Sketching Process Models by Mining Participant Stories

Ana Ivanchikj and Cesare Pautasso

Software Institute, USI, Lugano, Switzerland
name.lastname@usi.ch

Abstract. Producing initial process models currently requires gathering knowledge from multiple process participants and using modeling tools to produce a visual representation. With traditional tools this can require significant effort and thus delay the feedback cycle where the initial model is validated and refined based on participants' feedback. In this paper we propose a novel approach for process model sketching by applying existing process mining techniques to a sample process log obtained directly from the process participants. To that end, we specify a simple natural language-like domain-specific language to represent process traces or fragments of process traces. We also illustrate the architecture of a live modeling tool, the Sketch Miner, implementing the proposed approach. The tool produces a draft visual representation of the control flow which is updated in real-time as the traces are written down. The draft model generated by the tool can later be refined and completed by the business analysts using traditional tools.

Keywords: Draft Process Model · Process Mining · Process Requirements · Textual Modelling DSL

1 Introduction

One of the identified challenges of business process management in a recent survey is the involvement of people with different skills and background [1], such as process participants with business domain knowledge, the business analysts with process modelling knowledge, and the software engineers with IT background. In the requirements gathering phase for the implementation of a new process in Process Aware Information Systems (PAIS), or when trying to model/improve existing processes, the people with detailed knowledge about the AS-IS or TO-BE business process are the participants in the process. Although they have deep knowledge about the activities that they perform [15], they might lack global knowledge about the end-to-end process [3]. Furthermore, abstracting from the process instances and thinking on the process model level is not always straightforward, especially for people with little or no modelling experience.

Drafting a model of the process with traditional modelling techniques, requires a dedicated person to step in the role of a business analyst, gather the process knowledge from different participants and use a graphical editor to manually create the initial draft process model. This requires time and significant cognitive effort by the business analyst, thus causing a delay in obtaining feedback on the draft model by the actual process participants. A comparative study by Damij [5] has shown that the process participant's role when using flowcharting modeling techniques, boils down to observation. While when using more natural language like techniques, such as activity tables, the process participants are actively involved to ensure that their activities are correctly captured in the model. On the same note, the study by Ottensooser et.al. [18] has shown that people with no formal training in business process modeling, understand better textual notations describing business processes, such as written use-cases. However, their understanding of the described business process increases by reading a BPMN diagram after having read the written use-cases. To summarize, the existing state of the art process modelling techniques do not empower nor motivate process participants to become directly involved in the initial drafting of the process model. Their only possible involvement is during the interviews with the business analyst, who is then responsible for abstracting the concepts and creating the general picture, which might not align with the reality. However, a potential misalignment will only be discovered after the first draft of the model is finished by the business analyst and shown to the participants, thus creating delays in the feedback cycle.

With the work presented in this paper we would like to get the process participants to become actively involved in drafting the model, thus speeding up the feedback cycle. To that end, we propose a simple Domain Specific Language (DSL) that would allow process participants to describe their user stories in a predefined textual format, similar to task lists written in natural language. By transforming these lists into a process log we leverage on existing process mining algorithms to discover the process described by the participants. This Model Sketching by Mining approach allows the mining algorithm to use the unified knowledge of different participants to deduce the control flow branches and to infer the presence of loops, and thus output in real-time an initial draft of the business process in the visual language of choice. The role of the creator of the initial draft model is transferred from the business analysts to the mining algorithm, while the business analyst steps in later, in the refinement and finalization of the model, which can be done with the traditional modeling tools. This does not mean that the business analyst cannot use the DSL to "take notes" and sketch the model while interviewing process participants. That types of involvement allow for a quick initial draft of the process model which can be used to facilitate the discussion and the validation of the model with the stakeholders. While traditionally process mining has been used for discovery of existing processes based on system logs [22], we propose to extend the use of existing mining algorithms to sketch processes out of possible user scenarios or hypothetical participant stories.

This paper is structured as follows. In Sec. 2 we discuss briefly the existing textual notations for modelling business processes and motivate the need of defining our own DSL for capturing process knowledge, which we present in Sec. 3. In Sec. 4 we describe the architecture of the Sketch Miner, the proof of concept tool for Model Sketching by Mining, which translates the DSL into complete process traces to be used as input to a mining algorithm. To show the use of the DSL and the tool, in Sec. 5 we model a travel reimbursement process, which we refer to when discussing the benefits and the limitations of our approach in Sec. 6. In Sec. 7 we discuss the related work, while in Sec. 8 we conclude the paper and present work that we plan to conduct in the future.

