Supporting Process Model Validation through Natural Language Generation

Henrik Leopold, Jan Mendling, and Artem Polyvyanyy

Abstract—The design and development of process-aware information systems is often supported by specifying requirements as business process models. Although this approach is generally accepted as an effective strategy, it remains a fundamental challenge to adequately validate these models given the diverging skill set of domain experts and system analysts. As domain experts often do not feel confident in judging the correctness and completeness of process models that system analysts create, the validation often has to regress to a discourse using natural language. In order to support such a discourse appropriately, so-called verbalization techniques have been defined for different types of conceptual models. However, there is currently no sophisticated technique available that is capable of generating natural-looking text from process models. In this paper, we address this research gap and propose a technique for generating natural language texts from business process models. A comparison with manually created process descriptions demonstrates that the generated texts are superior in terms of completeness, structure, and linguistic complexity. An evaluation with users further demonstrates that the texts are very understandable and effectively allow the reader to infer the process model semantics. Hence, the generated texts represent a useful input for process model validation.

Index Terms—Business Process Model Validation, Natural Language Text Generation, Verbalization

1 Introduction

The adequate transformation of business requirements into a formal system specification is a crucial step in any business-related software development project. Indeed, business process models have proven to be an effective means of specification [1]. However, to avoid problems in later stages, it is of paramount importance to extensively validate the created models already in the beginning of the development project [2].

One of the general challenges in this context is the communication gap between the domain expert and the system analyst [3]. As business professionals often do not feel confident in understanding and interpreting process models, the validation of these models frequently has to rely on a discourse in natural language. For data modeling, verbalization techniques have been defined to provide for a direct mapping from model to natural language [4], [5]. This verbalization concept has been emphasized as an important advantage for the discourse between system analysts and domain experts in the context of requirements engineering [6].

In contrast to data modeling, there are no sophisticated verbalization techniques available for process models. While lists of elements and arcs could be easily transformed into statements like "Activity A is followed by Activity B", this is hardly the way a domain expert would describe and think about a process. The technical challenge here is to create a textual description that captures the full meaning of the process, yet also to organize the text in a way that is intuitive to the reader. The reason why a proper process model verbalization technique is still missing might be a result of the difficulty to meet this challenge. A process model verbalization technique has to serialize the non-sequential structure of a process model into sequential, yet execution-order preserving, text. In addition, it must be capable of analyzing the short and grammatically varying labels of process model elements and of annotating them with their semantic components like action or business object. Furthermore, the verbalization technique needs to handle optionality of certain pieces of information.

In this paper, we follow up on a proposal of adapting the idea of natural language text generation to business process models [7]. To this end, we define a technique that automatically transforms process models into intuitive natural language texts. Our technique can deal with the short and grammatically varying process model element labels as well as with complex process models of arbitrary structure. For demonstrating the capabilities of the proposed technique, we conduct a two-step evaluation. First, we utilize a test collection of model-text pairs from [8] and a set of text metrics to investigate in how far the generated texts are comparable to manually created texts. Second, we ask users to translate the generated texts back into process models. In this way, we investigate whether humans can successfully make sense of the generated texts.

The remainder of this paper is structured as follows.

- *H. Leopold is with the WU Vienna, Austria. E-mail: henrik.leopold@wu.ac.at*
- *J. Mendling is with the WU Vienna, Austria. E-mail: jan.mendling@wu.ac.at*
- *A. Polyvyanyy is with the Queensland University of Technology, Brisbane, Australia. E-mail: artem.polyvyanyy@qut.edu.au*

Section 2 summarizes research on model validation and challenges of text generation from process models. Section 3 provides formal foundations. Section 4 introduces our technique for generating text from process models. Section 5 presents the results of the model-text pair comparison and of the user evaluation. Section 6 discusses the necessary steps for adapting the technique to languages other than English. Finally, Section 7 closes the paper.

2 Background

In this section, we first clarify the role of natural language in process validation to illustrate the need for text generation from process models. Afterwards, we investigate the challenges that are associated with generating text from process models.

2.1 Role of Process Models in Requirements Engineering

The setting of specifying requirements using business process models can be described in terms of a system modeling life cycle including the four stages of elicitation, organization, verification, and validation [9]. First, the system analyst interacts with domain experts to elicit process-relevant information. Traditional approaches for the *elicitation* of requirements include interviews, focus groups, and protocol analysis [10]. More advanced tools use natural language processing for identifying concepts in documents to support the system analyst in an unknown domain [11]. Also tools for automatically inferring requirements from large information sources [12] and for automatically deriving models from natural language documents exist [13]. Second, the system analyst organizes this information by modeling parts of the process. The *organization* of the information in the context of a process model requires the system analyst to adequately formalize the elicited requirements. Various business process modeling languages exist for supporting this task. Also several quality frameworks and modeling guidelines have been defined including the Seven Process Modeling Guidelines (7PMG) [14], the Guidelines of Modeling [15], and the Sequal framework [16]. The modeling itself is supported by many professional modeling tools like ARIS [17] or Signavio[1]. Third, the system analyst conducts a *verification* step on the model with the aim to resolve inconsistencies. This step is largely supported by automated techniques. Petri-net concepts permit the verification of control flow properties such as soundness [18], [19], [20]. Also the correctness of the data flow [21], [22], [23] and the satisfiability of resource constraints [24], [25], [26] can be checked. All these first three steps are rather well covered in related research with viable solutions being available.

1. www.signavio.com

TABLE 1
Overview of Model Validation Approaches

Approach	Author
Prototyping	
Transformational Prototyping	Lindland & Krogstie [29]
Semantic Prototyping	Loucopoulos et al. [30]
Abstraction and Filtering	
Specification Abstraction	de Caso et al. [31]
Hierarchical Goal Organization	Breaux et al. [32]
Scenario-Based RE	Sutcliffe et al. [33]
Scenario-Based Validation	Heymans & Dubois [34]
Specification Visualization	
Visualisation for Validation	Lalioti & Loucopoulos [35]
Requirements Animation	Mashkoor & Matoussi [36]
Property Checking	
Validation against Ontology	Kof et al. [37]
Validation against Plans	Costal et al. [38]
Validation against Questions	Queralt & Teniente [39]
Rule-based Consistency Checking	Egyed [40]
Language Generation	
NIAM	Verheijen & Bekkum [4]
ORM	Nijssen & Halpin [5]
UML Verbalization	Meziane et al. [41]
Object Model Verbalization	Lavoie et al. [42]
Conceptual Model Verbalization	Dalianis[43]
Process Model Verbalization	Malik & Bajwa [44], Coşkunçay [45]

The situation differs for the fourth task of *validation*. Here, the system analyst shows the model to the domain expert, explains the content, and tries to get feedback to check the validity. Such domain experts are typically familiar with the details of the process, but have limited understanding of business process modeling techniques [27]. The system analyst, in contrast, has elaborate methodological skills, but has to rely on the information provided by the domain expert on how the business process works in practice. Ignoring this knowledge gap between domain expert and system analyst has been identified as a major pitfall of process modeling [28]. As domain experts often lack confidence in reading process models, the validation discourse frequently has to rely on natural language. In general, various techniques have been defined for supporting the validation of conceptual models. They can be subdivided into five groups including prototyping, abstraction and filtering, specification visualization, property checking, and natural language generation. Table 1 gives an overview of existing approaches.

Approaches that validate models through *prototyping* use an implementation of the modeled system in order to collect early feedback from domain experts. For instance, Lindland and Krogstie as well as Loucopoulos et al. define methods for systematically transforming requirement models into prototypes [29], [30]. Approaches for *abstraction and filtering* aim at reducing the information content that is provided to the user. For example, de Caso et al. [31] define an approach for automatically obtaining abstractions for validation purposes. The approach from Breaux et al.

[32] allows for using natural language queries to obtain a goal specialization hierarchy. An alternative filtering technique is the generation of scenarios. By generating a set of scenarios from the considered model, domain experts can more easily reflect upon the coverage of their requirements [33]. Building on such scenarios, there are also approaches that *visualize* requirements. This may include the simple visualization of scenarios in terms of a graphical model [35] or even the animation of scenarios [36]. An alternative validation strategy is pursued by approaches that compare models against a *formal property specification*. Such formal specifications may include domain ontologies [37], user questions [39], predefined property sets [38], and consistency rules [40]. The last validation strategy is the *generation of natural language* from conceptual models in order to ease the communication between system analysts and domain experts. The first methods that supported text generation for conceptual models were Object-Role Modeling (ORM) and its predecessor the Natural language Information Analysis Methodology (NIAM) [4], [5]. A key feature of ORM is a direct mapping from data models to natural language text called *verbalization*. Other authors introduced natural language generation for UML class diagrams [41], object models [42], and conceptual models in general [43]. First attempts for verbalizing process models have been undertaken in [44], [45]. However, both works rely on transformation templates and neither properly address control flow structures such as splits, joins, and unstructured process parts, nor do they consider linguistic variations in the process model. As a result, the generated texts are simple and incomplete. All in all, only a few approaches for generating natural language texts from conceptual models are available. In fact, to the best of our knowledge, there is currently no technique for supporting the validation of process models via natural language generation.

However, given a situation where process models are chosen for representing requirements, verbalization would be associated with many benefits. Empirical research has demonstrated that verbalization can significantly improve the level of domain understanding in the context of requirements engineering [6]. Moreover, verbalization does not require much effort. If a generation technique is available, a text can be generated with a single click. So, it is the objective of this paper to close the research gap on natural language generation from process models. In the following, we discuss specific challenges that need to be addressed to achieve this goal.

2.2 Challenges of Text Generation from Process Models

There are a number of challenges for the automatic generation of text from process models. To illustrate these challenges, we use the exemplary process model depicted in Figure 1. It shows a business process from a hotel using the Business Process Model and Notation (BPMN). The process comprises four roles and is, hence, subdivided into four lanes. The process begins with a start event, which is denoted by the circle in the upper left corner. It continues with the activity *Take down order*. The subsequent diamond-shaped symbol containing a cross is referred to as xor-gateway and denotes a decision point, i.e., only one of the outgoing branches can be followed. Hence, the subsequent steps depend on the status and the solvency of the customer. If the customer has the status *premium* or is considered as solvent, the room-service manager approves the order without debit. Otherwise, the guest's account is debited directly. Note that an xor-gateway with multiple incoming arcs waits until one of the incoming branches was completed. The subsequent gateway containing a plus symbol denotes an and-split, i.e., all outgoing branches will be executed. Hence, this gateway indicates that the process is split into two concurrent streams of action: First, the order ticket is submitted to the kitchen and a meal is prepared. Second, beverages are prepared if they were ordered. Note that the gateway containing a circle is denoting an inclusive or-decision, i.e., either one or both outgoing branches can be executed. Once the two concurrent streams of action have been completed, the order is delivered to the customer and the process is finished.

In order to develop a proper understanding of the challenges that are associated with generating text from process models, we investigated the respective literature on natural language generation systems. The challenges can be assigned to one of four different categories including text planning, sentence planning, surface realization, and flexibility. Table 2 provides an overview of the identified challenges (bold font) and the according approaches addressing them (standard font).

The *text planning* phase is associated with three main challenges. First, we have to adequately infer the linguistic information from the 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. As discussed in [46], this is complicated by the shortness of process model labels and the ambiguity of the English language. The second challenge is the linearization of the process model to a sequence of sentences. The problem in this context is that process models rarely consist of a plain sequence of tasks. Typically, they include non-sequential behaviour represented by decision points and concurrent branches. To address this problem, one can use a Refined Process Structure Tree (RPST). The concept of an RPST was introduced by Vanhatalo et al. and Polyvyanyy et al. [47], [48] and facilitates the linearization of a process model. Nevertheless, as

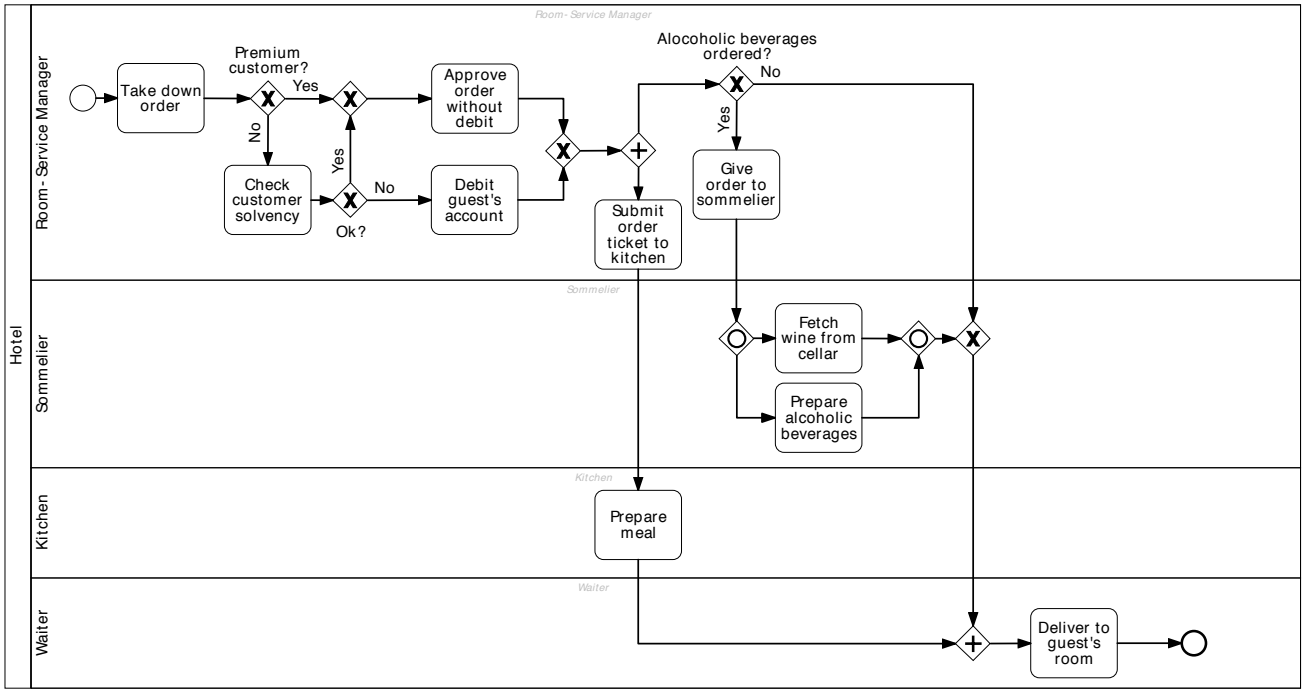


Fig. 1. Exemplary BPMN Process

this is not the primary purpose of the tree, we need to adapt the technique to successfully employ it for generating text. In this context, it should be noted that the verbalization of concurrency is not addressed by existing natural language generation techniques. The third challenge is to decide how techniques of text structuring and formatting such as paragraphs and bullet points should be applied. To adequately address this problem, a lot of research has been conducted in the field of natural language generation (see e.g. [49], [50], [51]). However, these approaches do solely work on the text and do not include information from related resources like process models.

