Supporting the Process of Learning and Teaching Process Models

Josep Sànchez-Ferreres, Luis Delicado, Amine Abbad Andaloussi, Andrea Burattin, Guillermo Calderón-Ruiz, Barbara Weber, Josep Carmona, and Lluís Padró

***Abstract*—The creation of a process model is primarily a formalization task that faces the challenge of constructing a syntactically correct entity which accurately reflects the semantics of reality, and is understandable to the model reader. This paper proposes a framework called *Model Judge*, focused towards the two main actors in the process of learning process model creation: novice modelers and instructors. For modelers, the platform enables the automatic validation of the process models created from a textual description, providing explanations about quality issues in the model. *Model Judge* can provide diagnostics regarding model structure, writing style, and semantics by aligning annotated textual descriptions to models. For instructors, the platform facilitates the creation of modeling exercises by providing an editor to annotate the main parts of a textual description, that is empowered with natural language processing (NLP) capabilities so that the annotation effort is minimized. So far around 300 students, in process modeling courses of five different universities around the world have used the platform. The feedback gathered from some of these courses shows good potential in helping students to improve their learning experience, which might, in turn, impact process model quality and understandability. Moreover, our results show that instructors can benefit from getting insights into the evolution of modeling processes including arising quality issues of single students, but also discovering tendencies in groups of students. Although the framework has been applied to process model creation, it could be extrapolated to other contexts where the creation of models based on a textual description plays an important role.**

***Index Terms*—Natural language processing (NLP), process of process modeling, textual annotation, Business Process Model and Notation (BPMN).**

I. Introduction and Motivation

PROCESS models play an important role in the analysis and improvement of business processes [1]. Moreover, they are often used as basis for process execution. Due to their wide usage in organizations their quality is paramount: Process models should be syntactically correct, and follow the rules provided by the modeling grammar (i.e., syntactic quality), they should be complete in terms of requirements and only contain correct and relevant statements of the domain (i.e., semantic quality), and be understandable to the human model reader (i.e., pragmatic quality) [2], [3].

Research has shown that industrial process models often contain errors [4]. While some of these errors can be automatically identified by the verification support of state-of-the art modeling environments [5], others—especially those concerning the semantics of a model—are not sufficiently supported up to now. Moreover, a recently conducted exploratory study by Haisjackl *et al.* investigated how humans inspect process models [6], and came to the conclusion that a systematic support for model inspection would be fundamental, either in form of test-driven development, or in form of (automated) checklists.

This paper picks up this challenge, and proposes a framework called *Model Judge* for the automatic validation of process models and providing explanations about quality issues in therein. The framework proposed is inspired by similar practices in other educational contexts, e.g., the success of judges in supporting learning to program [7]–[11]. Remarkably, we use natural language processing (NLP), together with optimization techniques [12], to validate process models with respect to textual descriptions of processes.

Model Judge provides diagnostics regarding issues on syntactic, pragmatic and semantic quality. The framework underlying *Model Judge* is grounded on aligning annotated textual descriptions with the models as basis for providing diagnostics [12]. Using *Model Judge*, novice modelers can be guided through continuous feedback. Instructors, in turn, can create new exercises, and obtain insights into modeling behaviors and better understand the evolution of error types and their lifetime in the model. Currently, we have applied our tool it in the context of five modeling courses totaling around 300 students. To demonstrate the applicability of our framework we monitored and analysed student data from one of the courses with 26 students and the results show that *Model Judge* has potential to support process modeling learners in improving the quality of their models.

In this paper we apply our framework to the creation of process models. However, a similar approach could be taken for other types of models, like Universal Modeling Language (UML) class diagrams or UML activity diagrams, in settings where learners need to create a model starting from a textual

Manuscript received September 19, 2019; revised March 18, 2020; accepted **MM DD, 2020**. Date of publication **MM DD, 2020**; date of current version **MM DD, 2020**. This work was supported in part by the Spanish Ministry of Science, Innovation and Universities, under the grant TIN2017-86727-C2-1-R. (*Corresponding author: Josep Carmona*).

J. Sànchez-Ferreres, L. Delicado, J. Carmona, and L. Padró are with the Department of Computer Science, Universitat Politècnica de Catalunya, Spain (e-mail: jsanchezf@cs.upc.edu; ldelicado@cs.upc.edu; jcarmona@cs.upc.edu; padro@cs.upc.edu).

A. Abad Andaloussi and A. Burattin are with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark (e-mail: amab@dtu.dk; andbur@dtu.dk).

B. Weber is with the Institute for Computer Science, University of St. Gallen, Switzerland (e-mail: barbara.weber@unisg.ch).

G. Calderón-Ruiz is with the Department of Physics and Formal Sciences and Engineering, Universidad Catolica de Santa María, Peru (e-mail: gcalderonr@gmail.com).

Digital Object Identifier: **Add doi here when available**

description.

The remainder of this paper is structured as follows: Section II gives an overall description of the *Model Judge* platform and its features. Section III explores the related work in the existing literature. Section IV provides the preliminaries for our work. Subsequently, Section V presents the specific modeling guidelines that were used in the design of *Model Judge* as well as the details behind the implementation of the platform. Finally, Section VI evaluates *Model Judge*'s performance in a modeling course before Section VII concludes the paper.

II. A Tour Through the Model Judge

This section describes the main features that users can find in the *Model Judge*. The *Model Judge* is a web-based, platform, that can be accessed through any web browser at https://modeljudge.cs.upc.edu. It is designed both for helping students in the process of creating a process model, and instructors in the task of designing modeling exercises in an agile way.

To use *Model Judge*, a user (being student or instructor) needs to create an account, with its associated storage space. In this space, a student can save models, which will be always accessible once she logs in. On the other hand, instructors have the ability to create courses and exercises. A course is designed as a set of exercises the students must solve, which can be taken from the existing list of examples, or created by the instructors.

Registered students can attempt to solve any exercise for the courses they are enrolled to. This enrolling process is performed by introducing a course code provided by the instructor. Upon selecting an exercise, the main working zone will be displayed (see Fig. 1): on the left part, a textual description of the process is provided, so that the student understands the process to model. Next to the textual description, a model editor allows the students to create the process model. The students enrolled in a particular course may enable the platform to record every few minutes a copy of their work-in-progress. This can be used for analyzing their behavior, which can be then reported back to both the student and the instructor.

During the modeling session, two important checks are available to help novice modelers:

- ***Validation***: returns an aggregated diagnostic that reflects if the model has some errors (more details on the types of errors detected by *Model Judge* can be seen in Section V-C), however, the exact source of the error is omitted from this message. The motivation for this check is to allow for a mild test to guide the students. This is shown at the right part of Fig. 1, using a traditional color code (green meaning positive facts, red meaning negative facts, and yellow meaning warnings).
- ***Complete Validation***: This check provides a more thorough list of all the problems detected, which individually addresses each of the issues with the model. The motivation for this check is to allow an assessment similar to the one obtained if a teacher was correcting the model.

Two important monitoring features are included in *Model Judge* for instructors to get further insights from the modeling sessions. On the one hand, instructors have access to a real-time dashboard during the modeling session that can be used to monitor the students' progress. This dashboard allows inspecting useful statistics like how much are students using the validation functionality, their overall progress measured in the number of error diagnostics, and the actual process models. This dashboard is shown in Fig. 2.

On the other hand, instructors have the option to access the data stored from students of the course once their modeling task is finished. This data contains the final models, which can be used for assessment, but also the intermediate models and their diagnostics in order to perform a more detailed analysis with the progression of the students during the exercise, similar to what is shown in Section VI.

Furthermore, an exercise editor is available for instructors, in order to create new exercises. For this, instructors must provide both the problem statement text and its corresponding annotations, which are used to describe the exercise solution to *Model Judge* (see Section V). An on-line exercise editor is provided which handles the generation of partial annotations and allows its further refinement, as seen in Fig. 3. Typically, the necessary work to create a new exercise consists of removing the automatically highlighted elements that are not relevant from the process perspective, as well as manually annotating any activities that were implicitly described. From our experience with the tool, we have estimated that a trained instructor can create a new exercise for a well-known process in less than an hour.

In summary, the *Model Judge* platform consists of three inter-related tools, each focused on a different aspect of an exercise's lifecycle. We next detail the ecosystem behind our platform, with the main functionalities of every tool.

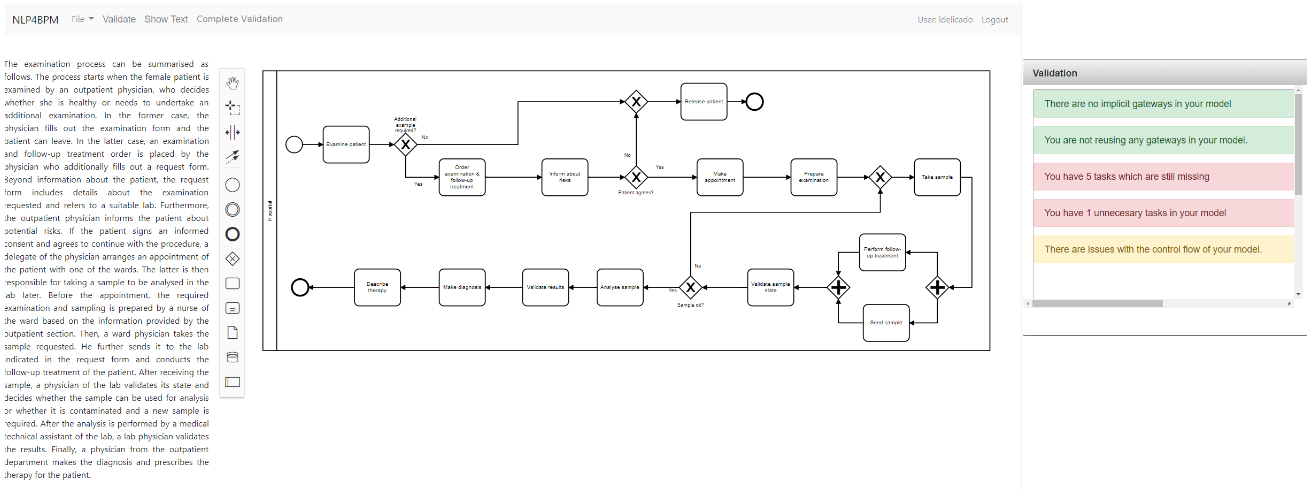
1) MJ-Core: MJ-Core is the main tool in the *Model Judge* platform. Its main purpose is to present a space for students to solve exercises in, and instructors to create and manage their courses. Additionally, the core platform provides the login and registration capabilities for all other applications in the platform. The source code for this tool is available inside the *modeljudge* project at our source code repository [13].

