface (1996) 10
23. Goldberg, E., Driedger, N., Kittredge, R.: Using natural-language processing to produce weather forecasts. IEEE Expert **9**(2) (1994) 45–53
24. Dixon, R.: Deriving Verbs in English. Language Sciences **30**(1) (2008) 31–52
25. Leopold, H., Smirnov, S., Mendling, J.: Refactoring of Process Model Activity Labels. In: NLDB 2010. Volume 6177 of LNCS., Springer (2010) 268–276
26. Klein, D., Manning, C.D.: Fast Exact Inference with a Factored Model for Natural Language Parsing. In: NIPS 2003. Volume 15., MIT Press (2003)
27. Miller, G.: WordNet: a lexical database for English. Comm. ACM **38**(11) (1995) 39–41
28. Mel'cuk, I., Polguère, A.: A formal lexicon in the meaning-text theory (or how to do lexica with words). Computational Linguistics **13**(3-4) (1987) 261–275
29. zur Muehlen, M., Recker, J.: How much language is enough? theoretical and practical use of the business process modeling notation. In: CAiSE 2008. Volume 5074 of LNCS., Springer (2008) 465–479
30. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Information Systems (2012) to appear
31. Flesch, R.: How to test readability. (1951)
32. L. Iordanskaja, R. Kittredge, A.P.: Lexical selection and paraphrase in a meaning-text generation model. Natural Language Generation in Artificial Intelligence and Computational Linguistics (1991) 293–312
33. B. Lavoie, O. Rambow, E. Reiter.: The modelexplainer. In: Proceedings of the 8th international workshop on natural language generation. (1996) 9–12
34. Meziane, F., Athanasakis, N., Ananiadou, S.: Generating natural language specifications from uml class diagrams. Requirements Engineering **13** (2008) 1–18
35. Dalianis, H.: A method for validating a conceptual model by natural language discourse generation. Advanced Information Systems Engineering (1992) 425–444
36. Rolland, C., Proix, C.: A natural language approach for requirements engineering. In: CAISE 1992. Volume 593 of LNCS. Springer (1992) 257–277
37. Whitley, K.N.: Visual programming languages and the empirical evidence for and against. J. Vis. Lang. Comput. **8**(1) (1997) 109–142
38. Ottensooser, A., Fekete, A., Reijers, H.A., Mendling, J., Menictas, C.: Making sense of business process descriptions: An experimental comparison of graphical and textual notations. Journal of Systems and Software (2012) to appear.
39. Mayer, R.: Multimedia Learning. 2nd Ed. Cambridge Univ. Press (2009)
40. Ouyang, C., Dumas, M., van der Aalst, W., ter Hofstede, A., Mendling, J.: From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. **19**(1) (2009)
41. Friedrich, F., Mendling, J., Puhlmann, F.: Process Model Generation from Natural Language Text. In: CAiSE 2011. Volume 6741 of LNCS., Springer (2011) 482–496
42. OMG: Semantics of Business Vocabulary and Business Rules (SBVR) (2008)