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. Particularly in the context of natural language generation without natural language input, the choice of words represents a considerable challenge [52], [53], [54]. However, for process models this problem does not occur to the same degree. Due to the extensive natural language input, the lexical choice is reduced to the proper integration of the linguistic information from the process model such that the process is described in an understandable manner. The aspect of message refinement refers to the construction of texts. It includes the aggregation of messages, the introduction of referring expressions such 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 it has to be decided where it can be applied to improve the text quality. As these problems represent one of the core tasks of natural language generation, they have been extensively discussed in literature. For instance, Hovy investigates the general role of aggregation in language [55]. Reape and Mellish [56] as well as Dalianis [57] present a variety of concrete aggregation approaches. However, these approaches also abstract from the existence of related resources like process models. 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 decision on where to apply referring expressions is, for instance, discussed in [58]. 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 sentence is performed. This requires the determination of a suitable word order, the inflection of words, the introduction of function words (e.g., articles), and tasks such as punctuation and capitalization. Up until now, many components for surface realization have been introduced [59], [60], [61], [62], [69]. They are based on different theoretical concepts such as the Meaning Text Theory [70], functional grammar [71], or the simple frame language [72], and, hence, ask for different input formats.

Besides the core natural language generation tasks, we consider *flexibility* to be an important feature. As

TABLE 2
Challenges for Generating Text from Process Models

Challenges	Applicable Concepts
Text Planning	
Linguistic Information Extraction	
Linguistic Label Analysis	Leopold et al. [46]
Model Linearization	
Computation of RPST	Vanhatalo et al. [47]
Computation and Generalization of RPST	Polyvyanyy et al. [48]
Text Structuring	
Multi-paragraph Text Segmentation	Heinonen [49]
Efficient Text Planning	Meteer [50], [51]
Sentence Planning	
Lexicalization	
Corpus-based Lexical Choice	Bangalore & Owen [52]
Lexical Choice Criteria	Stede [53], [54]
Message Refinement	
Role of Aggregation in Language	Hovy [55]
Role and Definition of Aggregation	Reape & Mellish [56]
Redundancy Reducing Aggregation	Dalianis [57]
Framework for Pronominalisation	Kibble & Power [58]
Surface Realization	
Fast and Portable Realizer	Lavoie & Rambow [59]
Integrated Surface Realization	Busemann [60]
Plug-in Realization Component	Michael & Robin [61]
Real-time Realization for Dialogues	McRoy et al. [62]
Flexibility	
Varying Input	
Summaries from Multiple Articles	McKeown & Radev [63]
Multiple On-line Sources	Radev & McKeown [64]
Output Adaptation	
Stylistically Varying Texts	Hovy [65]
Multi-lingual Generation	Goldberg et al. [66]
Multi-lingual Generation	Bateman [67]
Language and Graphics	Wahlster et al. [68]

we do not expect the input models to adhere to certain conventions, we have to deal with considerably differing characteristics of the input models. The issue of varying input was also discussed for other natural language generation systems [63], [64]. Although the general problem is similar, these works mainly deal with textual inputs. As natural language only represents one of many dimensions that is subject to variation, we also need to address further aspects. For instance, if a model provides a role description, the sentence can be presented in active voice (e.g. *The room-service manager takes down the order*). If it is unknown who performs the considered task, the description must be presented in passive voice (*The order is taken down*). As a result, template-based approaches, which simply fill the gaps of a predefined sentence, are not appropriate [73]. An example for a flexible generation technique is defined in [65]. It generates stylistically varying texts. Other examples include the generation of multiple languages [66], [67] or the generation of graphics and text [68]. All these approaches have in common that they do not work with text templates, but use linguistic representations to maintain the required level of flexibility. The typical architecture of such approaches has been intensively studied by Reiter [74].

Against the background of these challenges, we propose a technique for generating natural language text from process models. To properly define this technique, the next section introduces formal preliminaries.

3 Formal Preliminaries

This section discusses the formal preliminaries of our work. In particular, we provide a formal description of BPMN process models as they represent the input for the presented generation technique.

The Business Process Model and Notation is a standardized modeling language for graphically specifying business processes [75]. It has been initially developed by the Business Process Management Initiative (BPMI) and is now maintained by the Object Management Group (OMG). BPMN was designed to support both IT and business users, yet providing the means for representing complex real-world semantics [76, p. 13]. As of today, it is available in version 2.0. A comprehensive (yet incomplete) formalization of BPMN has been proposed by Dijkman [77]. Due to the extensive symbol set, a complete formalization of BPMN would introduce unnecessary complexity. Hence, we adapt the formalization from [77], and focus on frequently employed elements. Moreover, we respectively include the aspects of labeling.

Definition 3.1. (BPMN Process Model). A BPMN process model $P_{BPMN} = (A, E, G, F, R, P, L, \rho, \pi, \lambda, \gamma, \epsilon, \tau)$ consists of six finite sets A, E, G, R, P, L, a binary relation $F \subseteq (A \cup E \cup G) \times (A \cup E \cup G)$, a surjective function $\rho : (A \cup E) \rightarrow R$, a surjective function $\pi : R \rightarrow P$, a partial function $\lambda : (A \cup E \cup G \cup F \cup R \cup F) \nrightarrow L$, a function $\gamma : G \rightarrow \{and_S, and_J, or_S, or_J, xor_S^D, xor_S^E, xor_J\}$, a function $\epsilon : E \rightarrow \{plain, timer, message, error\}$, and a function $\tau : E \rightarrow A$ such that

- A is a finite non-empty set of activities.
- E is a finite set of events.
- G is a finite set of gateways.
- We write $N = A \cup E \cup G$ for all *nodes* of the BPMN model.
- F is a set of sequence flows. Each sequence flow $f \in F$ represents a directed edge between two nodes.
- R is a finite set of roles.
- P is a finite set of pools.
- We write $U = N \cup R \cup P \cup F$ for all *units* of the BPMN model which can carry a label.
- L is a finite set of text labels.
- The surjective function ρ specifies the assignment of a role $r \in R$ to activities and events $A \cup E$. Hence, every activity and every event can be associated with exactly one role.

- The surjective function π specifies the assignment of a role $r \in R$ to a pool $p \in P$. Hence, a pool represents a group of roles.
- The partial function λ defines the assignment of a label $l \in L$ to a process model unit $u \in U$.
- The function γ specifies the type of a gateway $g \in G$ as and_S, and_J, or_S, or_J, xor_S^D, xor_S^E, xor_J. The subscript S denotes a gateway that splits the sequence flow into multiple branches, the subscript J denotes a Gateways that joins multiple sequence flows into one, and the superscripts D and E denote whether a split is based on data (D) or the occurrence of an event (E).
- The function ϵ specifies the type of an event $e \in E$ as *plain, timer, message*, or *error*. The type of the event specifies the condition which is associated with it. While a *plain* event is not associated with a specific condition, a *timer* event is connected with a temporal condition, a *message* event is associated with the receipt of a message, and an *error* event is associated with the occurrence of an error.[2]
- The function τ assigns an event $e \in E$ to an activity (attached event). If such an event occurs, the execution of the respective activity is interrupted.

To precisely address the required features of a process model for formally describing our text generation technique, we define the set of predecessors and successors and the set of labeled elements.

Definition 3.2. (Predecessors and Successors of Nodes). Let N be a set of nodes and $F \subseteq N \times N$ a binary relation over N representing the sequence flows. For each node $n \in N$, we define the set of preceding nodes $\bullet n$ with $\{x \in N \mid (x,n) \in F\}$ and the set of successing nodes $n\bullet$ accordingly with $\{x \in N \mid (n,x) \in F\}$.

Definition 3.3. (Labeled Elements). Let U be a set of units, L a set of text labels, and λ a partial function assigning units to labels. Accordingly, the set of labeled units U_λ is given by $dom(\lambda)$.

Building on these definitions, we define a technique for generating natural language texts from process models in the next section.

4 Text Generation Approach

This section defines our approach to natural language generation from business process models[3]. Section 4.1 gives an overview of the architecture and its components. Sections 4.2 through 4.7 introduce each component in detail.

2. Note that this does not reflect the entire event symbol set of BPMN. In this paper, we use the most common event types for illustrating how events can be treated in the context of natural language generation.

3. The source code can be obtained from http://www.henrikleopold.com/downloads

4.1 Overview

The architecture of our text generation technique is building on the traditional pipeline concept from natural language generation systems [74]. The basic rationale of the technique is to utilize the existing information from the model to generate a text. In order to derive a sequence of sentences, we linearize the model via the creation of a tree structure. In particular, the text generation technique comprises six components (see Figure 2):

1) *Linguistic Information Extraction*: Extraction of linguistic components from the process model element labels.
2) *Annotated RPST Generation*: Linearization of process model through the generation of a tree structure. In addition, each node is annotated with the linguistic information from the previous step.
3) *Text Structuring*: Application of text structuring techniques, such as the insertion of paragraphs and bullet points, based on the computed tree structure.
4) *DSynT-Message Generation*: Generation of an intermediate linguistic message structure for each node of the tree. This component represents the core of the generation technique.
5) *Message Refinement*: Refinement of the generated messages through aggregation or the introduction of referring expressions and discourse markers.
6) *RealPro-Realizer*: Transformation of intermediate message structures to grammatically correct sentences.

In the following sections, we introduce and explain each of these components in detail.

4.2 Linguistic Information Extraction

The goal of this component is the adequate inference of linguistic information from all labeled process model elements. To this end, we employ the parsing and annotation techniques defined in [46] to annotate activities, events, and gateways.

For instance, the activity *Take down order* is decomposed into the action *take down* and the business object *order*. Analogously, events and gateways are analyzed and enriched with the according component annotation. For example, the gateway *Alcoholic Beverages Ordered?* is annotated with the action *order* and the business object *alcoholic beverages*. Note that gateways represent an important source of information in the context of text generation. As they define decision points, they are transformed into conditional sentences describing alternative behavior in the process. Once the annotation has been conducted for all labeled process model elements, the annotation records are handed over to the next component.

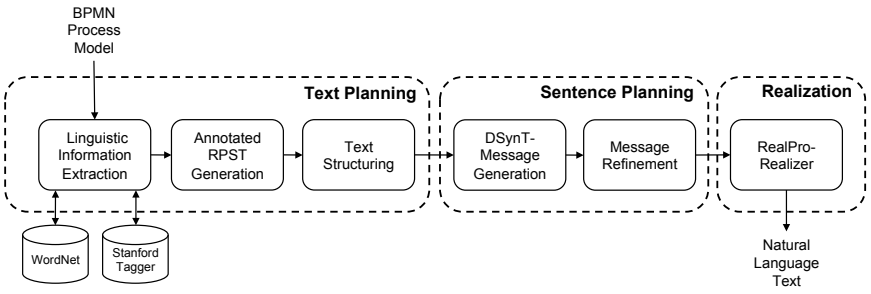


Fig. 2. Architecture of our Natural Language Generation System

4.3 Annotated RPST Generation

The RPST Generation component derives a tree representation for every pool of the input model. In particular, we compute a Refined Process Structure Tree, which is a parse tree containing a hierarchy of sub graphs derived from the original model [47], [48]. The RPST is based on the observation that every workflow graph can be decomposed into a hierarchy of logically independent sub graphs having a single entry and single exit. Such sub graphs are referred to as *fragments*. In an 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 composed of a single arc.

Every RPST fragment belongs to one of four structured classes: trivial fragments (T), bonds (B), polygons (P), or rigids (R). Trivial fragments consist of two nodes connected with a single arc. As an example, consider the connection between the start event and the activity *Take down order* from the example process in Figure 1. A bond represents a set of fragments sharing two common nodes. In BPMN process models, this generally applies to regions between split and join gateways, including more complex split and join structures such as loops. As an example, consider the region of the example process in Figure 1 which is confined by the or-split and the or-join. This fragment containing the two activities *Fetch wine from cellar* and *Prepare alcoholic beverages* represents a bond. Polygons capture sequences of other fragments. Hence, any sequence in a process model is reflected by a respective polygon fragment. As an example, consider the connection between the and-split, the activity *Submit order ticket to kitchen*, and the activity *Prepare meal* from the example process. The individual connections are trivial fragments, while the entire path is a polygon. In case a fragment cannot be classified as trivial, bond, or polygon, it is categorized as a rigid. As an example for a rigid, consider the region of the example process in Figure 1 between the first xor-split (labeled with *Premium customer?*) and the xor-join before the and-split. Due to the combination of multiple decisions and activities that cannot be represented by trivial, bond, or polygon fragments, this model region is classified as a rigid.

Figure 3 illustrates the previously discussed concepts using an abstracted version of the hotel process and its corresponding RPST. To adapt the original RPST generation algorithm to the specific requirements of text generation, we extend it with three additional features: automatic ordering of the fragments, processing of models with multiple entries and exits, and the annotation of the nodes with the extracted linguistic information.

Since the existing RPST algorithms by [47], [48] do not explicitly *define an order* of the fragments, we adapt the RPST computation in a suitable way. For each level in the RPST, we determine the order by arranging the fragments according to their appearance in the process model. Hence, the first level starts with trivial fragment *a1*, connecting the start event and vertex *V1*. Respectively, trivial fragment *a2*, rigid fragment *R1*, etc. are following. If the order is not defined, as for instance in case of parallel branches, an objective criterion such as the path length is employed for determining an order that is conducive for text generation purposes. As rigids define behavior that cannot be represented by sequences of RPST nodes, the ordering is not applied to rigids.

To generate RPSTs from process models *with multiple entries and exits*, we use the algorithm described in [48], [78]. In particular, we augment process models with multiple start events with an additional start node and add an arc from this new node to each of the original start events. Respectively, a model with multiple end events is augmented with an additional end node and an arc from each original end event to the newly introduced end node. At the end of the structuring process, we remove the additionally introduced nodes. As a result, the RPST can also be computed for process models with multiple start and end events.