2) MJ-Editor: MJ-Editor is a tool aimed at instructors that allows the creation of new exercises for the core platform. In order to create a new exercise, a natural language description of a process must be annotated. This editor, shown in Fig. 3, is a standalone web application which can also be used offline. This editor is built using the React framework for Javascript, and its source code is also available online inside the *atd-editor* project at our source code repository [13].

3) MJ-Dashboard: MJ-Dashboard is a tool that helps instructors to follow their existing courses by summarizing student and exercise data. The tool is built using a custom framework on top of the *Compojure* library. Currently it is only available to instructors, due to the advanced nature of the information displayed.

III. Related Work

There exist very few works in the literature that jointly propose a learning framework for process modeling, and



The examination process can be summarised as follows. The process starts when the female patient is examined by an outpatient physician, who decides whether she is healthy or needs to undertake an additional examination. In the former case, the physician fills out the examination form and the patient can leave. In the latter case, an examination and follow-up treatment order is placed by the physician who additionally fills out a request form. Beyond information about the patient, the request form includes details about the examination requested and refers to a suitable lab. Furthermore, the outpatient physician informs the patient about potential risks. If the patient signs an informed consent and agrees to continue with the procedure, a delegate of the physician arranges an appointment of the patient with one of the wards. The latter is then responsible for taking a sample to be analysed in the lab later. Before the appointment, the required examination and sampling is prepared by a nurse of the ward based on the information provided by the outpatient section. Then, a ward physician takes the sample requested. He further sends it to the lab indicated in the request form and conducts the follow-up treatment of the patient. After receiving the sample, a physician of the lab validates its state and decides whether the sample can be used for analysis or whether it is contaminated and a new sample is required. After the analysis is performed by a medical technical assistant of the lab, a lab physician validates the results. Finally, a physician from the outpatient department makes the diagnosis and prescribes the therapy for the patient.

There are no implicit gateways in your model

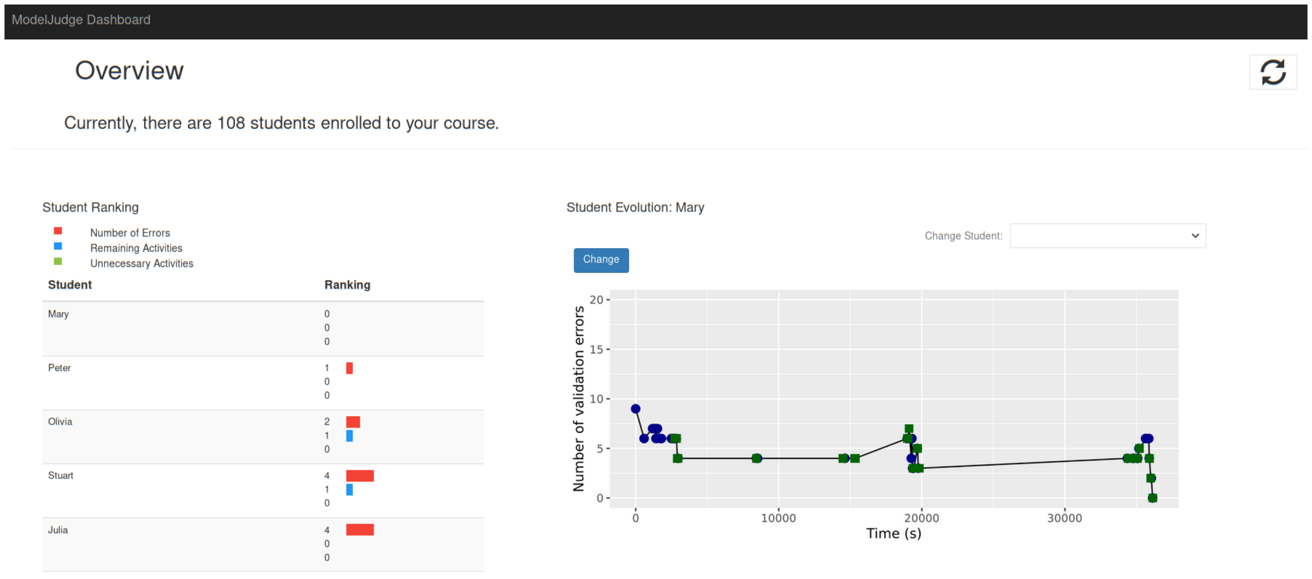
You are not reusing any gateways in your model.

You have 5 tasks which are still missing

You have 1 unnecessary tasks in your model

There are issues with the control flow of your model.

Fig. 1. The *Model Judge*: (Left) The textual description, (Center) the created model, (Right) The result of a validation.



Overview

Currently, there are 108 students enrolled to your course.

Student Ranking

🟥 Number of Errors
🟦 Remaining Activities
🟩 Unnecessary Activities

Student	Ranking
Mary	0 0 0
Peter	1 0 0
Olivia	2 1 0
Stuart	4 1 0
Julia	4 0 0

Student Evolution: Mary

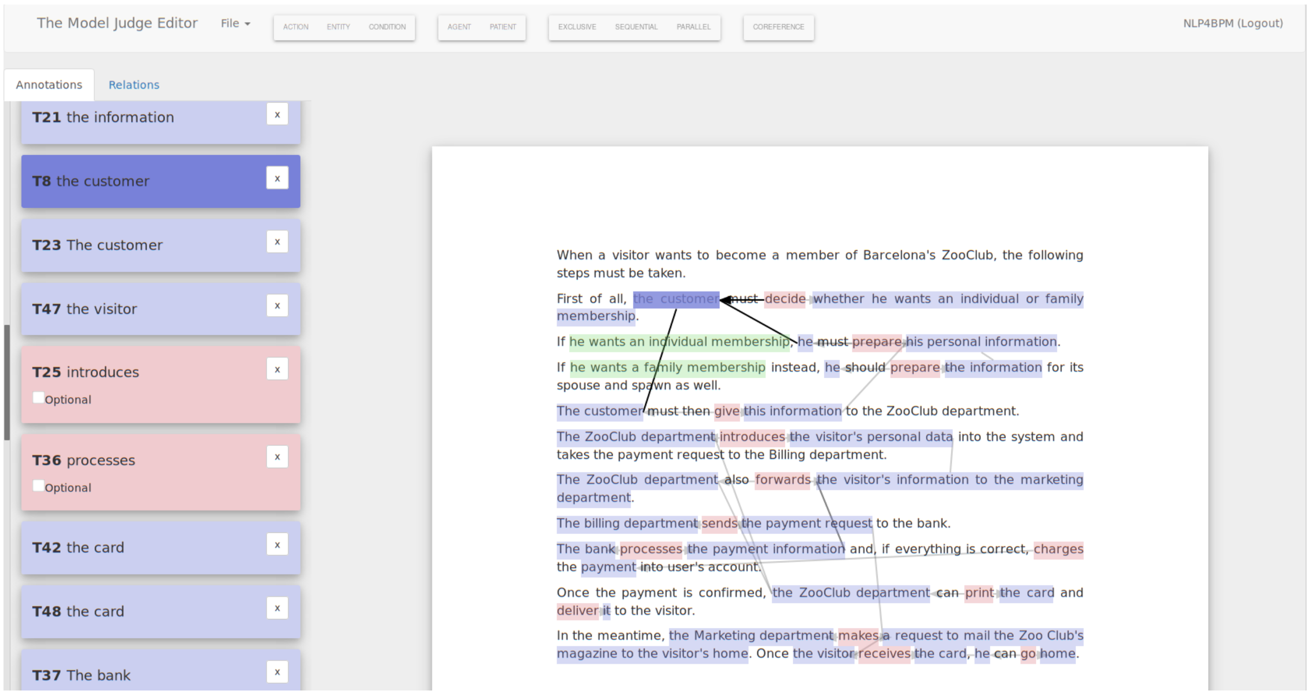
Fig. 2. The *Model Judge* Dashboard: (Left) ranking of students according to the gathered diagnostics (left), (Right) selected student evolution.

exploit NLP capabilities for automating the validation of created process models. Still, we can find existing examples of learning frameworks designed to aid students in conceptual modeling, such as the one presented in Pimentel *et. al*, where a gamification setup is defined to engage students in the task of creating $i*$ models [14]. Another are the works presented in Bogandova *et. al* and Buchmann *et. al*, which aim to create a taxonomy of common student errors when learning conceptual modeling [15], [16].

There have also been prior efforts to design e-learning systems for modeling notations, such as UML [17]. In this system, a very shallow textual validation for the main model components is done, and it is required to decide a priori all the possible solutions for a given exercise. Our framework is much more flexible, so that the space of solutions is merely defined by the problem statement itself. NLP applied to both text and model enables to implicitly account for all possible correct solutions without enumerating them explicitly.

There is related work that explores the human behavior in modeling, an aspect that influences the work of this paper. In Störrle *et al.*, a study is conducted to evaluate how human modelers visually parse UML diagrams using eye-tracking technology [18]. A more recent study follows a similar approach for the case of reading process models during a modeling session [19]. Incorporating insights arising from these studies, as well as well-known principles and guidelines, can be used to automatically improve the readability of a model [20].

Finally, the interaction between NLP and Business Process Management has some prior studies outside the scope of education. There has been work addressing the problem of the translation from textual to graphical representations of models (e.g., UML diagrams [21], [22], or Business Process Model and Notation (BPMN) [23], [24]) as well as schema [25] or process model [26] matching. The alignment technique presented in this paper is an adapted version of a prior work



T21 the information ×

T8 the customer ×

T23 The customer ×

T47 the visitor ×

T25 introduces ×
Optional

T36 processes ×
Optional

T42 the card ×

T48 the card ×

T37 The bank ×

When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken.

First of all, the customer must decide whether he wants an individual or family membership.

If he wants an individual membership, he must prepare his personal information.

If he wants a family membership instead, he should prepare the information for its spouse and spawn as well.

The customer must then give this information to the ZooClub department.

The ZooClub department introduces the visitor's personal data into the system and takes the payment request to the Billing department.

The ZooClub department also forwards the visitor's information to the marketing department.

The billing department sends the payment request to the bank.

The bank processes the payment information and, if everything is correct, charges the payment into user's account.

Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor.

In the meantime, the Marketing department makes a request to mail the Zoo Club's magazine to the visitor's home. Once the visitor receives the card, he can go home.

Fig. 3. The *Model Judge* Exercise Editor: the text describing the process (right) can be annotated (left) and relations between annotations reported (arcs in the text).

[12].