2 Textual Notations for Process Modeling

Using textual notations to generate visual models has been gaining on importance in the modelling languages that target primarily developers. For instance, in Ballerina[1], a programming language aimed at the implementation of microservices and API integration, the textual and the visual syntax are kept synchronized as they are being edited independently. There are also tools, such as PlantUML, which support textual modeling of many of the Unified Modeling Language (UML) visual diagrams which are aimed at documenting software systems artifacts. One of these UML diagrams, the Activity diagram, is intended as a graphical representation of workflows and as such can be used to model business processes. It can be modelled graphically with tools such as StarUML[2], but it can also be modelled textually with tools such as PlantUML[3], following the syntax rules for distinguishing between different diagram constructs. The syntax uses: 1) *keywords*, such as "start" and "stop" to denote the beginning and end of a diagram, "if", "then", "else", "elseif" for conditionals, "repeat", "while" for loops, or 2) *punctuation signs*, such as "`:`" to denote an activity, "`( )`" to denote a gateway, "`| |`" to denote swimlanes etc. A survey of textual notations for UML is available in [19].

In the domain of process modelling, the Business Process Execution Language (BPEL) is a textual XML based language for specifying process behaviour with no standard graphical visual notation[4]. It uses nesting of constructs to represent the control flow, using keywords to distinguish among different control structures such as `<sequence>`, `<flow>`, `<if>` etc. The target audience of this language are developers. Nitzsche et.al. [17] have proposed $BPEL^{light}$ to describe interactions only as message exchanges, regardless of the interface definition, thus separating the business logic from the technical protocols and messaging infrastructure. $BPEL^{light}$ extends BPEL with the `<conversation>` and `<interactionActivity>` elements which group the interaction activities and thus simplify the original BPEL language, facilitating the modelling of business processes at a more abstract level. However, also $BPEL^{light}$ does not provide a visual rendering of the process model.

[1] https://ballerina.io
[2] http://staruml.io
[3] http://plantuml.com/activity-diagram-beta
[4] https://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm

The Business Process Model and Notation (BPMN) is a standard process modelling language, designed starting from a visual syntax and later enhanced with an XML-based serialization. Recently a textual representation of BPMN has been proposed in plantBPMN [11]. Its target audience are developers, and potentially business analysts, and thus it supports a large number of BPMN constructs. It requires knowledge of BPMN terminology as specific keywords are used for all the node constructs such as "pool", "start", "start timer", "split", "task" etc. It also includes symbols that mimic the graphical constructs, such as the "`->`" for denoting the sequence flow.

The above mentioned DSLs require the users to learn and remember a syntax with different constructs, or rules for expressing different visualizations. As these rules are often similar to expressions used in programming languages, they might be intuitive for developers, but not necessarily for our primary target audience, i.e., process participants. The process participants would use the DSL for model sketching by mining only when mapping their everyday activities to a process, which is expected to happen rarely in their career. Thus, the textual DSL to be used by them should be as simple as possible and as close to natural language as possible, so that it can be explained and memorized fast, with no need of specialized training. The aim of such DSL is not to generate an executable process model, but rather to rapidly paint a model sketch for discussion purposes by inferring a process template from multiple scenarios representing individual instances. To do so, it does not need the full expressiveness of an executable process modeling language, such as BPMN or BPEL. Other authors have also pointed to the benefits of a simpler notation when gathering requirements [14]. By using mining to deduce the control flow constructs, we can avoid having to specify them in the textual DSL, and thus propose a simpler DSL than the ones mentioned above.

3 A DSL for Process Model Sketching by Mining

When the goal is obtaining an initial draft model fast, an important decision to make is what is the minimal necessary user input in order to maximize what can be expressed in the output. The desired output in our case is a draft process model, for which the required minimal input are the names of the activities in the process and the sequence in which they have to be completed. Although the DSL can be used by all involved parties, the primary targeted users are the process participants, who are generally not computer scientist, and thus are not familiar with the syntax of structured programming languages, and have limited knowledge of process modelling languages and process modelling. Therefore, one of the requirements for the DSL is that it should be as simple as possible and as close to natural language as possible. Additionally, as we propose using a mining algorithm to automatically generate the initial sketch of the process model, the input to the algorithm needs to mimic a set of traces of process instances. To get the full model, a complete process instance trace is required for each possible path that can be taken in the process. There are several issues with obtaining