In addition to these amendments, we also *annotate* the RPST with the *linguistic information* from the extraction phase and with additional meta information. For instance, vertex *V1* from trivial fragment *a1* is annotated with the action *take down*, the business object *order*, and the role *room-service manager*. Bond *B1* is annotated with the action *order*, the business object *beverages*, and the adjective *alcoholic*. Furthermore, the bond is tagged as an xor-gateway with *Yes/No*-arcs

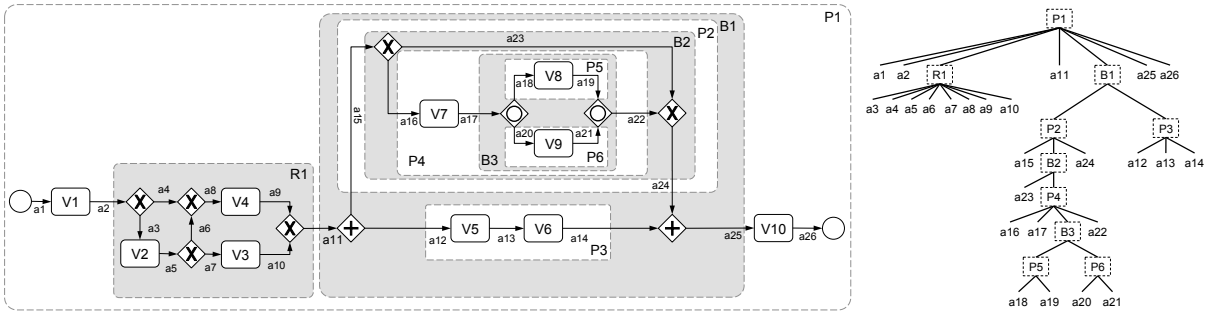


Fig. 3. Abstract Version of Figure 1 and its RPST

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 an option to skip the activities on the alternative branch.

4.4 Text Structuring

The question of how to optimally structure natural texts using paragraphs has been widely discussed in prior research. Many methods employ a similarity metric such as the semantic relatedness between words to compute the lexical cohesion between the sentences of a text [79], [80], [81]. Based on the resulting similarity distribution, a text can be heuristically subdivided into multiple paragraphs. More sophisticated approaches try to use the similarity distribution for identifying the optimal fragment boundaries [49]. However, while the cohesion in standard natural language texts must be completely derived from its semantics, a text generated from a process model can be also structured by building on the features of the model.

Against this background, we introduce two approaches for obtaining a manageable and well readable text. First, we use bullet points to properly communicate the branches of splits. As a result, the text is partitioned into semantically related paragraphs. Moreover, parallel as well as alternative branches are clearly explicated in the text. In case of nested splits, the bullet points are indented respectively. That enables the reader to easily keep track of nested structures. In addition to bullet points, we partition the text using multiple paragraphs. A particular problem in this context is that there is no consensus concerning the optimal number of sentences or words per paragraph. Nevertheless, experiments demonstrated that paragraphs containing more than 100 words are already less understandable than paragraphs with fewer words [82]. Building on this insight, we include an editable parameter for defining the size of a paragraph and predefine this parameter with a value of 75 words. Once this threshold is reached, we use a change of the performing role or an intermediate event as indicator for semantic cohesion and respectively introduce a new paragraph.

4.5 DSynT-Message Generation

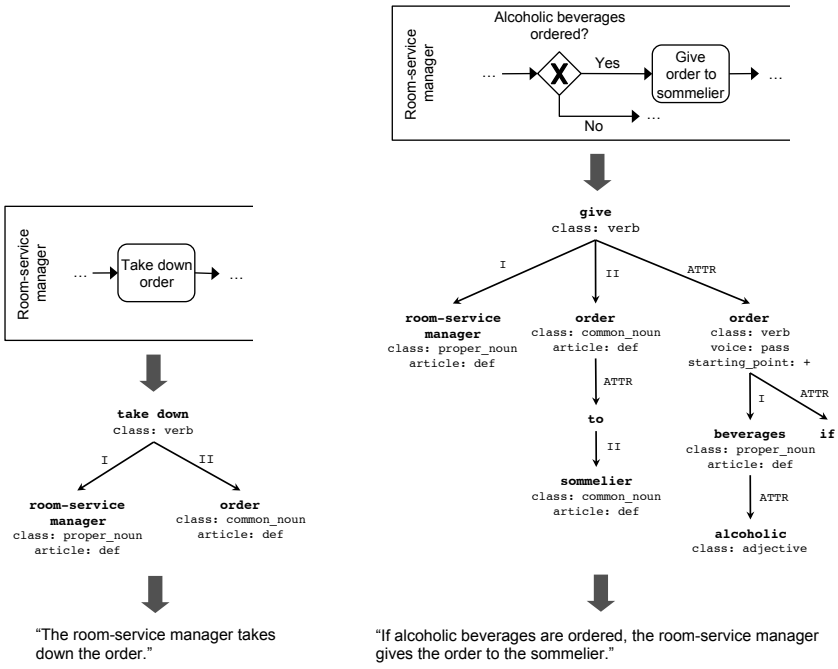
This section introduces the message generation component. It transforms the annotated RPST into a list of intermediate messages. Therefore, it recursively traverses the annotated RPST and derives an intermediate message structure for each RPST node. In the beginning of this section, we introduce the *Deep-Syntactic Tree* as format of the generated messages. Then, we provide a detailed explanation of the transformation of the process model elements. Finally, we discuss how the sub steps are integrated into the overall transformation technique.

4.5.1 Deep-Syntactic Trees

Each message derived from the annotated RPST is stored in a so-called deep-syntactic tree (DSynT). A deep-syntactic tree is a dependency representation that was introduced in the context of the Meaning Text Theory [70], [83]. It is used to represent the most significant aspects of the syntactic structure of a sentence. The advantage of such trees is the rich yet manageable representation. Furthermore, there exist several off-the-shelf surface realizers which directly transform deep-syntactic trees into grammatically correct sentences (see e.g. [59], [60], [61], [62]). Hence, we decided to design an algorithm that maps the given RPST into a list of DSynT-based messages.

In a deep-syntactic tree, each node carries a semantically full lexeme, meaning that words such as conjunctions or auxiliary verbs are excluded. Moreover, each lexeme in a deep-syntactic tree is enriched with grammatical meta information, so-called *grammemes*. Grammemes, for instance, include the voice and tense of verbs or the number and definiteness of nouns. The egdes of the DSynT define the relationship between two nodes. The set of possible relations is rather small. For the purpose of text generation from process models, we employ two actantial[4] relations (denoted with I and II). The first relation specifies the subject and the second relation determines the object of the referenced verb. In addition, we make use of an attributive

4. An actant is a linguistic term for denoting a noun phrase that is functioning as the agent of the main verb.



(a) Simple DSynT (b) DSynT Resulting in Conditional Sentence

Fig. 4. Two Examples for Deep-Syntactic Trees and their Transformation

relation (ATTR) and a relation for conjoining elements (COORD).

Figure 4 illustrates the DSynT concept by showing two example trees, the respective process model fragments as well as the natural language text they are associated with. As stated, Figure 4(a) represents the sentence *The room-service manager takes down the order* and Figure 4(b) captures the conditional sentence *If alcoholic beverages are ordered, the room-service manager gives the order to the sommelier*. The two examples illustrate that the root of a deep-syntactic tree is always formed by the main verb of the sentence and the actantial relations are used to specify subject and object. The ATTR relation is applied in two ways. First, to append adjectives to nouns, and second, to append a conditional sentence to the main sentence. The grammemes of the depicted trees essentially include the *word class* and the *definiteness* of articles. In addition, the lexeme *order* carries the grammeme $starting\_point$ specifying the position of the conditional clause. As a result, the deep-syntactic trees contain all information that is required for constructing proper sentences.

4.5.2 Traversing the Annotated RPST

Starting point for the transformation of the annotated RPST into a set of intermediate messages is the traversing of the tree. Based on the type of the considered RPST node, the respective algorithm for the transformation is triggered. Algorithm 1 formalizes this procedure at the highest abstraction level. The algorithm requires an RPST node as input and returns an ordered list of DSynT messages. Note that the input node may represent the root or any other node of the RPST.

Algorithm 1: transformRPSTNode(RPSTNode n)

```
1: List msgs = new List();
2: Global List passedMsgs = new List();
3: for all RPSTNode child ∈ n.getChildren() do
4:     if child.getNodeType() = TRIVIAL then
5:         if child.getEntry().getElemType() = ACTIVITY then
6:             msgs.add(transformActivity(child.getEntry()));
7:         else if child.getEntry().getElemType() = EVENT then
8:             msgs.add(transformEvent(child.getEntry()));
9:         end if
10:    else if child.getNodeType() = BOND then
11:        msgs.add(transformBond(child));
12:    else if child.getNodeType() = RIGID then
13:        msgs.add(transformRigid(child));
14:    else if child.getNodeType() = POLYGON then
15:        msgs.add(transformRPSTNode(child));
16:    end if
17: end for
18: return msgs;
```

In the beginning, a list for the DSynT messages and a global list for *passed massages* is created (line 1-2). The latter list serves as a stack for messages that need to be incorporated into the text at a later point of time. It is particularly important for the transformation of bonds. In the following loop, each child node of the current RPST node is analyzed (lines 3-17). If the child is a trivial node, it is further checked whether it is representing an activity or an event. As an RPST node always consists of a connection of two vertices, each representing a process model element, this is done by deriving the data from the

entry element of the RPST node. Since the exit element is accordingly included as an entry element in the subsequent RPST node, this procedure avoids that a single process element is considered twice. If the node entry represents an activity, the respective function for transforming activities is triggered (line 6). Otherwise, if the node entry is an event, the event transformation function is executed (line 8). The return value of these functions is a set of DSynT messages which have been created for conveying the semantics of the respective process model element. For the RPST node types *bond* and *rigid*, the algorithm proceeds analogously. If the considered child node represents a bond or a rigid, the respective transformation function is called (lines 11 and 13). Polygon nodes are treated differently. As they represent a sequence of RPST nodes, the algorithm *transformRPSTNode* is recursively triggered for a polygon (line 15). As a result, the comprised nodes are transformed in the underlying iterations. Note that bonds and rigids also contain further RPST nodes. The called transformation functions make equally use of *transformRPSTNode* to convert the subsumed elements into DSynT messages. Finally, after all nodes have been transformed into intermediate messages, the list of messages is returned (line 18).

In the following subsections, we discuss the specific steps for the transformation of trivial fragments, bonds, and rigids.

4.5.3 Transformation of Trivial RPST Nodes

As trivial fragments are always leaves of the RPST, they either represent activities or events. Thus, every activity and every event is transformed into a single sentence. This is accomplished by properly representing the activity or event as a deep-syntactic tree. Due to the annotation, all required information is readily available. In total, four pieces of information must be properly organized in a DSynT: action, business object, additional information, and the role.

Algorithm 2 illustrates the required steps for activities. It requires an RPST node pointing to an activity as input and returns a deep-syntactic tree representing the resulting sentence.

The algorithm starts with deriving the activity from the RPST node (line 1). As previously discussed, an RPST node always consists of a connection of two vertices, each representing a process model element. In order to avoid that an element is considered twice, we always extract the entry vertex from the given RPST node. After the activity object has been obtained, a new DSynT and a DSynT node for the action is created (lines 2-3). The class of the action node is specified with *verb*, and the lemma is determined with the annotated infinitive of the action (lines 4-5). Subsequently, the node is added to the deep-syntactic tree representation (line 6). As the action node represents the root of the tree, the *relation* attribute is not further specified.

Algorithm 2: transformActivity(RPSTNode n)

```
1: Activity a = n.getEntry();
2: DSynT dsynt = new DSynT();
3: DSynTNode actionNode = new DSynTNode();
4: actionNode.setClass('verb');
5: actionNode.setLemma(a.getAnnotation().getAction());
6: dsynt.addNode(actionNode);
7: if a.getAnnotation().getObject().isEmpty() = false then
8:    DSynTNode boNode = new DSynTNode();
9:    boNode.setClass('noun');
10:   boNode.setLemma(a.getAnnotation().getObject());
11:   boNode.setRelationType('II');
12:   actionNode.addNode(boNode);
13: end if
14: if a.getAnnotation().getAdd().isEmpty() = false then
15:   DSynTNode prepNode = new DSynTNode();
16:   prepNode.setClass('preposition');
17:   String preposition = a.getAnnotation().getAddPrep();
18:   prepNode.setLemma(preposition);
19:   prepNode.setRelationType('ATTR');
20:   DSynTNode addNode = new DSynTNode();
21:   addNode.setClass('noun');
22:   String addition = a.getAnnotation().getAdd();
23:   prepNode.setLemma(addition);
24:   addNode.setRelationType('II');
25:   prepNode.addNode(addNode);
26:   actionNode.addNode(prepNode);
27: end if
28: if a.getAnnotation().getRole().isEmpty() = false then
29:   DSynTNode roleNode = new DSynTNode();
30:   roleNode.setClass('noun');
31:   roleNode.setLemma(a.getAnnotation().getRole());
32:   roleNode.setRelationType('I');
33:   verbNode.addNode(roleNode);
34: else
35:   actionNode.setVoice('passive');
36: end if
37: if passedMsgs.getSize() > 0 then
38:   dsynt = mergeSentences(dsynt, passedFragments);
39: end if
40: return dsynt;
```

If the considered activity contains a business object, a respective node is created. Therefore, the class of the node is determined with *noun* and the relation is specified with type *II*. Then, the business object node is appended to the action node as a child (lines 8-12). As a result, the business object plays the grammatical role of an object. Considering the represented sentence, this means that the business object is positioned after the verb. For the incorporation of the additional fragment, the insertion of two nodes is required. The first captures the preposition introducing the addition. The second node contains the additional fragment itself. In case the considered activity includes an addition, a node of the class *preposition* with the relation type *ATTR* is created (lines 14-19). Then, a node for the addition belonging to the class noun and the relation type *II* is introduced. The incorporation of the nodes into the DSynT is realized by appending the addition node to the preposition node and the preposition node to the action node (lines 25-26). Afterwards, the role insertion is handled.

If a role description is available, a respective node is created (line 29). As the role plays the grammatical role of a subject, the relation is specified with type

I (line 32). In case no role description is available, the *voice* attribute of the verb is set to *passive* (line 35). As a result, a missing role description causes a passive sentence, i.e., the activity *Prepare Meal* would be transformed into *The meal is prepared*.