To the best of our knowledge, the proposal of this paper is the first one to tackle the challenges in the development of an automated system to support novices in the creation of business process models. Commercial tools (e.g., Signavio Process Manager) are already capable of providing feedback concerning some predefined modeling conventions. This feedback, however, is typically quite elementary and completely domain agnostic. Instead, the framework proposed is inspired on code judges for programming, which are extremely successful on guiding novice programmers towards obtaining solutions to programming exercises [7]–[11] by providing suggestions specifically tailored to the context of the exercise. As it is done in some of the aforementioned learning environments (e.g., Petit *et al.* [11]), in our framework we allow for a detailed diagnostic list, enabling the modeler to easily locate errors.

IV. Preliminaries

A. Contextual Example: Modeling from Textual Descriptions

To illustrate the kind of modeling task we address in our work, let us consider the process of patient examination based on the description provided by the Women's Hospital of Ulm. Both the text and the model are extracted from the paper by Cabanillas *et al.* [27]. A part of the textual description of this process is provided in Fig. 4. A novice modeler would be confronted to this text in order to perform the creation of the corresponding process model. Fig. 5 shows a possible solution to this modeling task, i.e., a process model in BPMN [28].

For readers not familiar with the BPMN language's features, a short description is provided in Section IV-B.

The process starts when the female patient is examined by an outpatient physician, who decides whether she is healthy or needs to undertake an additional examination. (...) In the latter case, an examination and follow-up treatment order is placed by the physician. (...) A delegate of the physician arranges an appointment of the patient with one of the wards. The latter is then responsible for taking a sample to be analyzed in the lab later. (...) After receiving the sample, a physician of the lab validates its state and decides whether the sample can be used for analysis or whether it is contaminated and a new sample is required. (...) Finally, a physician from the outpatient department makes the diagnosis and prescribes the therapy for the patient.

Fig. 4. Fragment of a textual description for a patient examination process.

B. Process Modeling Notations

Process models can be created using a variety of modeling languages, such as Petri nets, Event-Driven Process Chains (EPCs), and BPMN. Although we focus in BPMN, the contributions of this paper are independent of the specific notation used to define a process model.

In particular, we focus on BPMN 2.0, a notation created as standard for business process modeling. BPMN has three different kinds of elements. First, the main elements are the nodes in the diagram, which may belong to three different types: *Activities* (Fig. 6, a), which represents some task that is performed; *Events* (Fig. 6, b), which represen that something happens; and *Gateways* (Fig. 6, c), which split or join flow control according to their type: parallel tasks are defined

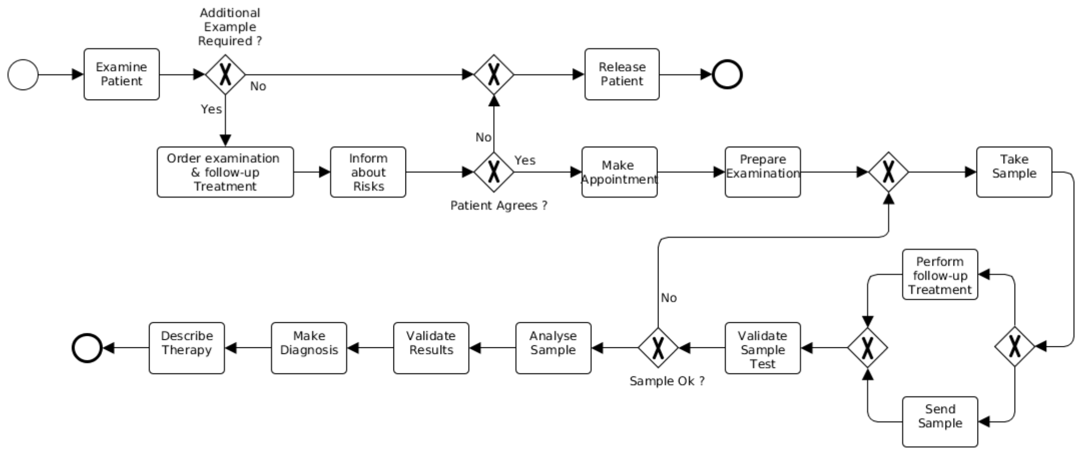


Fig. 5. Process model for the patient examination process.

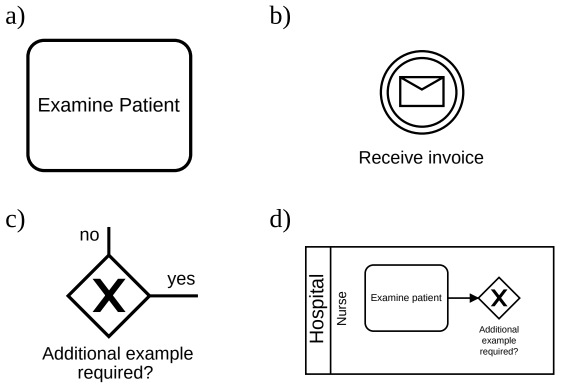


Fig. 6. Different constructs in the BPMN notation. Activity (a), Event (b), Gateway (c), Pool and Lane (d).

through the + gateway, whilst alternative paths are defined through the x gateway. Second, the notation has different edges to connect nodes, so that control-flow dependencies can be defined. A solid line indicates the process workflow, while dashed lines represent messages sent between process participants. Finally, there are organization elements such as *lanes* that contain activities performed by the same participant, and *pools* (Fig. 6, d), that group several related lanes. Fig. 5 shows an example of the BPMN model corresponding to the patient examination process described in Fig. 4. Note that, for simplicity, in the process model of Fig. 5 we do not show pools or lanes.

C. Natural Language Processing and Annotation

In order to compare textual descriptions (like the one shown in Fig. 4) with a formal model, the text must be analyzed and converted to a structured representation that conveys the semantics of the text, and can be compared to the formal model.

To perform this step, we rely on existing tools produced by the NLP community, such as FreeLing [29]. The NLP analyzers are used to obtain a semantic representation of the text. Particularly, we extract the *actions* being described in the text, the *agents* who perform each action, and the *patients* upon which each action is performed. Note that agent and patient terms are borrowed from linguistics, and the latter has no relation to the medical patients described in our running example. For instance, in Fig. 4 the sentence *an examination and follow-up treatment order is placed by the physician*, the *agent* is *physician*, the *action* is *place*, and the *patient* is *order*.

Additionally, we also extract per-word information such as the part of speech (PoS) tag and its ontological sense. By collecting this lower-level information, we allow our technique to fall back to the word level whenever the predicate level description is ambiguous or incomplete (e.g., detecting that word *place* occurs as a *verb* in a sentence is useful to discover that such action is being described, even if the information about the agent or the patient could not be extracted by the NLP tools).

More specifically, we apply a NLP pipeline consisting of tokenization, sentence splitting, morphological analysis, part-of-speech tagging, named entity recognition, word sense disambiguation, dependency parsing, semantic role labelling, and coreference resolution.

The steps up to word sense disambiguation provide word-level information such as the PoS (e.g., *placed* $\rightarrow$ *verb past participle*), the lemma (e.g. *placed* $\rightarrow$ *place*), or the sense (an entry in an ontology that will link to synonyms—e.g., *arrange for*— or other semantically close words—e.g., *request*, *order*, *ask for*).

The three last steps are of special relevance, since they allow the top-level predicate construction, and the identification of actors throughout the whole text: Dependency parsing identifies syntactic subjects and objects (which may vary depending, e.g., on whether the sentence is active or passive), while semantic role labeling identifies semantic relations (the *agent* of an action is the same regardless of whether the sentence is active or passive). Coreference resolution identifies several mentions of the same actor as referring to the same entity (e.g., in Fig. 4, *a delegate of the physician* and *the latter* refer to the same person, as well as the same object is mentioned as *the sample requested* and *it*).

To compare formal models to textual descriptions, we will need annotated versions of those texts (i.e., enriched with all the linguistic information described above). Ideally, those annotations would be automatically performed by NLP tools, but given the limitations of the current NLP technology, we will resort to a certain amount of human annotation to improve the quality of our semantic representations. More details are provided in Section V-A.

Creating annotated versions of texts is usual in the NLP field, where many approaches are based on machine learning and require annotated text corpora both for training and for evaluating the performance of the developed systems. This need for annotated text has led the NLP community to develop several general-purpose annotation tools (e.g., Brat [30]). However, inspired on those resources, we have developed our own annotation tool tailored to *Model Judge*'s use case in order to offer a better user experience. This tool is also able to automatically infer partial annotations, as seen later in Section V-A.

V. Framework Underlying the Model Judge

By observing the grading process of several process modeling courses, we have established a set of diagnostics that are suitable for being computed automatically. As done in the literature [2], we have split these diagnostics in three different categories: *Syntactic* diagnostics consider the control flow of the model. *Pragmatic* diagnostics consider the interpretability aspects of the model. Finally, *semantic* diagnostics check if there is no missing information from the underlying process and all the information provided is relevant and valid. Note that we use the terms *syntactic*, *semantic*, and *pragmatic* as it is usually done in the Process Modeling field, which differs from the meaning these terms have in linguistics and NLP.

This section describes our proposed framework for the automatic computation of diagnostics. The *Model Judge* produces a list of recommendations that the student can generate upon demand during the modeling process. Fig. 7 shows an overview diagram. The inputs to the system are the student's *Process Model* and the *Textual Description*, in natural language, that is being used as the exercise's statement.

The textual description is analyzed by the *Text Annotation* module, to produce a structured annotated description where the relevant concepts in the process model are identified. This annotation is then completed with human intervention to include domain expertise not available to NLP tools. After these steps—performed only once when the instructor creates a new exercise—students can work with the exercise.

In order to produce the diagnostics for a student-built model, the *Alignment* between the student process model elements—such as activities or gateways—and the annotations in the text is computed. Then, this alignment is used to compare the process model with the textual description in the *Diagnostics* module. In the remainder of this section we describe the aforementioned modules in detail.

A. Formalizing Annotated Textual Descriptions

Using natural language alone for the formal description of process models has some drawbacks. On one hand, the lack of structure in natural language text makes automatic analysis more difficult. On the other hand, the inherent ambiguity of written text makes it difficult to establish which parts of the text are relevant to describe the process.

NLP tools are getting increasingly better at extracting structure out of plain text. However, some ambiguities remain an open issue to this day. Furthermore, we argue that some of these issues cannot be automatically solved reliably enough, since they require extensive domain knowledge. For instance, in the example from Fig. 4, the sentence *"The latter [physician] is then responsible for taking a sample to be analysed in the lab later"* highlights two important sources of ambiguity in the analysis of textual descriptions: *(i)* The action *analyse* is described, but it may not happen until later into the process, as hinted by the word *later*. Extracting this kind of temporal relationships from textual descriptions is a very complex task [31], with no good solutions in the context of business processes. *(ii)* Depending on the purpose of the process, the same *analyze* verb could actually be irrelevant from an organizational point of view, if the *lab* were to be considered an external entity. This presents an issue that cannot be solved without deep understanding of the organization. This is why in Fig. 7 we incorporate a stage on human annotation, so that these ambiguities can be resolved.

In order to bridge the gap between the textual descriptions and the formal process models, we introduce the concept of *Annotated Textual Descriptions* (ATD). The goal of this representation is to use text annotations, typically found in the context of textual corpus labeling, to define a structured representation of text that still benefits from the flexibility and interpretability offered by natural language. Annotations are used as an intermediate language, and are generated in two steps: First, an automatic NLP tool generates a candidate annotation. Afterwards, the annotations are refined by a human (in our case, the instructor) to refine the annotation, e.g., discard unnecessary labels, or add missing or relevant details using domain expertise.

Even though obtaining accurate annotations requires some human intervention, our e-learning framework for self-assessment still enables for automation, since the annotations only need to be made once for an exercise, and from that moment on, they can be used to correct any number of proposed models for that exercise.

Formally, we can define an ATD as a $\langle T, A, R\rangle$ tuple, where

T is a string of characters representing the textual description.

A is a set of *annotations*. Each annotation $\alpha = \langle type, start, end\rangle$ marks a relevant aspect of the business process. The $start$ and end integers mark the positions of a substring of T, and the $type$ is used to add semantics to the annotation.

R is a set of triples $\langle t, \alpha_i, \alpha_j\rangle$ representing binary relations of type t between pairs of annotations α_i, α_j

Next, we describe each of the annotation and relation types

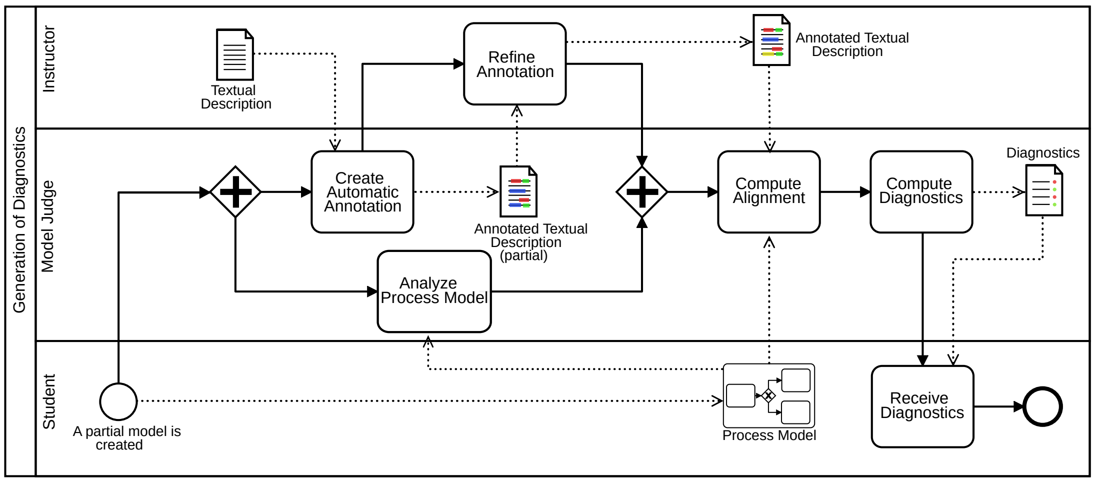


Fig. 7. Overview of the *Model Judge* framework to support the creation of process models.

that we consider for ATDs. Table I shows an example of an ATD which we will use for the remainder of this section.

1) Annotations: Annotations are used to mark an important region of the text. In the case of ATDs we consider the following types of annotations which directly map to some well-known concepts in the context of business processes.

Action. The *action* annotation is used to represent the steps of the business process model, which are often associated with verbs. An example action would be *placing* (α_1) an examination order. Each action corresponds to a single activity in the process model. We additionally allow to specify an action as optional. The *optional action* annotation is used to indicate that the associated action could be elided from the process description without a substantial change in its semantics. For instance, in the sentence *"the patient can leave"*, the action of *leaving* (α_8) could be considered as part of the process. However, that action does not add a substantial amount of semantic value, and thus can be considered *optional*. In our framework, optional actions can be used by the instructor to allow for a more flexible answer to the exercise, since both the solution that contains the action and the one that does not can be considered correct.

Role. The role annotation is used to represent the autonomous actors involved in the process, like the *outpatient physician* (α_5). Any mention of an entity that performs some Action is considered a Role. Roles are associated with swimlanes in business processes (see Fig. 6 (d)).

Business Object. The business object annotation is used to mark all the relevant elements of the process that do not take an active part in it, such as an *examination order* (α_6). In the business process, correspondences with business objects are typically found in activity labels and explicit elements such as data object references.

Condition. A condition annotation indicates a part of the text that represents a pre-requisite for some part of the process being executed, such as the patient *being healthy* (α_7). Conditions are typically associated with exclusive gateways, their labels and their surroundings.

2) Relations: Relations represent binary relationships between annotations. In ATDs, we consider the following relations. Note that the terms *agent* and *patient* have been chosen to highlight that the relations are consistent with the linguistic definition of the sentence semantic roles, as typically seen in linguistics (see Section IV).

Agent. A *role* is the *agent* of an *action* whenever the entity represented by the role performs that action. For instance, relation r_1 tells us it is the *physician* who *places* something.

Patient. A *role*, or *business object* is the *patient* of an *action* whenever it acts as the recipient of the action. Relation r_2 tells us that what is *placed* is an *examination order* something.

Control Flow. The *sequential*, *exclusive* and *parallel* binary relationships, which are borrowed from *behavioral profiles* [32], are used to indicate the order between *action*s in the textual description. Due to the characteristics of natural language text, there is an open world assumption on the set of control flow relationships: Assuming an absence of contradictions, everything that is stated as relationship is enforced. However, no assumptions are made on things that are not specified. In our example, *analyse* (α_2) is *sequential* (r_3) to—i.e., happens before—*validates* (α_3)

Coreferences. A *role* is a coreference of another *role* when they refer to the same entity. The coreference relation forms a graph with one connected component per process entity. All ocurrences of the "patient" role in the example text are coreferences. However, there are two different "physician" roles in the text, the "outpatient physician" (α_4, α_5) and the "physician of the lab" (α_{14}), which form two disconnected coreference graphs.

B. Alignment of Models and Annotated Texts

The goal of aligning a process model and a textual process description is to identify correspondences between the elements of the process model and parts of the textual process description [12], [33], [34]. In particular, all elements that

	Type	Text Fragment
α_1	*action*	"an examination and follow-up treatment order is **placed** by the physician"
α_2	*action*	"is then responsible for taking a sample to be **analysed** in the lab"
α_3	*action*	"After receiving the sample, a physician of the lab **validates** its state"
α_4	*role*	"an examination and follow-up treatment order is placed by the **physician**"
α_5	*role*	"the female patient is examined by an **outpatient physician**"
α_6	*object*	"an **examination and follow-up treatment order** is placed by the physician"
α_7	*condition*	"decides whether **she is healthy** or needs to undertake additional examination"
α_8	*optional, ending*	"the patient can **leave**"
α_9	*implicit*	"The patient is **asked** to sign an informed consent" (Added text)
α_{10}	*action*	"The required examination and sampling is **prepared** by a nurse"
α_{11}	*action*	"The required examination and sampling is **prepared** by a nurse"
α_{12}	*object*	"The required **examination** and sampling is prepared by a nurse"
α_{13}	*object*	"The required examination and **sampling** is prepared by a nurse"
α_{14}	*role*	"After receiving the sample, a **physician** of the lab validates its state"

	Type	α_i	α_j
r_1	*agent*	α_1	α_4
r_2	*patient*	α_1	α_6
r_3	*sequential*	α_2	α_3
r_4	*coreference*	α_4	α_5
r_5	*patient*	α_{10}	α_{12}
r_6	*patient*	α_{11}	α_{13}
r_7	*fusionable*	α_{10}	α_{11}

TABLE I
EXAMPLE FRAGMENT OF ANNOTATED TEXTUAL DESCRIPTION WITH ANNOTATIONS (LEFT) AND RELATIONS (RIGHT)

describe the flow in a process model, i.e., *activities*, *events*, and *gateways*, are aligned to (parts of) the sentences of a textual process description. For instance, the sentence containing the text *"... a delegate of the physician arranges an appointment of the patient with one of the wards."* would be aligned with the task "Make Appointment" of the BPMN model. Similarly, the gateway with the decision *"Patient Agrees ?"* must be mapped to the sentence *If the patient signs an informed consent ..."*. This previous example clearly shows the challenge of computing alignments when textual descriptions are not annotated. Fig. 8 shows an example alignment for an initial fragment of our running example.

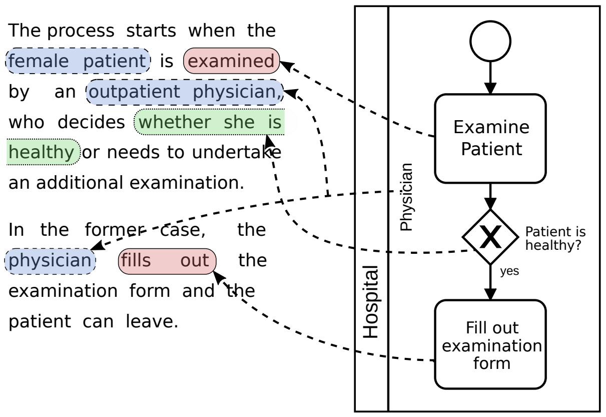

The process starts when the female patient is examined by an outpatient physician, who decides whether she is healthy or needs to undertake an additional examination.
In the former case, the physician fills out the examination form and the patient can leave.
Hospital
Physician
Examine Patient
Patient is healthy?
yes
Fill out examination form

Fig. 8. Alignment between annotated textual description and BPMN process model.

In *Model Judge*, establishing the correspondence between the annotations in the annotated textual description and the elements in the business process model is necessary in order to compute some diagnostics, such as the semantic-related ones. The goal of this computation is to be able to assess which parts of the process model have been correctly modeled by the student, according to the textual description.

In this paper we extend our alignment framework presented in earlier works [12], [34] by assuming that textual descriptions are annotated. This assumption allows strengthening some of the constraints in the algorithm by having the certainty that information that comes from annotated texts is precise and complete.