In case there are passed messages from prior transformation steps, such as conditions from splits and joins, these messages are incorporated into the activity (lines 37-39). Finally, the created DSynT object *dsynt* is returned (line 40). Due to complexity, the presented algorithm only discusses the main steps. Exceptional cases and further modifications such as the number of nouns or the specification of articles are not covered. Based on the circumstances, the generation algorithm automatically decides about these features. The details are implemented in a rule system.

Since the transformation steps for events are very similar to the activity transformation, we do not provide a formal description of the event transformation function *transformEvent*. The key difference is that events are enriched with additional meta information. For example, we provide an XML DSynT template for intermediate timer events that communicate that a certain time condition must be met before the process can continue. In a similar way, message, error, and other event types are transformed into natural language text. For attached events, we implemented a special treatment. As they typically lead to an alternative path in the model, they often result in a rigid. In order to communicate attached events in the most intuitive fashion, we create an extra RPST for each attached event and the respective alternative path. By linking this RPST to the source activity, attached events are transformed into text without using the transformation algorithm for rigids.

4.5.4 Transformation of Bonds

We previously defined a bond as a set of RPST fragments sharing two common nodes. In addition, we pointed out that this applies to block-structured splits and joins. In order to adequately transform bonds into natural language, it is necessary to investigate which particular bond types we may encounter in process models.

Table 3 gives an overview of the main bond types in BPMN process models and how they are transformed into text. It illustrates that there are five main process model structures that are captured by bonds. Each of them is conveying semantics that need to be addressed slightly differently in terms of natural language generation. Bonds containing an xor-split (types 1-3) may capture three different scenarios. First, an xor-split can be used for indicating a choice between different activities. Second, it can be employed for providing the possibility to skip one or more activities. Such a construction is characterized by an empty arc from the xor-split to the xor-join. Third, an xor-split might be used to implement a loop. A loop construction is characterized by an arc that is flowing back into the xor-split. Besides the three xor-based structures, bonds may also consist of an and-split and an and-join (type 4), or of an or-split and an or-join (type 5). They consequently represent the opportunity to express concurrent behavior or a choice between one or more options. It is quite intuitive that each of these bond types requires a slightly different textual representation.

The essential idea for transforming bonds into text is to complement the sentences that are generated for activities with additional explanations. Such an explanation sentence may either stand separately or may be incorporated into an activity sentence. Table 3 provides an overview of the basic sentence templates for each bond type. The table distinguishes between labeled and unlabeled gateways and the scope of the sentence. The choice about one of the options depends on three parameters:

- *The Existence of a Gateway Label*: If a gateway carries a label (which is generally only the case for xor and or-splits), we use this label to create a sentence that explains the condition of the split. If a considered gateway is unlabeled, we use a set of predefined sentences to explain the semantics. Note that the predefined sentences are stored as DSynTs in external XML files. Hence, the set can be easily complemented or adapted to the specific needs of the user.
- *The Gateway Type*: As an and-split must be communicated differently than an xor-split, we use differing sentences for each of the previously introduced bond types. If a gateway is labeled, the gateway annotation is accordingly incorporated. Nevertheless, the way a given label is employed is still depending on the bond type. As a result, the generated text for a skip construction varies from the text generated for a general xor-split.
- *The Number of Outgoing Arcs*: As our goal is to communicate the semantics of the model as naturally as possible, we also consider the number of outgoing arcs. Rather simple splits consisting of two arcs do not necessarily require additional meta sentences in the text. For example, an xor-split with two outgoing arcs is transformed into the sentence *If <cond.> then <1. branch>, otherwise <2. branch>*. Hence, we differentiate between splits with exactly two outgoing arcs and splits with more than two outgoing arcs. In the former case, we use an integrated sentence template that incorporates the activities. In the latter case, we employ an isolated sentence such as *If <cond.> then one of the following branches is executed* to explain the model semantics. Nevertheless, if the considered gateway does not carry a label, we abstract from the number of outgoing arcs and employ an isolated sentence to convey the

Algorithm 3: transformBond(RPSTNode n)

```
1: List msgs = new List();
2: Gateway g = n.getEntry();
3: DSynT splitSen = new DSynT();
4: DSynT joinSen = new DSynT();
5: if | g• |> 2 then
6:     if g ∈ G_λ then
7:         splitSen = deriveFromGatewayLabel(g, 'separate');
8:     else
9:         splitSen = loadSplitSentence(g.getType(), 'separate');
10:    end if
11:    joinSen = loadJoinSentence(g.getType(), 'integrated');
12:    if passedMsgs.getSize() > 0 then
13:        splitMsg = mergeSentences(splitMsg,
           passedFragments);
14:    end if
15:    msgs.add(splitSen);
16:    for all RPSTNode child ∈ n.getChildren() do
17:        msgs.add(transformRPSTNode(child));
18:    end for
19:    passedMsgs.add(joinSen);
20: else if | g• |= 2 then
21:    if g ∈ G_λ then
22:        splitSen = deriveFromGatewayLabel(g, 'integrated');
23:    else
24:        splitSen = loadSplitSentence(g.getType(), 'separate');
25:    end if
26:    joinSen = loadJoinSentence(g.getType(), 'integrated');
27:    passedMsgs.add(splitSen);
28:    msgs.add(transformRPSTNode(n.getChildren().get(1)));
29:    passedMsgs.add(joinSen);
30:    msgs.add(transformRPSTNode(n.getChildren().get(2)));
31: end if
32: return msgs;
```

behavior of the model.

To illustrate this procedure, consider the bonds *B2* and *B3* from the example process in Figure 1. Bond *B2* contains an xor-gateway with two outgoing arcs and the label *Alcoholic Beverages Ordered*. Accordingly, we use the annotation of the gateway to derive the sentence fragment *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 respective DSynT is depicted in Figure 4(b). Similarly, we can accomplish the transformation of the join-gateway. The join-condition clause is analogously passed to the first activity after the bond (*Deliver to Guest's Room*) and incorporated into the sentence. As opposed to bond *B2*, the gateway in bond *B3* does not carry a label. Hence, we use the predefined sentence *one or both of the following branches are executed* to describe the or-split although it only has two outgoing arcs. This procedure is used to handle bonds of different size and type. Within the bond, the recursive transformation from Algorithm 1 is executed accordingly.

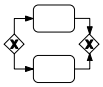
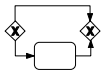
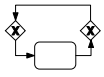
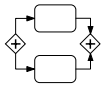
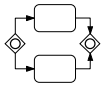
Algorithm 3 formalizes this procedure. In the beginning, a new list for the generated messages is created, and the gateway object is extracted from the RPST node (lines 1-2). Then, two new DSynTs for the split and the join sentences are created. If the considered gateway has more than two outgoing arcs, it is handled by the lines 6-19. In case the gateway is labeled, the sentence explaining the split is extracted from the gateway (line 7). Otherwise, it is loaded from the external XML files (line 9). In either case, a *separate* sentence is constructed as the bond consists of more than two branches. As the split sentence was created as a separate sentence, it can be directly added to the message list. If there exist passed messages that need to be incorporated, the split sentence is first merged with the passed messages (lines 12-14). A scenario where such a situation occurs is the direct succession of a join gateway by a split gateway. Technically, the combination of two or more messages is trivial. Each DSynT of a passed message is added to the main DSynT using the ATTR or COORD relation. In the following loop, the RPST main transformation algorithm is executed for each child fragment of the bond (lines 16-18). Afterwards, the join sentence is added to the global set of *passedFragments* (line 19). Hence, the sentence is incorporated into the next activity of the process model. The handling of gateways with exactly two outgoing branches is implemented by the lines 20-31. The key difference is that a labeled gateway is transformed into an *integrated* split sentence (line 22), and that the join sentence is passed to the first activity of the second branch (line 29). Finally, the message list is returned (line 32).

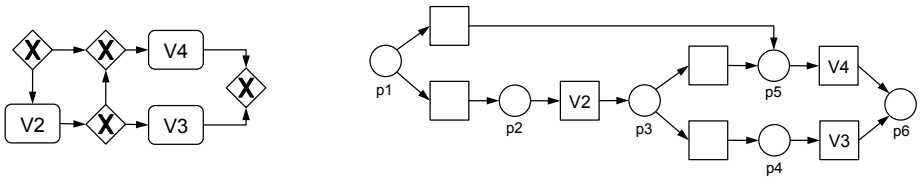
4.5.5 Transformation of Rigids

As discussed earlier, a rigid is a region of a process model that captures arbitrarily structured behavior and hence cannot be characterized by the means of nested bonds, polygons, or trivial fragments. Thus, the previously defined transformation techniques are not sufficient for textualizing the behavior of rigids. In order to properly communicate the behavior of rigids, we explain the different execution options to the reader. More specifically, we discuss one particular execution sequence through the rigid from start to end and then explain the possible deviations from this path. To automatically derive such an execution sequence and its deviations, we transform the rigid into a Petri Net. From this Petri Net we then compute a set of *concurrent runs* covering all activities of the original rigid [84], [85]. Figure 5 shows an abstract version of the rigid from Figure 1 and the corresponding Petri Net. The BPMN to Petri Net transformation can be accomplished using standard transformation algorithms [86], [87], [88]. As we use element identifiers to link transitions and places to the respective BPMN elements, we do not lose any semantics of the BPMN model. After the execution sequences have been computed, we can still associate the places and transitions of the Petri Net with the respective BPMN elements.

Building on the definitions from [84], [85], we designed an algorithm to automatically construct concurrent runs from the derived Petri Net. The rationale behind this approach is to start with the root of the

TABLE 3
Sentence Templates for Transforming Bonds

Bond Type	Appearance	$g \in G_\lambda$	Scope	Sentence Template
Choice		yes	split	*If <cond.> then one of the following branches is executed.*
			join	*Once one of the alternative branches was executed ...*
			integ.	*If <cond.> then <1. branch>, otherwise <2. branch>*
		no	split	*One of the following branches is executed.*
			join	*Once one of the alternative branches was executed ...*
			integ.	-
Skip		yes	split	*If <cond.> then ...*
			join	*In any case ...*
			integ.	-
		no	split	*If required ...*
			join	*In any case ...*
			integ.	-
Loop		yes	split	-
			join	*As long as <cond.> the <role> repeats the latter steps and continues with ... Once <cond.> ...*
			integ.	-
		no	split	-
			join	*If required <role> repeats the latter steps and continues with.... Once the loop is finished ...*
			integ.	-
Parallelism		yes	split	*The process is split into <no.> parallel branches.*
			join	*Once all <no.> branches were executed ...*
			integ.	-
		no	split	*The process is split into <no.> parallel branches.*
			join	*Once all <no.> branches were executed ...*
			integ.	-
Inclusive Choice		yes	split	*If <cond.> then one or more of the following branches is executed*
			join	*Once all desired branches were executed ...*
			integ.	-
		no	split	*One or more of the following branches is executed*
			join	*Once all desired branches were executed ...*
			integ.	-



(a) Rigid (b) Petri Net Representation

Fig. 5. Rigid from Fig. 1 and the Corresponding Petri Net Representation

spanning tree and to first determine the longest path. This path is determined as the main run. The unvisited branches accordingly represent the deviations and are extracted analogously.

Figure 6 shows the result of the run computation. It illustrates that the considered rigid can be covered using three runs. The first run captures the case that the customer has no premium status and is hence debited directly. The second run represents the case of a premium customer. In that case, the order is approved without debit. The third run clarifies that the solvency check may also lead to a positive decision. In this case, the execution continues with place *p5*. These runs demonstrate that the pursued strategy is well-suited for properly describing the possible behavior of a rigid. While the first run shows one possible path through the rigid, the remaining runs describe the possible deviations. In particular, the second run describes an alternative after the beginning of the rigid, and the third run represents an alternative after the execution of activity *v2*.

Using the concept of the concurrent run generation, the verbalization of a rigid can be accomplished with the previously introduced transformation techniques. As every run represents a Petri Net process model, the introduced algorithms can be applied in a straight-

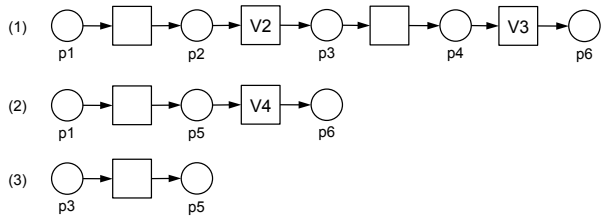


Fig. 6. Runs Computed from the Petri Net Representation

Algorithm 4: transformRigid(RPSTNode $rigid$)

```
1: List msgs = new List();
2: DSynT rigidIntroSentence = loadRigidIntroSentence();
3: DSynT rigidDeviationSentence = loadRigidDevSentence();
4: PetriNet p = transformToPetriNet(rigid);
5: List runs = generateConcurrentRuns(p);
6: PetriNet mainRun = runs.get(1);
7: msgs.add(rigidIntroSentence);
8: msgs.add(transformRPSTNode(mainRun.getAnnoRPST()));
9: msgs.add(rigidDeviationSentence);
10: runs.remove(1);
11: for all PetriNet run ∈ runs do
12:     msgs.add(transformRPSTNode(run.getAnnoRPST()));
13: end for
14: return msgs;
```

forward manner. Algorithm 4 formalizes the required steps. It requires an RPST node representing a rigid as input and returns a list of deep-syntactic trees representing the resulting text.

Algorithm 4 starts by defining a list and two meta sentences for describing the behavior of the rigid (lines 1-3). The *rigidIntroSentence* is used for communicating that a rigid captures several execution sequences and how the following text describes its behavior. The *rigidDeviationSentence* is inserted for introducing the list of possible deviations. After defining the required variables, a Petri Net is derived from the given rigid (line 4). Then, a list of concurrent runs is computed from the Petri Net. In order to present the main run separately, the first run is extracted and transformed into text (lines 6-8). As Petri Nets can be easily mapped to BPMN process models, the textualization of Petri Nets can be accomplished without further adaptations of the text generation algorithm. After transforming the main run, the remaining runs are paraphrased accordingly (lines 11-13). Once all runs have been transformed, the list of messages is returned (line 14). Note that Algorithm 4, for the purpose of maintaining a well-readable presentation, does not cover the case of rigid runs. If a run represents a rigid itself, we employ the concept of behavioral profiles [89] for pair-wise explaining the order of tasks. Given two activities A and B that are part of a rigid run, behavioral profiles enable us to infer whether A and B are exclusive to each other, whether A and B must be executed in parallel, or whether there is a particular order between A and B. Depending on the result, we explain the relationship between the task pair to the reader. Although the technical implementation of this step is trivial, it ensures that our approach can handle rigids of arbitrary complexity.