An overview of the details behind of our alignment technique can be seen in Fig. 9: *(i)* First, linguistic features from the textual description and the process model are extracted to map both representations into a canonical form in a *Feature Extraction* step. During this step we extract high-level information such as word senses (using WordNet) and semantic roles. In order to do this, we apply the full NLP pipeline described in Section IV-C. *(ii)* We can then use well-known similarity functions such as the Jaccard Index or Cosine Similarity to compare the feature vectors in the *Similarity Computation* phase. *(iii)* After that, the similarity information is encoded in an optimization algorithm, implemented as an Integer Linear Program, to find a valid alignment which maximizes similarity. *(iv)* Finally, in order to detect unnecessary activities, i.e., activities that are present in model but not in the text, we use *predictors*, adapted from Van der Aa *et al.* [33].

We refer the interested reader to the aforementioned work for a detailed description of the techniques.

C. Automatic Generation of Diagnostics

The core of our approach consists on the automatic generation of diagnostics. This computation uses the information from the *annotated textual description*, the *process model*, and the *alignment* between them. In this section we detail which diagnostics are currently generated by our tool and how they are computed.

1) Syntactic Diagnostics: A good process model should have a clear and unambiguous control flow. Syntactic diagnostics identify common patterns that typically result in less understandable and maintainable process models. Illustrative examples for the presented syntactic errors can be found in Fig. 10.

Gateway Reuse and Implicit Gateways. (Fig. 10, a and b) Gateway reuse refers to a gateway that acts both as a *split* (more than two ouptuts) and a *join* (more than two inputs). Implicit Gateways exist when an activity has multiple input or output flows. The semantics for these two constructs are not clear and can lead to hidden modeling errors. Because of that, avoiding gateway reuse

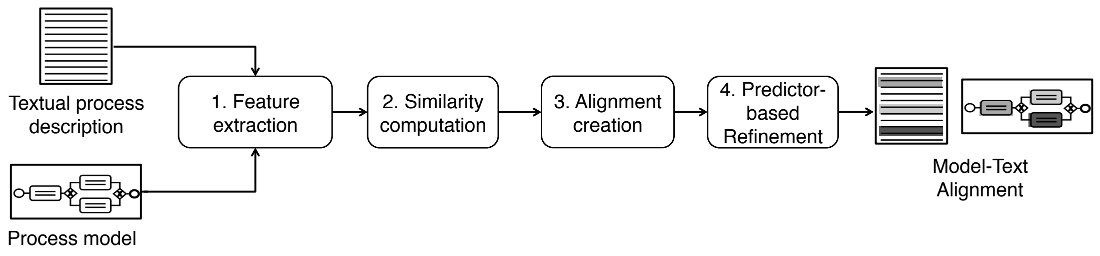

Textual process description
Process model
1. Feature extraction
2. Similarity computation
3. Alignment creation
4. Predictor-based Refinement
Model-Text Alignment

Fig. 9. Overview of alignment between textual descriptions and process models.

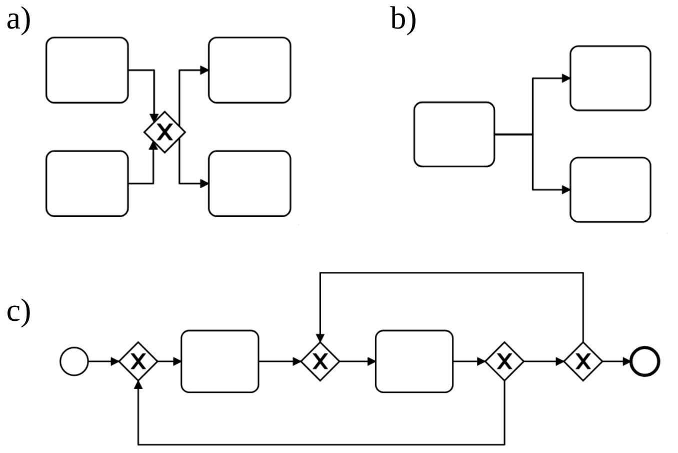


Fig. 10. Examples of several syntactic errors: Gateway Reuse (a), Implicit Gateway (b), Non-natural Loop (c).

and implicit gateways is a well-known best practice in business process models [6], [20], [35].

Non-Natural Loops. (Fig. 10, c) Due to their similarity, some desirable properties of program's control flow graphs are also relevant in the context of process model diagrams. Ideally, process models should contain only *Natural Loops*. That is, there is only a single way to enter the loop. We have observed non-natural loops are a common pattern among novice students.

Soundness. A well-known desirable property of process models is *soundness* [36], which guarantees the process model is free from livelocks, deadlocks, and other anomalies that can be detected without domain knowledge.

Notice that the last two syntactic diagnostics can also be applied to other languages, like Petri nets. The computation of syntactic diagnostics only requires the process model, and they can be implemented use well-known algorithms. Particularly, checking the use of implicit gateways or gateway reuse consists of a very simple structural check. On the other hand, the presence of non-natural loops can be determined by checking for cycles in the process graph after removing all its *back edges*, computed from the *dominator tree* [37].

2) Pragmatic Diagnostics: Pragmatics in process modeling can be described as the interpretability of the model, which can be impacted by factors such as complexity, modularity, secondary notation or activity labelling style [38]. It is that last aspect of pragmatics that *Model Judge* is focused on. Process model diagrams define a large portion of their semantics using natural language. It is desirable to restrict the language to a strict writing style (e.g, the *verb-object* rule in Mendling *et al.* [20], G6), in order to avoid ambiguous phrasing [39]. A simple and strict style is also important when considering automatic analysis of the process model language. For example, while it is acceptable in the text to include the sentence: *"The latter is then responsible for taking a sample to be analysed in the lab later"*, having that sentence as an activity label in the process model adds unnecessary complexity, since the aim is to have label text to be as simple as possible, and not contain details about the control flow of the process. A simpler model label using the *verb-object* rule for the aforementioned sentence would be *"Analyse sample"*.

Previous studies [39]–[42] have established the common structures in label descriptions. However, these structures are not always followed by novice students. Because of that, an automatic detection of invalid writing styles is beneficial for student self-improvement.

The approach we present for checking adherence to a writing style is based on using a custom NLP parser for labels. The core of this technique consists of parsing the labels with a custom-made context-free grammar. This grammar is able to recognise most label syntactic structures defined by Pittke *et al.* and Leopold *et al.* [41], [42], and is built in such a way that the root node of the parse tree will be labeled with the identified writing style. For example, a label written in the *noun-action* style will have this information reflected at the root node of the tree. Parsing is then done as follows:

1) The n most likely part-of-speech (PoS) sequences of the label are computed.
2) For each of the part-of-speech sequences, the label is parsed using a *chart parser* with the aforementioned context-free grammar.
3) If all the parse tree roots are found to have an undesired style or none of the parse trees could be parsed by the grammar, the label writing style is not correct.

For instance, let us consider the activity label *sample patient*. The possible PoS sequences are $\langle noun, noun\rangle$ (i.e., a sample patient) and $\langle verb, noun\rangle$ (i.e., to sample a patient). After trying to parse the text considering each sequence, we see that the most likely sequence, $\langle noun, noun\rangle$, does not match any valid label pattern. Thus, the second sequence is chosen, which matches one of the valid *action* patterns. We can then say that the label's writing style is correct, while if no valid pattern is found in any of the possible sequences, the label is considered to have an incorrect style.

In our implementation, patterns of labels describing double actions are also detected (e.g. *Close Ticket and Inform the Manager*) and, in this case, the system suggests the student to create two separate activities. It is worth noticing that this technique can be adapted so any instructor can enforce a label writing style in a flexible way, just defining a simple grammar to detect label patterns, and accept or reject the labels depending on the matched pattern.

Furthermore, this analysis helps avoiding common issues with general-purpose NLP tools when faced with the kind of text in business process models. For example, a general parser lacks the necessary context to infer that activity labels should always describe an action. As we can see with *sample patient*, adding knowledge about the context of a sentence into the parser solves the issue.

3) Semantic Diagnostics: A process model diagram has to communicate the semantics of the underlying process in a clear and unambiguous way. All the information provided has to be correct, and in the right order. On the other hand, unnecessary information introduces noise that can generate confusion. Semantic diagnostics help enforcing these properties on the process model.

Missing/Unnecessary activities. This problem arises when a relevant activity of the process is omitted from the process model or, symmetrically, when additional activities are added which are either wrong or add no relevant information. This can be caused by an oversight or a poor understanding of the process being modeled.

Missing/Unnecessary Roles. When process models use role information, such as swimlanes in BPMN, the same diagnostics of *missing* and *unnecessary* roles can be applied as well, to ensure all the relevant actors are properly modeled.

Control-Flow Consistency. All the information in the process model should be consistent with the control flow of the process being described. For example, if the text states that an activity A must happen before an activity B can be executed, it is incorrect to model them as two separate branches of a choice. If a temporal relation described in the text is not accurately incorporated in the process model, then a control-flow consistency violation would be communicated to the novice modeler.

Semantic checks are concerned with the underlying process semantics. Computing these diagnostics requires all the information from the annotated textual description, the process model and the alignment.

To detect that an activity is missing from the process model, the alignment information is used. Particularly, if there is an action in the ATD with no correspondences from the process model, the activity is missing from the process model. The system can then inform the modeler by generating a detailed error message using the annotated data from the textual description.

Detecting unnecessary activities also relies on the alignment. In this case, there will be a process model activity aligned to some text action. If the similarity between them is low enough the activity is considered unnecessary, as there is no good match for the activity in the text.

Coverage of roles is performed similarly to activities. In this case, the similarity function is used to assess the similarity between *role* annotations and the process model's swimlanes.

Control flow validations have a preliminary implementation using the predictor technique described in the aforementioned prior work [12], [33]. However, there is still room for substantial improvements in this regard.

VI. Evaluation

The goal of this section is to provide an evaluation of the *Model Judge*. More specifically, we aim at investigating the following questions:

RQ1. Is the *Model Judge* perceived useful and accurate by the students? This question is answered in Section VI-A;

RQ2. How are the *Model Judge* diagnostics and validations associated with actual modeling errors? This question is answered in Section VI-B1;

RQ3. How does the process of process modeling evolve over time, when the *Model Judge* is available? This question is answered in Section VI-B2.

To answer the above research questions, data from a modeling session at the Technical University of Denmark (DTU) was analyzed. During the modeling session 26 students were asked to individually create a process model given a textual description using the *Model Judge* (cf. Section II). There was no time limit imposed. Students had a validation functionality available during model creation. The diagnostics generated by the validation functionality provided some indication about the type of error (e.g., missing activity), but did not reveal the exact source of the error. Moreover, *Model Judge* provides a complete validation functionality offering more detailed diagnostic feedback. Students were instructed to use the complete validation only at the end of the modeling process, once they have completed the modeling. This way, during the modeling students were incentivated to find out the reasons why certain types of errors arised, without actually knowing the exact errors. We believe this facilitates the learning process. It is important to note that in our experimental setting the conditions for having a control group were not met.

A. Subjective Perception (RQ1)

To answer **RQ1** a survey was conducted at the end of the modeling session. Students were asked to assess the validation and complete validation functionalities of *Model Judge* in terms of perceived *accuracy* and *usefulness*. Moreover, students were asked to write down any complaints and/or improvement suggestions concerning *Model Judge*. Overall, 18 out of 26 students participated in the survey.

Fig. 11 shows the results of the survey: Our results demonstrate that 67% (12 out of 18) found the validation functionality useful (i.e., strongly agree or agree). Similarly, 67% (12 out of 18) perceived the complete validation useful.

In terms of accuracy, 44% (8 out of 18) perceived the validation functionality of the *Model Judge* as accurate (i.e., strongly agree or agree). For the complete validation the agreement was with 56% (10 out of 18) slightly higher.

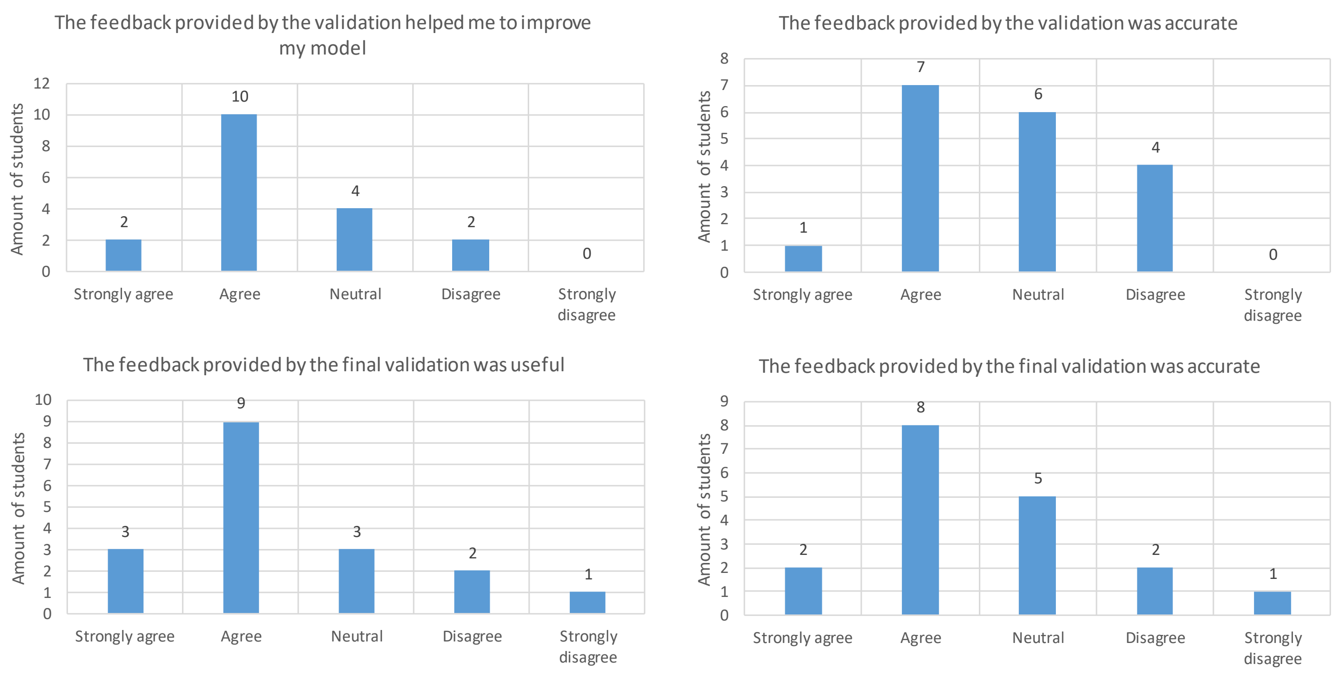


Fig. 11. Results of the student survey for the modeling courses. Each chart reports the results referring to a different question asked.

Additionally, it can be noted that the perceptions concerning usefulness were better than for accuracy. This is supported by the written feedback provided by the students. While in general, students appreciated the support provided by *Model Judge*, some students pointed out that the tool in did not recognize all their process model labels. In addition, some students stated that a more precise feedback would have been useful (e.g., *"I disliked the feedback* [provided by the validation functionality]. *I could not know what was wrong, just that something was."*.

Moreover, it can be noted that the complete validation functionality is perceived as slightly more useful and accurate when compared to the validation functionality. This is not surprising, since the complete validation functionality provided a more detailed feedback to the students.

In general, we can answer research question **RQ1** partially positively, i.e., *Model Judge* is perceived by the majority of the students as useful but, concerning accuracy, the majority agrees that just the complete validation is accurate.

B. Analysis of the Modeling Session Data (RQ2, RQ3)

To answer research question **RQ2** and **RQ3** we analyzed the recordings of the modeling session.

Data Collection. For every student, we stored periodically (every minute) information for the whole modeling session. Additionally, information was also saved each time the user performed a simple or complete validation. In particular, we recorded a total of 1584 intermediate models for 26 students. For the snapshots, we stored: *(i)* A unique user identifier; *(ii)* The process model in BPMN (XML) format; *(iii)* The timestamp of the snapshot; *(iv)* The type of information: automatic, validation or complete validation; and *(v)* The validation results of our tool for the particular process model. Note that the validation results were computed for all snapshots, despite the fact that the students only saw the ones they explicitly requested.

The dataset used to answer RQ2 and RQ3 is available online at the PADS-UPC research group website [43].

1) Association of the Tool on Diagnostics with Modeling Errors (RQ2): In this section, we provide an answer to **RQ2**: we analyze the association between diagnostics of the *Model Judge* and actual modeling errors.

The actual modeling errors were derived based on the evaluation criteria agreed by two researchers, a Ph.D student and an associate professor affiliated to different institutions. Both assessors have teaching experience in business process management: The PhD student has contributed in the teaching of the course for 2 consecutive years, while the associate professor has been teaching the course for the past 7 years. Additionally, both researchers are familiar with evaluating process models derived by students. The criteria were set based on the guidelines provided by the SEQUAL [44] and 7PMG [20] frameworks. All the covered criteria have been discussed by both assessors before proceeding with the evaluation of the models.

We analyzed to what extent the provided diagnostics (identified by the *Model Judge*) are associated with modeling errors (identified by manual inspections). In terms of model diagnostics we considered: D_{avg} = "Average number of bad diagnostics during the modeling session," D_{end} = "Number of bad diagnostics at the end of the exercise" and in term of modeling errors we considered: E_m = "Errors as missing activities," E_c = "Errors in the organization of the control-flow," and E_t = "Errors in the model semantics."

Results, expressed as Spearman's correlation tests, between diagnostics and actual errors are reported in Table II. The

		D_{avg}	D_{end}
E_m	Spearman's ρ	**.502**	**.576**
	p-value	.009	.002
E_c	Spearman's ρ	**.476**	**.569**
	p-value	.014	.002
E_t	Spearman's ρ	**.445**	**.520**
	p-value	.023	.007

TABLE II
SPEARMAN'S CORRELATION TEST RESULTS BETWEEN DIAGNOSTICS AND ACTUAL ERRORS

		D_{avg}	D_{end}
V_s	Spearman's ρ	**-.724**	**-.763**
	p-value	$< .001$	$< .001$
V_c	Spearman's ρ	**-.607**	**-.687**
	p-value	$< .001$	$< .001$

TABLE III
SPEARMAN'S CORRELATION TEST RESULTS BETWEEN DIAGNOSTICS AND NUMBER OF VALIDATIONS

correlation between the number of bad diagnostics and the different errors are moderate and all significant. This suggests that the *Model Judge*'s diagnostics is capable of approximating to a certain extent the actual modeling errors.

Starting from the previous results, we additionally studied the presence of correlations between the number of validations performed by the students and the number of bad diagnostics obtained. To that end, we computed a Spearman's correlations between V_s = "Number of validations," V_c = "Number of **complete** validations," D_{avg} (average bad diagnostics), D_{end} (bad diagnostics by the end). Besides the obvious correlations $V_c \sim V_s$ and $D_{avg} \sim D_{end}$, we also found significant and strong negative correlations between the two pairs of variables, as seen in Table III. This means that, as the number of validations grows, the number of bad diagnostics decreases.

In another analysis, we correlated V_s (i.e., number of validations) and V_c (i.e., number of complete validations) with the number of modeling errors observed by the end: E_m (errors as missing activities), E_c (errors in the organization of the control-flow), E_t (total semantics mistakes). The results are reported in Table IV and, as for the previous cases, all correlations are significant as well as negative. This means that as the number of validations grows the number of actual errors decreases. Based on previous results, this should not surprise: we already observed that diagnostics inversely correlate with

		V_s	V_c
E_m	Spearman's ρ	**-.487**	**-.430**
	p-value	.012	.028
E_c	Spearman's ρ	**-.576**	**-.547**
	p-value	.002	.004
E_t	Spearman's ρ	**-.457**	**-.441**
	p-value	.019	.024

TABLE IV
SPEARMAN'S CORRELATION TEST RESULTS BETWEEN ERRORS AND NUMBER OF VALIDATIONS

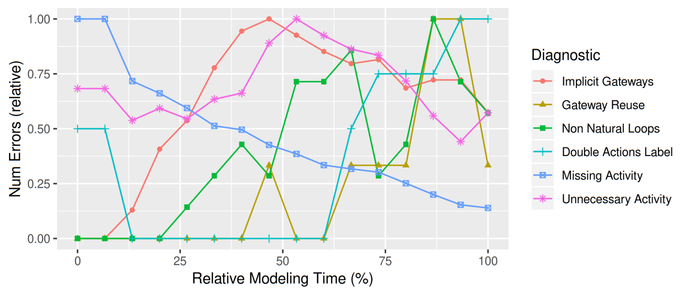


Fig. 12. Evolution of the diagnostic types for the modeling session.

the number of validations in (see Table III) and that diagnostics and errors positively correlate (see Table II).