4.6 Message Refinement

Within the message refinement component, we take care of message aggregation, referring expression generation, and discourse marker insertion. At this stage, the entire RPST has been transformed to a list of DSynT-based sentences. The resulting list of *messages* serves as input for the refinement component.

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*: If two successive activities are performed by the same role, the messages are merged to a single sentence. Instead of generating the two sentences *The waiter serves the customer* and *The waiter issues the invoice*, we merge the sentences to *The waiter serves the customer and issues the invoice*.
- *Business Object Aggregation*: Neighbouring activities sharing a common business object are aggregated in a similar fashion. For example, the two sentences *The bill is created* and *The bill is sent* are aggregated to *The bill is created and sent*.
- *Action Aggregation*: Analogously to activities sharing a common business object, we use identical actions two merge sentences. For instance, the sentences *The manager is informed* and the *The customer is informed* are consolidated to *The manager and the customer are informed*.

Note that aggregation may also include more than two activities. The minimum and maximum number of aggregations can be flexibly configured. The overall goal is to generate text that is as natural as possible.

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 word associated with the considered role. As a result, we obtain a set of more abstract words which semantically include the role description. If we, for instance, look up the role *waiter*, we can identify the hypernym *person* indicating that this role should 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 referenced with *it*.

For the *discourse marker introduction*, we identify messages appearing in a strict sequence. Using an extendible set of connectors such as *then*, *afterwards*, and *subsequently*, we randomly insert a suitable word. In this way, we obtain a well readable text with sufficient variety. As the technical implementation of

the discourse marker introduction is trivial, we do not provide a detailed algorithm.

4.7 Surface Realization

As already pointed out in Section 2.2, the complexity of the surface realization task led to the development of publicly available realizers such as TG/2 [60] or RealPro [59]. 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 a deep-syntactic tree as input and returns a grammatically correct sentence.

Building on the RealPro realizer, the technical implementation of the realization task is trivial. In a loop, each DSynT is passed to the realizer. The resulting natural language text is then added to the final output text. After all DSyntT-based messages have been transformed, the final text is presented to the user.

5 Evaluation

To demonstrate the capability of the proposed technique for generating natural language texts from process models, we conduct a two-step evaluation. First, in Section 5.1, we apply our technique to real-world process models and investigate how the generated texts compare to textual descriptions created by humans. Second, in Section 5.2, we study in how far humans are capable of making sense of the generated texts. To this end, we ask humans to transform the generated texts back into process models.

5.1 Technical Evaluation

The overall goal of the technical evaluation is to compare the generated texts with textual descriptions created by humans. Our hypothesis is that both generated and manually created texts result in text metrics being in a comparable range. Section 5.1.1 presents the general setup of the technical evaluation and the employed metrics. Section 5.1.2 introduces the test collection we utilize. Section 5.1.3 investigates the technique from a runtime performance perspective. Section 5.1.4 presents the results from the text generation and comparison. Section 5.1.5 provides a detailed discussion of the comparison.

5.1.1 Setup

In the context of the technical evaluation, we consider two dimensions for comparing generated and manually created texts: text structure and text content.

The *text structure* dimension is concerned with syntactic characteristics of the texts indicating their complexity. Since syntactic complexity imposes a higher cognitive load on the reader, it is often considered as an important factor decreasing the understandability of texts [90], [91]. Nevertheless, up until now there is no consensus regarding metrics that are best suited for assessing the syntactic complexity [92]. The first approaches that tried to automatically assess the quality and complexity of texts date back to the sixties to the works of Page [93], [94]. He employed simple text features such as word count or word length to evaluate text quality. Today, more sophisticated techniques are available, taking into account that humans typically have a more holistic view on a text [95], [96], [97]. For the purpose of this evaluation, we adapt the sentence complexity metrics proposed by Lu [92]. They include different characteristics for evaluating the syntactic complexity of sentences, and are, hence, well-suited for comparing the complexity of texts. In order to also cover the relationship between text and model structure, we further consider a metric capturing the number of sentences per process model node. Altogether, we employ the following metrics:

- *Average Number of Sentences (S)*: Average number of sentences per text.
- *Words per Sentence Ratio (W/S)*: Average number of words per sentence.
- *Clauses per Sentence Ratio (C/S)*: Average number of clauses per sentence.
- *T-Units per Sentence Ratio (T/S)*: Average number of t-units per sentence. A t-unit is a main clause that contains an attached or embedded subordinate clause or any non-clausal structure [98].
- *Complex T-Units per Sentence Ratio (CT/S)*: Average number of complex t-units per sentence. A t-unit is categorized as complex if it contains a dependent clause [99].
- *Average Sentences per Node Ratio (S/N)*: Average number of sentences used for describing a node of the process model.

The *text content* dimension refers to the extent the text reflects the semantics of the model. In general, we identified that the sentences of the generated and the manually created texts can be subdivided into three types: sentences describing the model content, i.e., the nodes of the model, sentences solely discussing the control flow, and sentences providing additional context information that is not captured by the model. Hence, we operationalize the text content dimension using the following metrics:

- *Activity Coverage (AC)*: Share of activities that are discussed in the text.
- *Event Coverage (EC)*: Share of events that are discussed in the text.
- *Gateway Coverage (GC)*: Share of gateways that are discussed in the text.
- *Sequence Flow Coverage (FC)*: Share of sequence flows that are discussed in the text.
- *Content Sentences (CS)*: Share of sentences explaining the model content, i.e., the semantics of the model nodes.
- *Meta Sentences (MS)*: Share of sentences that only

discuss the control flow of the model.
- *Information Sentences (IS)*: Share of sentences that provide additional context information that is not captured by the model.

5.1.2 Test Collection Demographics

For the evaluation, we employ the BPMN process model collection from [8]. The comprised models vary with regard to several dimensions such as model source, size, complexity, and the employed element set. Hence, it is well-suited for achieving a high external validity of the results. After removing one model from the set since it did not fulfil the required soundness criteria, we yielded a test set of 46 process models. As each model is also complemented with a manually created natural language text, the collection contains all the necessary material for comparing generated and manually created texts. Table 4 summarizes the main characteristics of the test collection aggregated by the comprised sources. In total, the test collection consists of models from ten different sources:

1) *Humboldt University of Berlin (HUB)*: The models from the HU Berlin represent BPMN exercises which are used in BPMN tutorials. The models and the texts were translated from German to English.
2) *Technical University of Berlin (TUB)*: The models from the TU Berlin were created in the context of a research project and are discussed in [100], [101].
3) *Queensland University of Technology (QUT)*: Similar to the models from the HU Berlin, the models from the QUT represent BPMN exercises with according solutions.
4) *Eindhoven University of Technology (TUE)*: The BPMN model from the TU Eindhoven was also created in the context of a research project. The details are discussed in [102].
5) *Vendor Tutorials (VT)*: The vendor tutorial models stem from the websites or online help documentations of the BPM tool vendors *Active VOS* and *BizAgi*.
6) *inubit AG (IAG)*: The models from the inubit AG represent modeling tutorials that are used in the context of client and employee trainings. All included texts and models were translated from German to English.
7) *BPM Practitioners (BPMP)*: This model represents an exercise that was provided by a BPM consultant. It is used in BPMN modeling tutorials.
8) *BPMN Practice Handbook (PHB)*: These models represent exercises from the BPMN Practice Handbook [103]. Both models and texts were translated from German to English.
9) *BPMN M&R Guide (MRG)*: Similar to the models from the BPMN Practice Handbook, these models represent exercises from the BPMN M&R Guide [104].
10) *FNA - Metrology Processes (FNA)*: This sample includes models from the Federal Network Agency of Germany. The models were initially provided as UML Sequence diagrams. Hence, they were transformed into BPMN as documented in [8]. In addition, texts and models were translated from German to English.

The data from Table 4 illustrates that the models from the different sources vary in many regards. While the models stemming from exercises and tutorials are rather small, some models from the research projects contain more than 50 nodes. Particularly, the number of gateways and arcs emphasizes that the majority of the models are not simple sequences of tasks, but frequently contain splits and joins. Furthermore, the number of pools and lanes highlight that the models also include different degrees of interaction. The models from the TU Eindhoven and the inubit AG include, on average, four or more lanes. The models from the TU Berlin and the FNA frequently make use of multiple pools. Concerning the number of BPMN symbols, we observe differences between 4 and 12 distinct BPMN symbols per model from a single source. In total, the models cover 22 different BPMN symbols including various event and gateway types, attached events, and subprocesses. Against this background, we consider the test sample to be suitable for illustrating the capability of the technique to successfully generate natural language texts.

5.1.3 Performance Results

The main application scenario for the presented text generation technique is to provide domain experts with a complementary text. Hence, the generation is not necessarily time critical. However, if the generation is included in a modeling tool, for instance to provide an alternative view on the model, the computation must be adequately fast. We tested the text generation on a MacBook Pro with a 2.26 GHz Intel Core Duo processor and 4 GB RAM, running on Mac OS X 10.6.8 and Java Virtual Machine 1.5. To exclude distortions due to one-off setup times, we ran the generation twice and considered the second run only.

Table 5 summarizes the average, minimum, and maximum execution times of the text generation technique. The numbers show that an average generation run consumes 5.66 seconds. Large deviations from this value can be only observed for extremely large or extremely small models. Thus, the longest generation run was measured for the largest model of the collection containing over 50 nodes and 4 pools. Considering the details, it becomes apparent that especially the number of pools increases the generation time. This can be explained by the fact that a model with multiple pools is split up into several individual models and that the generation is triggered for each pool separately. Nevertheless, the models from the TU Berlin illustrate that also large models are converted

TABLE 4
Overview of Test Data Set Characteristics by Source

ID	Source	Type	M	N	A	E	G	F	P	L	NS
1	HUB	Academic	4	20.8	9.0	5.3	6.5	22.8	1.3	2.5	10
2	TUB	Academic	2	54.5	22.5	21.5	10.5	55.5	3.5	3.5	11
3	QUT	Academic	8	10.6	6.1	2.5	2.0	10.5	1.0	1.6	8
4	TUE	Academic	1	30.0	18.0	4.0	8.0	33.0	1.0	5.0	7
5	VT	Industry	3	10.0	5.3	3.3	1.3	9.7	1.0	2.3	8
6	IAG	Industry	4	17.8	9.0	5.0	3.8	18.3	1.3	4.0	9
7	BPMP	Industry	1	8.0	4.0	3.0	1.0	8.0	1.0	2.0	4
8	PHB	Text Book	3	10.3	5.0	3.3	2.0	10.0	1.0	1.7	6
9	MRG	Text Book	6	18.3	7.0	8.2	3.2	18.2	1.3	1.8	12
10	FNA	Public Sector	14	20.1	8.0	9.0	3.1	18.9	2.3	2.3	8
	Total		**46**	**18.2**	**8.1**	**6.7**	**3.5**	**18.1**	**1.6**	**2.3**	**22**

Legend: M = Total Number of Models per Source, N = Nodes per Model, A = Activities per Model, E = Events per Model, G = Gateways per Model, F = Sequence Flows per Model, P = Pools per Model, L = Lanes per Model, NS = Total Number of Different BPMN Symbols per Source

TABLE 5
Average Generation Time for each Model by Source

ID	Source	Avg (s)	Min (s)	Max (s)
1	HUB	4.91	4.02	5.94
2	TUB	9.45	8.38	10.53
3	QUT	5.01	3.85	8.41
4	TUE	4.77	4.77	4.77
5	VT	5.07	3.78	6.32
6	IAG	4.99	4.07	6.93
7	BPMP	4.19	4.19	4.19
8	PHB	4.97	3.88	6.91
9	MRG	5.33	4.87	6.26
10	FNA	6.47	3.84	8.02
	Total	**5.66**	**3.78**	**10.53**

into text between 8 and 10 seconds. As the generation technique is not required to instantly present a result to the user, we consider this as reasonable performance.

5.1.4 Text Generation Results

From the evaluation experiment, we learned that the presented technique is capable of generating grammatically correct texts which appropriately describe the respective process models. As an example, consider the following text, which was generated by our technique and represents the process model from Figure 1. It illustrates the handling of labeled and unlabled gateways, nested structures, and rigids.

The process begins when the Room-Service Manager takes down an order. Subsequently, the process contains a region which allows for different execution paths. One option from start to end is the following:

- *The Room-Service Manager checks the customer solvency. If it is not ok, the Room-Service Manager debits the guest's account.*

However, the region also allows for a number deviations:

- *In case of a premium customer the Room-Service Manager approves the order without debit.*
- *After checking the customer solvency, the Room-Service Manager may also approve the order without debit.*

Then, the process is split into 2 parallel branches:

- *If it is necessary, the Room-Service Manager gives the order to the sommelier. In case alcoholic beverages were ordered, one or both of the following activities are executed:*
  - *The Sommelier prepares the alcoholic beverages.*
  - *The Sommelier fetches the wine from the cellar.*
- *The Room-Service Manager submits the order ticket to the kitchen. Then, the Kitchen prepares the meal.*

Once both branches were executed, the Waiter delivers to the guest's room. Afterwards, the process is finished.

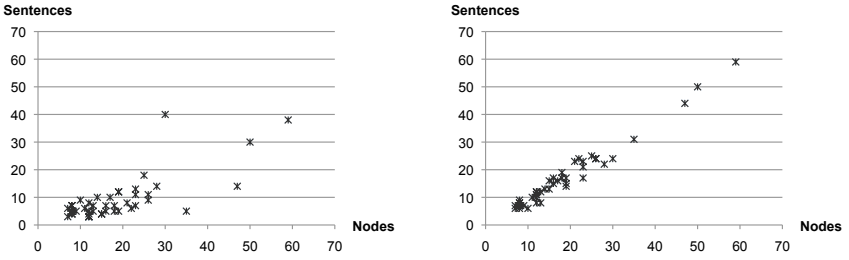
Table 6 summarizes the overall evaluation results for the structural dimension. A general observation is that the generated sentences are shorter than the manually created sentences. While the sentences from the original texts contain an average of 15.5 words, the generated sentences only include an average of 8.3 words. However, the shortness does not imply that the generated texts use less words to communicate the content. In fact, the shortness in terms of word count is compensated by a higher number of sentences. Still, the complexity metrics of clauses per sentence ratio (C/S), t-units per sentence ratio (T/S), and complex t-units per sentence ration (CT/S) indicate that the generated sentences are less complex with regard to the syntactic dimension. Particularly the number of clauses per complex t-units is, on average, much smaller for the generated sentences.

Considering the individual values from the employed sources reveals that these general observations do not equally apply to all collections. As the original texts were created by humans, they are subject to a certain degree of variation. For instance, the original text for the TUE model contains 40 sentences while

TABLE 6
Comparison of the Structural Dimension of Original and Generated Texts

		Original Texts						Generated Texts					
ID	**Source**	**S**	**W/S**	**C/S**	**T/S**	**CT/S**	**S/N**	**S**	**W/S**	**C/S**	**T/S**	**CT/S**	**S/N**
1	HUB	10.3	15.6	1.5	1.0	0.5	0.5	17.5	8.5	1.3	1.0	0.3	0.8
2	TUB	34.0	20.0	1.8	1.0	0.6	0.6	42.2	8.9	1.2	0.9	0.1	1.0
3	QUT	7.1	16.2	1.5	1.1	0.5	0.6	11.0	7.5	1.2	1.0	0.2	0.8
4	TUE	40.0	16.7	1.3	0.9	0.4	1.3	24.0	8.9	1.3	1.0	0.3	0.8
5	VT	7.0	16.6	2.0	0.8	0.7	0.7	9.0	8.7	1.1	1.0	0.1	0.9
6	IAG	11.5	16.3	1.5	1.2	0.6	0.7	15.5	8.5	1.4	1.0	0.3	0.9
7	BPMP	7.0	8.1	1.1	1.0	0.1	0.9	8.0	10.0	1.3	0.9	0.2	1.0
8	PHB	4.7	15.1	1.6	1.6	0.4	0.5	8.0	8.4	1.4	1.0	0.4	0.8
9	MRG	7.0	19.3	1.8	1.0	0.5	0.4	18.3	9.7	1.4	1.0	0.3	1.0
10	FNA	6.4	12.8	1.2	0.9	0.3	0.3	19.2	7.7	1.1	0.9	0.1	1.0
	Total	**9.0**	**15.5**	**1.5**	**1.0**	**0.5**	**0.5**	**16.8**	**8.3**	**1.2**	**0.9**	**0.2**	**0.9**

Legend: S = Sentence, W/S = Words per Sentence, C/S = Clauses per Sentence, T/S = T-Units per Sentence, CT/S = Complex T-Units per Sentence, S/N = Sentence per Node



(a) Original Texts (b) Generated Texts

Fig. 7. Node-Sentence Comparison of Original and Generated Texts

the generated text only consists of 24. We also observe significant deviations for the complexity of the manually created texts. As an example, consider the complex t-units per sentence ratio for the BPMP model (0.1) and the Vendor Tutorial models (0.7). Likewise, the sentence to node ratio illustrates great differences. Among the set of original texts, we observe differences between 0.3 and 1.3 sentences per node. Among the generated texts, we face a rather stable value of 0.9. Figure 7 further illustrates the relationship between the number of process model nodes and the number of sentences. For the original texts, it shows a rather weak linear relationship with some obvious exceptions. By contrast, it shows a clear linear relationship for the set of generated texts. This emphasizes that the results of the text generation are much more stable than the results produced by humans.

Table 7 shows the results for the content dimension. The data illustrates that the generated texts cover all aspects, i.e. activities, events, gateways, and sequence flows that are included in the models. Considering the design of the generation algorithm, this is actually a predictable result. As we traverse the RPST and convert the model node by node, it is technically not possible to miss an aspect of the model. Although the coverage values of the manually created texts are also very close to 100%, in total three activities and twelve events are not discussed in the investigated texts. While these cases do not significantly affect the ability of the reader to understand the models, it reflects the general risk that humans may miss activities when describing a model. A further observation in this context is that the manually created texts occasionally use other words to describe activities. For example, instead of using the word *message* as provided in the model activity, an original text used the word *notification*. While this can have the positive effect of introducing more variety, it may also confuse model readers as they need to identify the correspondence between sentence and activity.

Considering the type distribution of the sentences, we observe more substantial differences. First, the generated texts do not contain information sentences providing additional context information. By contrast, the manually created texts include an average of 8% information sentences. Typical examples for such information sentence are *A small company manufactures customized bicycles* or *The Evanstonian is an upscale independent hotel*. They help to understand the context of the model, but are not necessary for understanding the model semantics. Similar to the activity coverage, this is not surprising. Since the generation algorithm solely builds on the information from the model, all generated sentences are either content or meta

sentences. The second difference with regard to the sentence types is the share of meta sentences. While the generated texts often explicitly discuss control flow aspects such as splits, joins, and rigids, the manually created sentences do not discuss these aspects in isolation. They rather use short fragments like *in parallel* or conditional sentences to explain the control flow on a higher abstraction level. Although explicit discussions of such control flow aspects may reduce the impression of generated texts being natural, it may help readers to fully understand the model semantics.

5.1.5 Discussion of Results

The evaluation shows several similarities between the generated and the manually created texts. However, there are three types of differences: the stability of the structural text characteristics, the semantic coverage, and the cost for creation and adaptation.

The evaluation demonstrated that the generated texts are very *stable* with respect to structural characteristics. In particular, we learned that the generated sentences are comparably short and simple. Nevertheless, as this is the direct result of the parametrization of the algorithm, the complexity of the output can be adapted to the specific requirements of the users. If the readers perceive the generated sentences as too short, the configuration of the aggregation component can be adjusted accordingly. In case the readers face understandability problems due to long sentences, the aggregation can be turned off. Analogously, the segmentation of the text using bullet points or paragraphs can be adapted. In general, it is important to note that the structural characteristics of the generated text are configurable and, hence, not subject to notable variations. In comparison to manually created texts, this represents an important advantage. As illustrated by the varying complexity among the texts of the test sample, the complexity of manually created texts is hard to predict. To guarantee for a stable level of complexity among manually created texts, it would be necessary to sufficiently train the text writers.

With respect to the *semantic coverage*, the evaluation illustrates that the generated texts reliably cover all activities of the model. As the purpose of textual descriptions is to increase the model understanding, this is an important feature. The evaluation showed that the manual creation of texts might be associated with a coverage below 100%. This may either originate from unintentionally leaving out an activity due to the complexity of the model or from the conscious decision of a model writer to skip a certain step. Abstracting from the particular reason, we may conclude that the manual creation of a text is always subject to incompleteness in terms of semantic coverage. Due to the design of the algorithm, the generated texts do not suffer from this problem. Besides the activity coverage, we also learned that the generated sentences are semantically much closer to the model. This applies to content sentences as well as to meta sentences describing the control flow. As the presented technique builds on the information from the model, additional context information is only provided by manually created texts. Nevertheless, as context sentences are much more general than sentences describing the model, they are also not subject to frequent changes. Hence, we consider the possibility of including humans for complementing context sentences as a reasonable approach. A manually complemented sentence could easily be recognized by the technique and accordingly included in the text in a future generation run.

Concerning the *costs of creating or adapting* textual descriptions, it is apparent that the automatic generation is not associated with additional costs. Generated texts can be created with a single click. In contrast, a manual creation of a textual description is, depending on the size of the model, associated with considerable effort. The definition of requirements often involves the creation and validation of various process models. In a more general setting, large corporations maintain collections with up to thousands of process models [28], the manual creation does not appear to be a reasonable solution. It is also important to note that process models are typically subject to changes. Hence, the describing texts must be adapted accordingly. While the generated texts can be updated by repeating the generation run, the manual adaptation might be a cumbersome task. Especially, if the changes in the model are not well documented, the user must first identify the delta between model and text in order to subsequently update the text.

In conclusion, we can state that the introduced technique generates texts that are, in many regards, close to those created by humans. The generated texts are, however, more stable than the manually created ones. Important characteristics such as sentence complexity, text size, and also the full coverage of all process model nodes in the text are a direct result of the algorithm design and not subject to variation. While the manual creation of texts might be associated with considerable effort, the automatic generation is accomplished with a single click in a reasonable time. In view of these facts, the presented technique appears to be well suited for supporting process model validation with generated natural language texts.

5.2 User Evaluation

The goal of the user evaluation is to demonstrate that humans can successfully make sense of the generated texts. Section 5.2.1 introduces the general setup of the evaluation experiment. Then, Section 5.2.2 introduces the test data set and the participants of the experiment. Finally, Section 5.2.3 presents the results.

5.2.1 Setup

To demonstrate that the generated texts can be properly understood by humans, we employ an instrument from

TABLE 7
Comparison of the Content Dimension of Original and Generated Texts

		Original Texts							Generated Texts						
ID	**Source**	**AC**	**EC**	**GC**	**FC**	**CS**	**MS**	**IS**	**AC**	**EC**	**GC**	**FC**	**CS**	**MS**	**IS**
1	HUB	97%	81 %	100%	100%	81%	4%	15%	100%	100%	100%	100%	84%	16%	0%
2	TUB	100%	91%	100%	100%	99%	0%	1%	100%	100%	100%	100%	87%	13%	0%
3	QUT	100%	100%	100%	100%	96%	0%	4%	100%	100%	100%	100%	95%	5%	0%
4	TUE	94%	100%	100%	100%	55%	3%	43%	100%	100%	100%	100%	92%	8%	0%
5	VT	100%	100%	100%	100%	94%	0%	6%	100%	100%	100%	100%	88%	12%	0%
6	IAG	100%	100%	100%	100%	91%	0%	14%	100%	100%	100%	100%	90%	10%	0%
7	BPMP	100%	100%	100%	100%	100%	0%	0%	100%	100%	100%	100%	88%	13%	0%
8	PHB	100%	100%	100%	100%	89%	0%	11%	100%	100%	100%	100%	100%	0%	0%
9	MRG	98%	91%	100%	100%	94%	1%	5%	100%	100%	100%	100%	85%	15%	0%
10	FNA	100%	100%	100%	100%	100%	0%	0%	100%	100%	100%	100%	77%	23%	0%
	Total	**99%**	**96%**	**100%**	**100%**	**91%**	**1%**	**8%**	**100%**	**100%**	**100%**	**100%**	**85%**	**15%**	**0%**

Legend: AC = Activity Coverage, EC = Event Coverage, GC = Gateway Coverage, FC = Sequence Flow Coverage, CS = Content Sentences, MS = Meta Sentences, IS = Information Sentences

social science which is referred to as *back translation* [105], [106]. Back translation is typically used in the context of cross-cultural research when, for instance, questionnaires need to be distributed in multiple languages. To make sure that the original and the translated version are semantically equivalent, the translated version is blindly converted back into the original language. Amongst others, this procedure enables to identify information loss and to detect unclear concepts. Since the introduced text generation technique can be considered as a translation from a process model to a natural language text, we adopt the instrument of back translation to our context. Assuming that the text generation produces an equivalent and well understandable natural language text, the back translation by humans should yield the original process model.

5.2.2 Test Data and Participants

For the back translation experiment, we selected 12 process models from the test collection introduced in Section 5.1.2. We subdivided the models into four sets, each consisting of three models. Table 8 summarizes the main characteristics of the four model sets and the comprised models. It shows that we selected models with challenging characteristics and structures, i.e., large models, models containing rigids or subprocesses, and models with a variety of different symbols. In this way, we aim at maximizing the external validity of the evaluation experiment.

As participants, we recruited 11 graduate and PhD students from the Humboldt University Berlin, Technical University of Berlin, Hasso-Plattner-Institute, and WU Vienna. Main criterion for participation was a solid knowledge of BPMN. Nevertheless, to increase the heterogeneity of the group, we also made sure that the recruited students differed with respect to the final grade they received for their bachelor or master studies. For the experiment, we randomly assigned each participant to one of the four model sets and asked the participants to transform the three comprised texts into BPMN process models using the Signavio editor. The participants did not receive further instructions about the texts. They were only asked to stick to the original order of the texts and to not include additional aspects that were not mentioned by the texts. The participants were allowed to spend as much time on the translation task as they needed. We, however, made sure that they did not use any external knowledge sources for solving the task.

TABLE 8
Characteristics of the Back Translation Model Sets

Set	**MID**	**A**	**E**	**G**	**R**	**SP**	**F**	**P**	**L**	**NS**
1	1	26	23	10	0	0	59	4	4	10
	2	8	4	7	1	0	22	1	1	5
	3	9	2	8	0	0	22	1	3	5
2	4	19	20	11	0	1	52	3	3	9
	5	10	9	9	1	0	31	1	5	8
	6	8	11	4	0	1	22	2	2	9
3	7	17	6	4	0	0	26	1	6	6
	8	8	10	7	0	2	25	1	1	8
	9	12	2	9	0	0	28	1	4	6
4	10	18	4	8	0	0	33	1	5	6
	11	7	9	1	0	0	16	1	1	7
	12	6	10	3	0	2	17	1	2	10

Legend: MID = Model ID, A = Activities, E = Events, G = Gateways, R = Rigids, SP = Subprocesses, F = Sequence Flows, P = Pools, L = Lanes, NS = Number of different Symbols

5.2.3 Back Translation Results

As a result from the user evaluation, we received 33 back translated BPMN process models. By manually comparing these models with their corresponding originals, we classified each process model aspect described by the text as *properly translated* or as

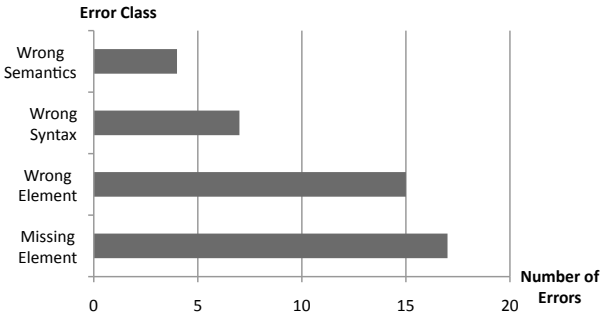


Fig. 8. Error Classes from Back Translation

erroneously translated. Table 9 gives an overview of the results by showing the share of aspects that have been properly translated.

In general, the results illustrate that the participants understood and interpreted the generated texts very well. From the 33 back translated process models, in total 22 models perfectly matched their original. Even the lowest obtained score of 83% still indicates that the text was well understood and that only details were erroneously translated. The fact that 3 out of 4 participants also understood the verbal rigid description highlights that the generated texts can also successfully describe complex process model behavior. However, a detailed consideration of the results also shows that a number of mistakes were made and that especially missing and wrong elements are the main reason for a non-perfect translation.