Finally, to answer research question **RQ2**, we can conclude that both the diagnostics and the validation capabilities of the *Model Judge* are associated with actual modeling errors. In particular, there are correlations between *Model Judge*'s diagnostics and actual errors. We also observed negative correlations between the number of validations and diagnostics and between number of validation and actual errors (which mean that an increase in the number of validations is associated with less errors).

2) Evolution of the Modeling Sessions (RQ3): The final investigation we performed aimed at understanding how the process modeling evolves when the *Model Judge* is available (**RQ3**). For this, we analyzed after the modeling course ended the collected snapshots with the goal to observe the evolution of the number of validation errors during the modeling sessions. Note that, for this analysis, we considered individual modeling sessions, with some students having performed more than one session.

In a first investigation, we analyzed how the frequency of different diagnostics varies during the sessions. Fig. 12 shows the evolution of the average amount of diagnostics for all students, per diagnostic type. To better observe the relative behavior of each type, regardless of the amount of diagnostics in the category, we plot the values relative to the maximum of each category. We have encountered substantial differences between diagnostic types. "Missing Activity" diagnostics decrease as the session advances, since less activities will be missing as the modeling session progresses. "Unnecessary Activity" and "Implicit Gateway" increase for the first half of the session, then decrease. This behavior is consistent with the fact that most students do not start using the complete validation feature until the second half of the session. The more detailed feedback of the complete validation then helps them finding the more subtle errors in their process model. The remaining diagnostic types have an oscillatory behavior, but still increase for the duration of the session. This can be explained by the fact that, as the modeling session progresses, there is a greater chance of a student introducing an error leading to one of these diagnostics. However, the drops after 75% progress could indicate that some students delay the correction of these errors until the end of the modeling session. This is in line with existing research that modelers differ in

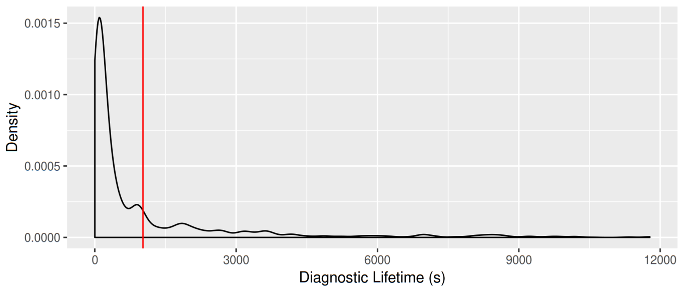


Fig. 13. Density plot of the diagnostic lifetimes variable.

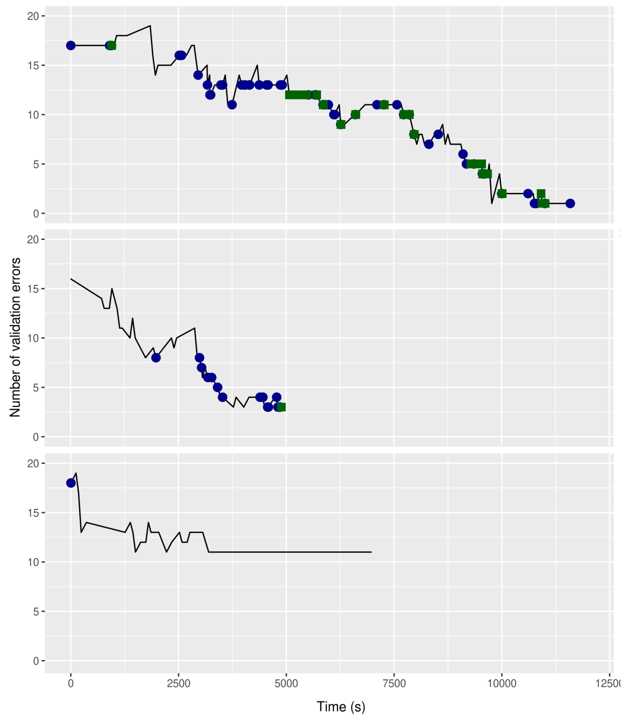


Fig. 14. Three characteristic behaviors observed in the modeling sessions. The blue circles represent simple validations, while green squares denote complete validations.

term of their validation behavior [45].

To get a deeper insight into the modeling session data, we computed the lifetime of the diagnostics given to the students. We define the diagnostic lifetime as the elapsed time between the moment a student introduces a mistake in the model, and the moment that mistake is corrected. Note that this metric is independent of the validations made by the student, since diagnostics are computed for all snapshots regardless of the student used the validation function or not. The average lifetimes follow a long-tailed distribution (Fig. 13). That is, in the average case mistakes are quickly corrected by the students. However, for a few cases, it can take a very long time to solve those mistakes.

Finally, by manual inspecting the session data, we identified some "modeling profiles," i.e., typical approaches followed by students while solving their modeling task. Fig. 14 shows a representative for each of the identified profiles, when plotting number of validation errors vs. time (in seconds) together with simple and complete validations. Note that there are a few outlier sessions that do not match any of the three profiles shown. An instructor can obtain a good overview of the students evolution by looking at these student's plots. These are the three profiles we identified: *(i)* The first group (on top of Fig. 14) is composed of modeling sessions where students frequently use the intermediate and complete validation functions, and ended up with almost no bad diagnostics. This group corresponds to 26.0% of the sessions; *(ii)* The modeling sessions from the second group (in the middle of Fig. 14) correspond to students who frequently use the validation, however, they only check the complete validation at the end of the session. The final amount of bad diagnostics for this group is comparable to the previous one. This group corresponds to 59.3% of the students; *(iii)* The modeling sessions from the third group (at the bottom of Fig. 14) correspond to students who started working on the exercise but finished before fixing the majority of bad diagnostics. We have observed that the students in this group performed substantially less validations. This group corresponds to 14.8% of the students.

In this section we answered **RQ3** by examining how the modeling sessions evolve when the *Model Judge* is used: we observed the distribution of the diagnostics as well as the density plot of the diagnostic lifetimes. Additionally, we identified three typical modeling profiles that can be used by instructors to gather some initial understanding on the modeling approach followed by students.

VII. Conclusion

In this paper we provide both the framework and an evaluation of the *Model Judge*. The framework is grounded in the use of NLP techniques together with an algorithm to align textual descriptions and graphical process models notations such as BPMN.

Our experience of applying *Model Judge* in different universities shows that it can be easily incorporated in a modeling course, where novice modelers can benefit from an environment that produces continuous support in the task of creating a process model. As for instructors, they are able to better support students by monitoring modeling sessions and easily create new exercises to fit their needs.

As future work we plan to extend the capabilities of the framework (multilingual support, expand the type and quality of diagnostics, among others), and apply it in more courses so that more conclusions can be drawn on the data gathered.

Acknowledgment

This work has been partially supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018, doi: 10.1007/978-3-662-56509-4.

[2] J. Krogstie, G. Sindre, and H. D. Jørgensen, "Process models representing knowledge for action: a revised quality framework," *EJIS*, vol. 15, no. 1, pp. 91–102, Dec. 2006, doi: 10.1057/palgrave.ejis.3000598.

[3] O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *IEEE Softw.*, vol. 11, no. 2, pp. 42–49, March 1994.

[4] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, ser. Lecture Notes Business Inf. Process. Springer, 2008, vol. 6, doi: 10.1007/978-3-540-89224-3.

[5] M. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, and D. Edmond, "Business process verification – finally a reality!" *Bus. Process Manage. J.*, vol. 15, no. 1, pp. 74–92, Feb. 2009, doi: 10.1108/14637150910931479.

[6] C. Haisjackl, P. Soffer, S. Y. Lim, and B. Weber, "How do humans inspect BPMN models: an exploratory study," *Softw. & Syst. Model.*, vol. 17, no. 2, pp. 655–673, May 2018, doi: 10.1007/s10270-016-0563-8.

[7] A. Kurnia, A. Lim, and B. Cheang, "Online judge," *Comput. & Educ.*, vol. 36, no. 4, pp. 299–315, May 2001, doi: 10.1016/S0360-1315(01)00018-5.

[8] M. Joy, N. Griffiths, and R. Boyatt, "The boss online submission and assessment system," *ACM J. Educational Resour. Comput.*, vol. 5, no. 3, p. 2, Sep. 2005, doi: 10.1145/1163405.1163407.

[9] A. Kosowski, M. Malafiejski, and T. Noinski, "Application of an online judge & contester system in academic tuition," in *Advances Web Based Learning - ICWL 2007, 6th Int. Conf., Edinburgh, UK, August 15-17, 2007*, pp. 343–354.

[10] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *ACM SIGPLAN Conf. Programming Language Design Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pp. 15–26.

[11] J. Petit, S. Roura, J. Carmona, J. Cortadella, A. Duch, O. Giménez, A. Mani, J. Mas, E. Rodríguez-Carbonell, A. Rubio, J. de San Pedro, and D. Venkataramani, "Jutge.org: Characteristics and experiences," *IEEE Trans. on Learn. Technol.*, 2017.

[12] J. Sànchez-Ferreres, H. van der Aa, J. Carmona, and L. Padró, "Aligning textual and model-based process descriptions," *Data Knowl. Eng.*, vol. 118, pp. 25–40, Nov. 2018, doi: 10.1016/j.datak.2018.09.001.

[13] PADS-UPC Research group. PADS-UPC source code repository on github. [Online]. Available: http://github.com/PADS-UPC

[14] J. Pimentel, E. Santos, T. Pereira, D. Ferreira, and J. Castro, "A gamified requirements inspection process for goal models," in *Proc. 33rd Annu. ACM Symp. Applied Computing, Pau, France, April 09-13, 2018*. ACM, pp. 1300–1307.

[15] D. Bogdanova and M. Snoeck, "Camelot: An educational framework for conceptual data modelling," *Inf. Softw. Technol.*, Feb. 2019.

[16] D. Bogdanova, e. R. A. Snoeck, Monique", D. Karagiannis, and M. Kirikova, "Learning from errors: Error-based exercises in domain modelling pedagogy," in *Practice Enterprise Modeling, Vienna, Austria, October 31 - November 2, 2018*. Cham: Springer Int. Publishing, pp. 321–334.

[17] B. Demuth and D. Weigel, "Web based software modeling exercises in large-scale software eng. courses," in *2009 22nd Conf. Software Engineering Education Training, Hyderabad, India, 17-20 Feb. 2009*, pp. 138–141.

[18] H. Störrle, N. Baltsen, H. Christoffersen, and A. M. Maier, "On the impact of diagram layout: How are models actually read?" in *PSRC@ MoDELs, Valencia, Spain, September 28 - October 3, 2014*, pp. 31–35.