To get a better understanding of the back translation mistakes, consider the bar chart in Figure 8. It summarizes the different error classes of the back translation experiment and their frequency. The frequency of the error classes shows that the majority of the back translation mistakes stems from *missing elements*. Particularly in big models (e.g., see results for models 1 and 4 in Table 9), the participants left out activities and events. Although we cannot claim that all these cases were unintended, the general results from these participants strongly indicate that these mistakes did not result from a misunderstanding, but rather happened through an oversight. A similar observation can be made for errors that resulted from *wrong symbols* and *syntax errors*. Cases of wrong symbol use can be traced back to the confusion of the *send* and the *receive* message event and to the confusion of different gateway types. Syntax errors mainly occurred when sequence flows were missing. Again, both error types are more likely to be a result of a lack of skill with respect to BPMN than of a serious misunderstanding of the text. Nevertheless, we also observed mistakes that doubtlessly resulted from misinterpretations of the text. Such *semantic errors* were mainly caused by the rigid in process model 2. While this represents a weakness of the generated text, it also has to be noted that the participants were only provided with the text in isolation. The actual intention of the generated text is to represent a complementary artifact. Hence, we expect that the combination of model and text is reducing the ambiguity of the textual description such that the description is also helpful for understanding complex model semantics. Against the background that participant 1 and 2 understood the rigid description correctly, we are confident that this is actually the case.

Altogether, we can say that the back translation experiment indicated that the generated texts are understandable and successfully allow the reader to comprehend the semantics of the original process models. An analysis of the error sources revealed that also oversight and a lack of BPMN knowledge may have contributed to the share of erroneously transformed aspects and that only a small number of errors can be doubtlessly traced back to misinterpretations.

While these results are very promising, we also have to point out that there are a number of aspects potentially reducing the validity of the back translation experiment. The *external validity* is affected by two main factors. First, the group of participants was relatively small and only consisted of students. Hence, we cannot claim that our findings are representative. However, the consistently high performance among all 33 models makes us confident that the findings of our study are not a coincidence. Other studies have also found that students represent good proxies for young industry professionals [107]. Still, the benefit of the generated texts for industry users still needs to be investigated in future experiments. The second factor affecting the external validity is the familiarity of the participants with the domain of process modeling. Due to their process modeling knowledge, the participants already had some idea of what to expect from a text describing a business process and were used to terms such as ”parallel branch” or ”region”. Nevertheless, it has to be noted that the actual use case of the generated texts is less complex. Users would not be required to use a generated text for creating a process model but for developing an understanding of an existing model. In case novices struggle with understanding terms from the process modeling domain, the presented technique offers the possibility to adapt these terms without editing the source code. From the perspective of *construct validity*, it must be noted that the experiment did not demonstrate that generated texts help to improve the understanding of process models. We showed that participants could successfully make sense of the generated texts and were able to create a corresponding process model. While we believe that these two aspects are highly correlated, further studies are needed to prove this effect. For the *conclusion validity*, we want to highlight that our results do not allow to conclude about the statistical power. However, our results credibly show that the generated texts did not contain systematic weaknesses that prevented the

TABLE 9
Share of Correctly Back-Translated Process Model Elements

Set	PID	MID	A	E	G	R	SP	F	P	L	Total
1	1	1	96%	91%	100%	n.a.	n.a.	100%	100%	100%	98%
		2	100%	100%	100%	100%	n.a.	100%	100%	100%	100%
		3	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
	2	1	100%	91%	80%	n.a.	n.a.	100%	100%	100%	97%
		2	100%	100%	100%	100%	n.a.	100%	100%	100%	100%
		3	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
	3	1	96%	91%	100%	n.a.	n.a.	100%	100%	100%	98%
		2	100%	100%	86%	0%	n.a.	91%	100%	100%	91%
		3	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
2	4	4	100%	100%	100%	n.a.	100%	100%	100%	100%	100%
		5	100%	100%	100%	100%	n.a.	100%	100%	100%	100%
		6	100%	100%	100%	n.a.	100%	100%	100%	100%	100%
	5	4	95%	95%	100%	n.a.	100%	92%	100%	100%	94%
		5	90%	89%	78%	100%	n.a.	100%	100%	100%	95%
		6	88%	109%	75%	n.a.	100%	100%	100%	100%	98%
3	6	7	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		8	100%	100%	100%	n.a.	100%	100%	100%	100%	100%
		9	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
	7	7	100%	100%	100%	n.a.	100%	100%	100%	100%	100%
		8	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		9	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
	8	7	100%	80%	100%	n.a.	100%	100%	100%	100%	96%
		8	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		9	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
4	9	10	100%	75%	75%	n.a.	n.a.	100%	0%	0%	87%
		11	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		12	67%	80%	100%	n.a.	50%	88%	100%	100%	83%
	10	10	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		11	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		12	100%	100%	100%	n.a.	100%	100%	100%	100%	100%
	11	10	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		11	100%	100%	100%	n.a.	n.a.	100%	100%	100%	100%
		12	100%	100%	100%	n.a.	50%	100%	100%	100%	98%

Legend: MID = Model ID, PID = Participant ID, A = Activities, E = Events, G = Gateways, R = Rigids, SP = Subprocesses, F = Sequence Flows, P = Pools, L = Lanes, n.a. = not applicable

participants from successfully transforming them into the corresponding process models.

6 Adaptation to Other Languages

Although the general approach to text generation technique is not language-specific, the adaptation to languages other than English requires the replacement of three resources: the parsing and annotation component, the predefined DSynT templates, and the surface realizer.

As the generation technique builds on the linguistic information from the input process model, the replacement of the *parsing and annotation component* is an essential task. Without the proper inference of action, business object, and addition, it is not possible to generate a text using the proposed technique. However, as the annotation technique from [46] can be effectively adapted to other languages, this component can be respectively exchanged.

In addition to the parsing and annotation component, it is required to replace the *predefined DSynT-templates*. As these templates significantly contribute to the verbalization of events, bonds, and rigids, a proper adaptation is an important prerequisite. As an example, consider the fragment *The process begins when*, which is used to communicate a start event. In a German implementation of the generation technique we use the fragment *Der Prozess beginnt wenn* and in a Portuguese implementation we use the fragment *O processo começa quando*. In total, we defined about 50 different fragments. However, the templates can be translated with reasonable effort as they represent short and rather simple text fragments. In addition, it should be kept in mind that this represents a one time effort.

As a last step, the surface realization component of the generation approach needs to be replaced with a component that is capable of realizing sentences of

the target language. Hence, one option is to replace the English realization component with an off-the-shelf realizer of the target language. Although such tools have been developed for various languages including German [108], French [109], and Portuguese [110], their availability is often restricted. The second solution is to replace the English realizer with a self developed solution. In general, the development of a full-fledged realizer represents a tremendous effort that requires profound linguistic knowledge. By contrast, the development of a realizer for generating texts from process models is significantly less complex. The reason is given by the limited complexity of the generated sentences. As an example, consider the sentence *The Room-Service Manager checks the customer solvency*, which was generated from the activity *Check customer solvency* and its associated role *Room-Service Manager*. The generation of this sentence essentially requires the conjugation of the verb, and the insertion of definite articles for the role and the business object. Hence, a realizer for the purpose of generating texts from process models can be developed with manageable effort.

In conclusion, it must be stated that the adaptation of the text generation technique requires additional implementation effort. However, if the discussed components are available, the technique can be adapted to many different languages. Considering the overall benefits of the text generation, the associated effort for the adaptation can be considered as reasonable. Once the text generation technique is available, text can be generated and updated with a single click.

7 Conclusion

In this paper, we addressed the problem of supporting business process model validation through the generation of natural language texts from BPMN process models. Building on a literature review on natural language generation, we identified several challenges that are associated with generating natural language from process models. In order to adequately address these aspects, we introduced a pipeline architecture with six components. We used the parsing and annotation technique from [46] and the Refined Process Structure Tree [47], [48] to transform BPMN process models into textual descriptions. The technical evaluation of the technique using a set of 46 process models from different sources demonstrated the applicability of the presented technique for successfully generating texts from process models. The comparison of the generated and the manually created texts showed that the generated texts convey the model semantics in a more compact and also syntactically less complex manner. Due to the design of the technique, the generated texts are closer to the model and describe the model content and control flow explicitly. The user evaluation demonstrated that the generated texts are very informative and can successfully be interpreted by humans. Altogether, the evaluation demonstrated that the presented technique is capable of generating appropriate and understandable texts that fully explain the model semantics. The fact that the generation is accomplished within a few seconds and without any manual effort, further emphasizes the usefulness of the technique.

Based on our findings, we conclude that the text generation technique presented in this paper has the potential to facilitate the validation discourse between domain experts and process analysts. First, the generated texts support domain experts in understanding the details of process models even if they are not familiar with process modeling. Second, the text generation may also train domain experts in reading and interpreting process models. As long as text and model are presented together, readers can see and learn about the connection between model and text. Thus, their overall familiarity with process models can be expected to increase in the long term.

In future work, we aim at investigating to what extent industry participants can benefit from the generated texts. To this end, we plan to conduct a study in the context of an industrial requirements engineering project to empirically investigate the effect of the generation technique on process model validation. Furthermore, we plan to incorporate the technique into a modeling tool. This will provide us with additional opportunities to study the impact of the proposed generation technique.

References

[1] E. Cardoso, J. Almeida, and G. Guizzardi, "Requirements engineering based on business process models: A case study," in *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, 2009, pp. 320–327.

[2] D. Damian and J. Chisan, "An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management," *Software Engineering, IEEE Transactions on*, vol. 32, no. 7, pp. 433–453, 2006.

[3] S. Chakraborty, S. Sarker, and S. Sarker, "An exploration into the process of requirements elicitation: A grounded approach," *J. AIS*, vol. 11, no. 4, 2010.

[4] G. Verheijen and J. V. Bekkum, "NIAM, an information analysis method," in *Proceedings of the Conference on Comparative Review of Information System Methodologies*. North-Holland, 1982.

[5] G. Nijssen and T. Halpin, *Conceptual schema and relational database design: a fact oriented approach*. Prentice Hall, 1989.

[6] A. Gemino, "Empirical comparisons of animation and narration in requirements validation," *Requirements Engineering*, vol. 9, no. 3, pp. 153–168, 2004.

[7] H. Leopold, J. Mendling, and A. Polyvyanyy, "Generating natural language texts from business process models," in *Proceedings of the 24th International Conference on Advanced Information Systems Engineering*, 2012.

[8] F. Friedrich, "Automated generation of business process models from natural language input," Master's thesis, Humboldt Universität zu Berlin, 2010.

[9] P. Frederiks and T. van der Weide, "Information modeling: The process and the required competencies of its participants," *Data & Knowledge Engineering*, vol. 58, no. 1, pp. 4–20, 2006.

[10] A. Davis, O. Dieste, A. Hickey, N. Juristo, and A. Moreno, "Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review," in *Requirements Engineering, 14th IEEE International Conference*, 2006, pp. 179–188.

[11] L. Goldin and D. M. Berry, "Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation," *Automated Software Engineering*, vol. 4, no. 4, pp. 375–412, 1997.

[12] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, 2007.

[13] D. Popescu, S. Rugaber, N. Medvidovic, and D. Berry, "Reducing ambiguities in requirements specifications via automatically created object-oriented models," in *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, ser. Lecture Notes in Computer Science, B. Paech and C. Martell, Eds. Springer Berlin Heidelberg, 2008, vol. 5320, pp. 103–124.

[14] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven Process Modeling Guidelines (7PMG)," *Information and Software Technology*, vol. 52, no. 2, pp. 127–136, 2010.

[15] R. Schuette and T. Rotthowe, "The guidelines of modeling - an approach to enhance the quality in information models," in *Proceedings of the 17th International Conference on Conceptual Modeling*. London, UK: Springer-Verlag, 1998, pp. 240–254.

[16] J. Krogstie, O. I. Lindland, and G. Sindre, "Defining quality aspects for conceptual models," in *Proceedings of the international working conference on Information system concepts: Towards a consolidation of views*. Chapman & Hall, Ltd., 1995, pp. 216–231.

[17] R. Davis and E. Brabänder, *ARIS Design Platform: Getting Started with BPM*, 1st ed. Springer, 2007.

[18] W. M. P. van der Aalst, *Business Process Management*, ser. LNCS. Springer, 2000, vol. 1806, ch. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques, pp. 161–183.

[19] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf, "Analysis on Demand: Instantaneous Soundness Checking of Industrial Business Process Models," *Data & Knowledge Engineering*, vol. 70, no. 5, pp. 448–466, 2011.

[20] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, ser. LNBIP. Springer, 2008, vol. 6.

[21] S. Sun, J. Zhao, J. Nunamaker, and O. Liu Sheng, "Formulating the Data-Flow Perspective for Business Process Management," *Information Systems Research*, vol. 17, no. 4, pp. 374–391, 2006.

[22] I. Weber, J. Hoffmann, and J. Mendling, "Beyond Soundness: on the Verification of Semantic Business Process Models," *Distributed and Parallel Databases*, vol. 27, no. 3, pp. 271–343, 2010.

[23] N. Sidorova, C. Stahl, and N. Trcka, "Soundness Verification for Conceptual Workflow Nets with Data: Early Detection of Errors with the Most Precision Possible," *Information Systems*, vol. 36, no. 7, pp. 1026–1043, 2011.

[24] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 65–104, February 1999.

[25] J. Crampton and H. Khambhammettu, "Delegation and Satisfiability in Workflow Systems," in *SACMAT 2008*. New York, NY, USA: ACM, 2008, pp. 31–40.

[26] M. Strembeck and J. Mendling, "Modeling Process-related RBAC Models with Extended UML Activity Models," *Information and Software Technology*, vol. 53, no. 5, pp. 456–483, 2011.

[27] M. Dumas, M. Rosa, J. Mendling, and H. Reijers, *Fundamentals of Business Process Management*. Springer, 2013.

[28] M. Rosemann, "Potential Pitfalls of Process Modeling: Part A," *Business Process Management Journal*, vol. 12, no. 2, pp. 249–254, 2006.

[29] O. Lindland and J. Krogstie, "Validating conceptual models by transformational prototyping," in *Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, C. Rolland, F. Bodart, and C. Cauvet, Eds. Springer Berlin Heidelberg, 1993, vol. 685, pp. 165–183.

[30] P. Loucopoulos, B. Theodoulidis, and D. Pantazis., "Business rules modelling: Conceptual modelling and object oriented specifications," in *Proceedings of the IFIP TC8/WG8.1 Working Conference*, 1991, pp. 323–342.