[19] A. Burattin, M. Kaiser, M. Neurauter, and B. Weber, "Eye tracking meets the process of process modeling: a visual analytic approach," in *Int. Conf. Business Process Management*. Springer, pp. 461–473.

[20] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7PMG)," *Inf. & Softw. Technol.*, vol. 52, no. 2, pp. 127–136, Feb. 2010, doi: 10.1016/j.infsof.2009.08.004.

[21] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating natural language specifications from UML class diagrams," *Requirements Eng.*, vol. 13, no. 1, pp. 1–18, Sep. 2008.

[22] I. S. Bajwa and M. A. Choudhary, "From natural language software specifications to UML class models," in *Enterprise Informagion Systems - 13th Int. Conf., ICEIS*, Beijing, China, June 2011, pp. 224–237.

[23] H. Leopold, J. Mendling, and A. Polyvyanyy, "Supporting process model validation through natural language generation," *IEEE Trans. Softw. Eng.*, vol. 40, no. 8, pp. 818–840, May 2014.

[24] F. Friedrich, J. Mendling, and F. Puhlmann, "Process model generation from natural language text," in *Adv. Information Systems Engineering - 23rd Int. Conf., CAiSE 2011, Beijing, China, June 8-11, 2011*, London, UK, pp. 482–496.

[25] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, Dec. 2001. [Online]. Available: http://dx.doi.org/10.1007/s007780100057

[26] U. Cayoglu, R. Dijkman, M. Dumas, P. Fettke, L. García-Bañuelos, P. Hake, C. Klinkmüller, H. Leopold, A. Ludwig5, P. Loos, J. Mendling, A. Oberweis, A. Schoknecht, E. Sheetrit, T. Thaler, M. Ullrich1, I. Weber, and M. Weidlich, "Report: The process model matching contest 2013," in *Business Process Management Workshops - BPM Int. Workshops, Beijing, China, August 26, 2013*, Beijing, China, pp. 442–463. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-06257-0_35

[27] C. Cabanillas, D. Knuplesch, M. Resinas, M. Reichert, J. Mendling, and A. R. Cortés, "Ralph: A graphical notation for resource assignments in business processes," in *Adv. Information Systems Engineering - 27th Int. Conf., CAiSE 2015, Stockholm, Sweden, June 8-12, 2015*, pp. 53–68.

[28] T. Allweyer, *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2010.

[29] L. Padró and E. Stanilovsky, "Freeling 3.0: Towards wider multilinguality," in *Proc. 8th Int. Conf. Language Resources Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, pp. 2473–2479. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2012/summaries/430.html

[30] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "Brat: a web-based tool for nlp-assisted text annotation," in *Proc. Demonstrations 13th Conf. European Chapter Association Computing Linguistics, Avignon, France, April 23-27, 2012*. Association for Comput. Linguistics, pp. 102–107.

[31] P. Mirza, "Extracting temporal and causal relations between events," Ph.D. dissertation, Int. Doctorate School Inf. Communication Technol., Univ. of Trento, Via Calepina, 14, 38122 Trento TN, Italy, 2016.

[32] S. Smirnov, M. Weidlich, and J. Mendling, "Business process model abstraction based on behavioral profiles," in *Service-Oriented Computing*. Springer, 2010, pp. 1–16.

[33] H. Van der Aa, H. Leopold, and H. A. Reijers, "Comparing textual descriptions to process models: The automatic detection of inconsistencies," *Inf. Syst.*, vol. 64, pp. 447–460, Mar. 2017.

[34] J. Sànchez-Ferreres, J. Carmona, and L. Padró, "Aligning textual and graphical descriptions of processes through ILP techniques," in *Adv. Information Systems Engineering - 29th Int. Conf., CAiSE 2017, Essen, Germany, June 12-16, 2017*, pp. 413–427.

[35] V. Bernstein and P. Soffer, "Identifying and quantifying visual layout features of business process models," in *Enterprise, Business-Process Information Systems Modeling - 16th Int. Conf., BPMDS 2015, 20th Int. Conf., EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015*, pp. 200–213.

[36] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, "Soundness of workflow nets: classification, decidability, and analysis," *Formal Asp. Comput.*, vol. 23, no. 3, pp. 333–363, Aug. 2011, doi: 10.1007/s00165-010-0161-4.

[37] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 1, pp. 121–141, Jan. 1979, doi: 10.1145/357062.357071.

[38] C. Haisjackl, P. Soffer, S. Lim, and B. Weber, "How do humans inspect BPMN models: an exploratory study," *Softw. & Syst. Model.*, vol. 17, Oct. 2016.

[39] H. Leopold, S. Smirnov, and J. Mendling, "On the refactoring of activity labels in business process models," *Inf. Syst.*, vol. 37, no. 5, pp. 443–459, Jul. 2012, doi: 10.1016/j.is.2012.01.004.

[40] J. Mendling, H. A. Reijers, and J. Recker, "Activity labeling in process modeling: Empirical insights and recommendations," *Inf. Syst.*, vol. 35, no. 4, Jun. 2010.

[41] H. Leopold, R.-H. Eid-Sabbagh, J. Mendling, L. Azevedo, and F. Baião, "Detection of naming convention violations in process models for different languages," *Decis. Support Syst.*, no. 56, pp. 310–325, Jun. 2013.

[42] F. Pittke, H. Leopold, and J. Mendling, "Automatic detection and resolution of lexical ambiguity in process models," in *Software Engineering 2016, Fachtagung des GI-Fachbereichs Softwaretechnik, 23.-26. Feb. 2016, Wien, Österreich*, pp. 75–76.

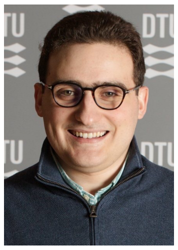
[43] PADS-UPC Research group. Modeljudge modeling session dataset. [Online]. Available: http://www.cs.upc.edu/~pads-upc/ModelJudgeData.zip

[44] J. Krogstie, *Model-based development and evolution of Inf. systems: A Quality Approach*. Springer Sci. & Business Media, 2012.

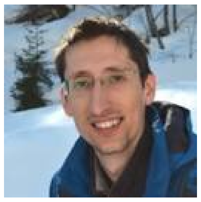
[45] J. Pinggera, "The Process of Process Modeling," Ph.D. dissertation, Univ. of Innsbruck, Dept. of Comput. Sci., Innrain 52, 6020 Innsbruck, Austria, 2014.

Josep Sànchez-Ferreres is a Ph.D. student at Universitat Politècnica de Catalunya, studying the relation between natural language and formal business process representations. His research interests include natural language processing, business process management and process mining but also Machine Learning and artificial intelligence. He has published papers in international conferences in the field (BPM, CAiSE).

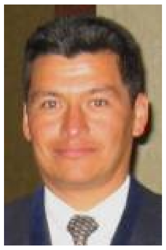
Luis Delicado Luis Delicado recieved the M.S degree in Computer Science mention in Computer Graphics at Universitat Politècnica de Catalunya. His research backgroud combines two different fields: Business Process and Crowd animations, with his role of Software developer inside and outside of his research. He has different papers published at international conferences in both fields (BPM, MIG).



Amine Abbad Andaloussi is a Ph.D. candidate at the Technical University of Denmark. He received his M.Sc. degree from the same university in 2018. His main research is about the understandability of business process models. With an academic background in computer science and strong research interest in cognitive psychology, Amine investigates the way end-users engage with business process models in order to improve the understandability of process modeling notations and enhance the existing modeling tool-support.



Andrea Burattin is Associate Professor at the Technical University of Denmark, Denmark. Previously, he worked as Assistant Professor at the same university, and as postdoctoral researcher at the University Innsbruck (Austria) and at the University of Padua (Italy). In 2013 he obtained his Ph.D. degree from University of Bologna and Padua (Italy). The IEEE Task Force on process mining awarded to his Ph.D. thesis the Best process mining Dissertation Award 2012–2013. His Ph.D. thesis has then been published as Springer Monograph in the LNBIP series. He served as organizer of BPI workshop since 2015, and special sessions on process mining at IEEE CIDM since 2013. He is also in the program committee of several conferences. His research interests include process mining techniques and, in particular, online approaches on event streams.



Guillermo Calderón Ruíz receives his Ph.D. in Engineering Sciences in the field of Computer Science from the Pontificia Universidad Catolica de Chile in 2011. He is a principal professor in Universidad Catolica de Santa Maria in Arequipa-Peru, he is also the Director of the School of Systems Engineering since 2017 in the same university. He also runs a software development center in the same university developing projects for the organizations in the southern of Peru. His research interest include process mining, data science, business process management, business intelligence and information technology. He is member of the IEEE Task Force on Process Mining, and he is also in the program committee of several conferences.

Barbara Weber Barbara Weber is Professor for Software Systems Programming and Development at the University of St. Gallen, Switzerland. She is Chair for Software Systems Programming and Development and Director of the Institute of Computer Science. Barbara's research interests include process model understandability, process of process modeling, process flexibility, and user support in flexible process-aware systems as well as neuro-adaptive information systems. Barbara has published more than 150 refereed papers, for example, in Nature Scientific Reports, Information and Software Technology, Information Systems, Data and Knowledge Engineering, Software and System Modeling, and Journal of Management Information systems and is co-author of the book "Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies" by Springer.

Josep Carmona received the M.S. and Ph.D. degrees in Computer Science from the Technical University of Catalonia, in 1999 and 2004, respectively. He is an associate professor in the Computer Science Department of the same university. His research interests include formal methods and concurrent systems, data and process science, business intelligence and business process management, and natural language processing. He has co-authored around 100 research papers in conferences and journals. He received best paper awards at the Int. Conf. on Application of Concurrency to System Design (ACSD 2009), at the Int. Conf. on business process management (BPM 2013), and at the International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016). He is the author of the book "Conformance Checking: Relating Processes and Models", by Springer.

Lluís Padró is Associate Professor at Universitat Poliècnica de Catalunya. His research area is framed in artificial intelligence, specifically in natural language processing, and more particularly in the development of language analysers (morphological, syntactic, dependency, sense disambiguators, semantic role labelers, named entity recognisers, etc). He has published more than 100 papers in the main conferences (ACL, ANLP, NAACL, EACL, COLING, EMNLP, etc) and journals (Computational Linguistics, Machine Learning, Journal on Language Resources and Evaluation, etc) in the area. He has served as program committee for lead conferences and journals (ACL, EACL, EMNLP, ICML, JLRE, Neurocomputing,...) He is the founder, administrator and main developer of project FreeLing (http://nlp.lsi.upc.edu/freeling), an open-source suite offering language analysis services in a variety of languages widely used both in academy and industry.