[31] G. De Caso, V. Braberman, D. Garbervetsky, and S. Uchitel, "Automated abstractions for contract validation," *Software Engineering, IEEE Transactions on*, vol. 38, no. 1, pp. 141–162, 2012.

[32] T. D. Breaux, A. I. Antón, and J. Doyle, "Semantic parameterization: A process for modeling domain descriptions," *ACM Trans. Softw. Eng. Methodol.*, vol. 18, no. 2, pp. 5:1–5:27, 2008.

[33] A. Sutcliffe, N. A. M. Maiden, S. Minocha, and D. Manuel, "Supporting scenario-based requirements engineering," *Software Engineering, IEEE Transactions on*, vol. 24, no. 12, pp. 1072–1088, 1998.

[34] P. Heymans and E. Dubois, "Scenario-based techniques for supporting the elaboration and the validation of formal requirements," *Requirements Engineering*, vol. 3, no. 3-4, pp. 202–218, 1998.

[35] V. Lalioti and P. Loucopoulos, "Visualisation for validation," in *Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, C. Rolland, F. Bodart, and C. Cauvet, Eds. Springer Berlin Heidelberg, 1993, vol. 685, pp. 143–164.

[36] A. Mashkoor and A. Matoussi, "Towards validation of requirements models," in *Abstract State Machines, Alloy, B and Z*, ser. Lecture Notes in Computer Science, M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, Eds. Springer Berlin Heidelberg, 2010, vol. 5977, pp. 404–404.

[37] L. Kof, R. Gacitua, M. Rouncefield, and P. Sawyer, "Ontology and model alignment as a means for requirements validation," in *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, 2010, pp. 46–51.

[38] D. Costal, E. Teniente, T. Urpí, and C. Farré, "Handling conceptual model validation by planning," in *Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, Eds. Springer Berlin Heidelberg, 1996, vol. 1080, pp. 255–271.

[39] A. Queralt and E. Teniente, "Verification and validation of uml conceptual schemas with ocl constraints," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 2, pp. 13:1–13:41, 2012.

[40] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *Software Engineering, IEEE Transactions on*, vol. 37, no. 2, pp. 188–204, 2011.

[41] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating natural language specifications from uml class diagrams," *Requirements Engineering*, vol. 13, pp. 1–18, 2008.

[42] B. Lavoie, O. Rambow, and E. Reiter, "The modelexplainer," in *Proceedings of the 8th international workshop on natural language generation*, 1996, pp. 9–12.

[43] H. Dalianis, "A method for validating a conceptual model by natural language discourse generation," in *Proceedings of the 4th international conference on Advanced Information Systems Engineering*, 1992, pp. 425–444.

[44] S. Malik and I. S. Bajwa, "Back to origin: Transformation of business process models to business rules," in *Business Process Management Workshops*. Springer, 2013, pp. 611–622.

[45] A. Coşkunçay, "An approach for generating natural lanaguage specifications by utilizing business process models," Master's thesis, The Middle East Technical University, 2010.

[46] H. Leopold, S. Smirnov, and J. Mendling, "On the refactoring of activity labels in business process models," *Information Systems*, vol. 37, no. 5, pp. 443–459, 2012.

[47] J. Vanhatalo, H. Völzer, and J. Koehler, "The refined process structure tree," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 793 – 818, 2009.

[48] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified computation and generalization of the refined process structure tree," in *Web Services and Formal Methods*, ser. LNCS. Springer, 2011, vol. 6551, pp. 25–41.

[49] O. Heinonen, "Optimal multi-paragraph text segmentation by dynamic programming," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*, ser. ACL '98. Stroudsburg, PA, USA: Association for Computational Linguistics, 1998, pp. 1484–1486.

[50] M. W. Meteer, *Expressibility and the Problem of Efficient Text Planning*. New York, NY, USA: St. Martin's Press, Inc., 1992.

[51] M. Meteer, "Bridging the generation gap between text planning and linguistic realization," *Computational Intelligence*, vol. 7, no. 4, pp. 296–304, 1991.

[52] S. Bangalore and O. Rambow, "Corpus-based lexical choice in natural language generation," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ser. ACL '00. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000, pp. 464–471.

[53] M. Stede, "Lexical choice criteria in language generation," in *Proceedings of the 6th conference on European chapter of the Association for Computational Linguistics*, ser. EACL '93. Stroudsburg, PA, USA: Association for Computational Linguistics, 1993, pp. 454–459.

[54] ——, "Lexicalization in natural language generation: A survey," *Artificial Intelligence Review*, vol. 8, pp. 309–336, 1994.

[55] E. H. Hovy, "Aggregation in natural language generation," in *Proceedings of the 4th European Workshop on Natural Language Generation*, 1993, pp. 28–30.

[56] M. Reape and C. Mellish, "Just what is aggregation anyway?" 2010.

[57] H. Dalianis, "Aggregation in natural language generation," *Computational Intelligence*, vol. 15, no. 4, pp. 384–414, 1999.

[58] R. Kibble and R. Power, "An integrated framework for text planning and pronominalisation," in *Natural language generation*. ACL, 2000, pp. 77–84.

[59] B. Lavoie and O. Rambow, "A fast and portable realizer for text generation systems," in *Applied natural language processing*. ACL, 1997, pp. 265–268.

[60] S. Busemann, "Best-first surface realization," 1996.

[61] M. Elhadad and J. Robin, "Surge: a comprehensive plug-in syntactic realization component for text generation," Tech. Rep., 1998.

[62] S. W. Mcroy, S. Channarukul, and S. S. Ali, "Text realization for dialog," in *Proceedings of the International Conference on Intelligent Technologies*, 2000.

[63] K. McKeown and D. R. Radev, "Generating summaries of multiple news articles," in *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '95. New York, NY, USA: ACM, 1995, pp. 74–82.

[64] D. R. Radev and K. R. McKeown, "Generating natural language summaries from multiple on-line sources," *Computational Linguistics*, vol. 24, no. 3, pp. 470–500, 1998.

[65] E. H. Hovy, "Pragmatics and natural language generation," *Artificial Intelligence*, vol. 43, pp. 153–197, May 1990.

[66] E. Goldberg, N. Driedger, and R. Kittredge, "Using natural-language processing to produce weather forecasts," *IEEE Expert*, vol. 9, no. 2, pp. 45–53, 1994.

[67] J. A. Bateman, "Enabling technology for multilingual natural language generation: the kpml development environment," *Natural Language Engineering*, vol. 3, no. 1, pp. 15–55, 1997.

[68] W. Wahlster, E. Andre, W. Finkler, H.-J. Profitlich, and T. Rist, "Plan-based integration of natural language and graphics generation," *Artificial Intelligence*, vol. 63, pp. 387 – 427, 1993.

[69] M. Meteer, "Portable natural language generation using spokesman," in *Proceedings of the 3rd conference on Applied natural language processing*, ser. ANLC '92. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 237–238.

[70] I. Mel'cuk and A. Polguère, "A formal lexicon in the meaning-text theory (or how to do lexica with words)," *Computational Linguistics*, vol. 13, no. 3-4, pp. 261–275, 1987.

[71] M. Kay., "Functional grammar," in *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*, 1979.

[72] G. Abrett, M. Burstein, and S. Deutsch, "Tarl: Tactical action representation language, an environment for building goal directed knowledge based simulations," BBN, Tech. Rep. 7062, 1989.

[73] E. Reiter, "Nlg vs. templates," in *Proceedings of the 5th European Workshop on Natural Language Generation*, 1995, pp. 95–106.

[74] E. Reiter and R. Dale, "Building applied natural language generation systems," *Natural Language Engineering*, vol. 3, pp. 57–87, 1997.

[75] Object Management Group, *Business Process Model and Notation (BPMN)*, 2011.

[76] T. Allweyer, *BPMN 2.0 - Business Process Model and Notation*, 2nd ed. Norderstedt: Books on De-mand GMBH, 2009.

[77] R. M. Dijkman, M. Dumas, and C. Ouyang, "Formal semantics and analysis of bpmn process models using petri nets," Queensland University of Technology, Tech. Rep., 2007.

[78] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring acyclic process models," *Information Systems*, vol. 37, no. 6, pp. 518 – 538, 2012.

[79] M. A. Hearst, "Multi-paragraph segmentation of expository text," in *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, ser. ACL '94. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, pp. 9–16.

[80] ——, "Texttiling: segmenting text into multi-paragraph subtopic passages," *Computational Linguistics*, vol. 23, no. 1, pp. 33–64, 1997.

[81] J. Morris and G. Hirst, "Lexical cohesion computed by thesaural relations as an indicator of the structure of text," *Computational Linguistics*, vol. 17, no. 1, pp. 21–48, 1991.

[82] G. E. Hynes and J. B. Bexley, "Understandability of banks' annual reports," in *Proceedings of the 69th Association for Business Communication Annual Convention*, 2003.

[83] S. Kahane, "What is a natural language and how to describe it? meaning-text approaches in contrast with generative approaches," in *Proceedings of the 2nd International Conference on Computational Linguistics and Intelligent Text Processing*. Springer Verlag, 2001, pp. 1–17.

[84] U. Goltz and W. Reisig, "The non-sequential behaviour of petri nets," *Information and Control*, vol. 57, no. 2-3, pp. 125 – 147, 1983.

[85] I. Lomazova, "On occurrence net semantics for petri nets with contacts," in *Fundamentals of Computation Theory*, ser. LNCS, B. Chlebus and L. Czaja, Eds. Springer Berlin Heidelberg, 1997, vol. 1279, pp. 317–328.

[86] G. Decker, R. M. Dijkman, M. Dumas, and L. García-Bañuelos, "Transforming bpmn diagrams into yawl nets," in *Proceedings of the 6th International Conference on Business Process Management*, ser. BPM '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 386–389.

[87] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in bpmn," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[88] N. Lohmann, E. Verbeek, and R. Dijkman, "Petri net transformations for business processes – a survey," in *Transactions on Petri Nets and Other Models of Concurrency II*, ser. Lecture Notes in Computer Science, K. Jensen and W. Aalst, Eds. Springer Berlin Heidelberg, 2009, vol. 5460, pp. 46–63.

[89] M. Weidlich, A. Polyvyanyy, N. Desai, and J. Mendling, "Process compliance measurement based on behavioural profiles," in *Proceedings of the 22nd international conference on Advanced information systems engineering*, ser. CAiSE'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 499–514. [Online]. Available: http://dl.acm.org/citation.cfm?id=1883784.1883833

[90] M. A. Just and P. A. Carpenter, "A capacity theory of comprehension: individual differences in working memory." *Psychological review*, vol. 99, no. 1, pp. 122–149, Jan. 1992.

[91] K. Rayner and A. Pollatsek, *The Psychology of Reading*, ser. Prentice-Hall international editions. Prentice-Hall, 1989.

[92] X. Lu, "Automatic analysis of syntactic complexity in second language writing," *International Journal of Corpus Linguistics*, vol. 15, no. 4, pp. 474–496, 2010.

[93] E. Page, "The imminence of grading essays by computer," *The Phi Delta Kappan*, vol. 47, no. 5, pp. 238–243, 1966.

[94] E. B. Page, "Statistical and linguistic strategies in the computer grading of essays," in *Proceedings of the conference on Computational linguistics*, ser. COLING '67. Stroudsburg, PA, USA: Association for Computational Linguistics, 1967, pp. 1–13.

[95] J. Burstein, "The e-rater scoring engine: Automated essay scoring with natural language processing," in *Automated essay scoring: A cross-disciplinary perspective*, M. D. Shermis and J. Burstein, Eds. Lawrence Erlbaum Associates, 2003, pp. 113–122.

[96] T. Landauer, D. Laham, and P. Foltz, "Automated scoring and annotation of essays with the intelligent essay assessortm," in *Automated Essay Scoring: A Cross-Disciplinary Perspective*, M. D. Shermis and J. Burstein, Eds. Lawrence Erlbaum, 2003.

[97] Y. Attali, J. Burstein, Y. Attali, J. Burstein, M. Russell, D. T. Hoffmann, Y. Attali, and J. Burstein, "The automated essay

scoring with e-rater v.2," *Journal of Technology, Learning, and Assessment*, 2006.
[98] K. W. Hunt, "Do sentences in the second language grow like those in the first?" *TESOL Quarterly*, vol. 4, pp. 195–202, 1970.
[99] C. P. Casanave, "Language development in students' journals," *Journal of Second Language Writing*, vol. 3, no. 3, pp. 179 – 201, 1994.
[100] O. Holschke, "Impact of granularity on adjustment behavior in adaptive reuse of business process models," in *Proceedings of the 8th international conference on Business process management*, ser. BPM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 112–127.
[101] ——, "Granularität als kognitiver Faktor in der adaptiven Wiederverwendung von Geschäftsprozessmodellen," Ph.D. dissertation, Technische Universität Berlin, 2010.
[102] H. A. Reijers, S. Limam, and W. M. P. van der Aalst, "Product-based workflow design," *Journal of Management Information Systems*, vol. 20, no. 1, pp. 229–262, 2003.
[103] J. Freund and B. Rücker, *Praxishandbuch BPMN 2.0*, 3rd ed. Carl Hanser Verlag GmbH & CO. KG, 2012.
[104] S. White and D. Miers, *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Future Strategies Incorporated, 2008.
[105] R. W. Brislin, *Field methods in cross-cultural research*. Sage, 1986, ch. The Wording and Translation of Research Instruments.
[106] D. W. Chapman and J. F. Carter, "Translation procedures for the cross cultural use of measurement instruments," *Educational Evaluation and Policy Analysis*, vol. 1, no. 3, pp. 71–76, 1979.
[107] E. Arisholm and D. I. Sjoberg, "Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software," *Software Engineering, IEEE Transactions on*, vol. 30, no. 8, pp. 521–534, 2004.
[108] S. Corston-Oliver, M. Gamon, E. Ringger, and R. Moore, "An overview of amalgam: A machine-learned generation module," in *Proceedings of the International Natural Language Generation Conference*, 2002, pp. 33–40.
[109] M. Smets, M. Gamon, S. Corston-Oliver, and E. Ringger, "The adaptation of a machine-learned sentence realization system to french," in *Proceedings of the 10th conference on European chapter of the Association for Computational Linguistics*, ser. EACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 323–330.
[110] E. M. de Novais, T. D. Tadeu, and I. Paraboni, "Text generation for brazilian portuguese: the surface realization task," in *Proceedings of the Workshop on Computational Approaches to Languages of the Americas*, ser. YIWCALA '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 125–131.