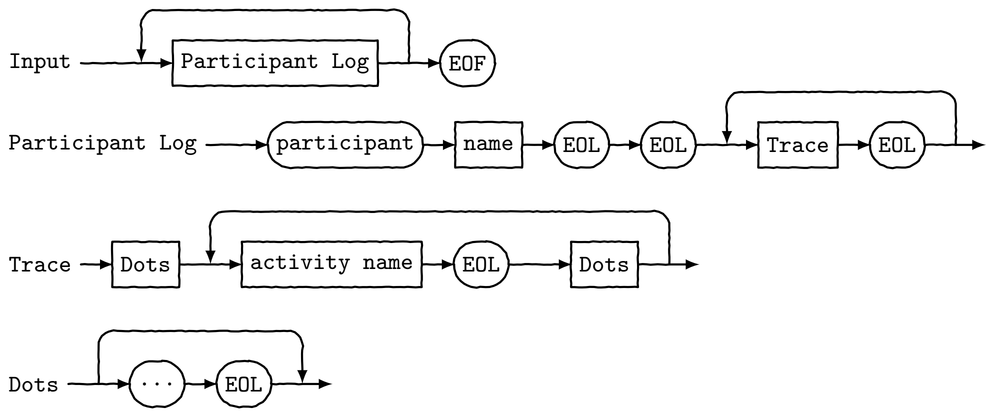


Fig. 1. Model Sketching by Mining DSL Grammar Definition (`EOL` indicates the end of the text line; `EOF` the end of the input file).

such traces directly from the user. First of all, although it is highly likely that individual process participants are at least aware of the activities that they need to perform and the immediate predecessor / successor activities, they may lack knowledge of the end-to-end process, and thus all the possible paths in the process. Another issue is that stating all the possible paths becomes repetitive when there are many alternative paths which have many shared activities.

The standard IEEE format for event logs used in process mining is the eXtensible Event Stream (XES) format[5]. XES is a tag-based language, where the information about the events belonging to a determined process instance lie inside a trace element, which contains one to many event elements. Different attributes can be defined for the event element, the timestamp and the activity name being the most frequently used ones. Although XES is human readable, it requires the knowledge of XML syntax, the specific tags and their meaning and is rather verbose for human users to write manually. Thus, we could not use this standard as a format for the user input, but nonetheless we used it as a target into which our DSL can be transformed.

Namely, in the DSL for Model Sketching by Mining we require the user to write the name of each new activity in a new line, in the order in which the activities are completed. No time-stamp is required as the sequential order is deduced from the order in which the activities' names are written. While system event logs would have a process instance ID and a participant ID associated with each event, in our DSL a new empty line in the text separates traces of different process instances, and the `participant` keyword followed by the name of a participant denotes who performs the activities listed after naming the participant. In Fig. 1 we provide a formal definition of the syntax of the DSL specifying the correct usage of the language constructs. Essentially, the allowed values in a single line of text are: 1) the `participant` keyword followed by the name of

[5] http://www.xes-standard.org

a participant; or 2) the name of the activity (empty spaces are allowed), or 3) an empty line (to mark the start of a new process instance); or 4) the dots "..." symbol (to indicate unknown fragments). While the names of the activities are sufficient to capture any complete trace of a process instance whose activities are performed by the same participant, we need the `participant` keyword as a construct to represent the hand-over of activities between participants. Furthermore, we need a construct to allow shortening the log by avoiding the repetition of the same sequences of activities in traces of process instances of different paths in the process. To that end, we introduce the "..." symbol. This symbol at the same time tackles the above mentioned issue of possible lack of knowledge of what the other participants are doing, which gives rise to the need to represent in the DSL fragments of process instance traces. The "..." symbol is essentially a placeholder for missing parts of the trace which are defined elsewhere. The precise semantics of the "..." symbol depends on its position relative to the start or end of the process instance trace, i.e., relative to the empty line. Following are the possible types of fragments of process instance traces and their semantics:

– Process instance trace that ***does not contain*** the "..." symbol: a sequence of activities which are all known to the user and which show a complete possible execution path of the process from the start to an end;

– Process instance trace that ***starts with*** the "..." symbol: the start of the process instance is not known to the user, or has been defined in the trace of other process instances. This fragment of a process instance trace states a sequence of activities that leads to the end of the process;

– Process instance trace that ***ends with*** the "..." symbol: the end of the process instance is not known to the user, or has been defined in the trace of other process instances. This fragment of a process instance trace states a sequence of activities that follows from the start of the process;

– Process instance trace that ***contains*** the "..." symbol: there is a part of the process instance that is not known to the user, or that has been defined in the trace of other process instances. This fragment of a process instance trace states a sequence of activities that starts and ends the process, but skips the middle of the process instance trace;

– Process instance trace that ***starts and ends with*** the "..." symbol: this type of trace is used as a fragment to fill in the placeholders in other traces in order to complete them. It does not correspond to a separate process instance.

4 Sketch Miner - A Tool for Model Sketching by Mining

As a proof of concept for the applicability of the proposed approach to use mining for sketching process models, we have designed the Sketch Miner, a tool which takes as input a process described with the DSL presented in Sec. 3 and provides as an output a draft BPMN model of the process (Fig. 2).

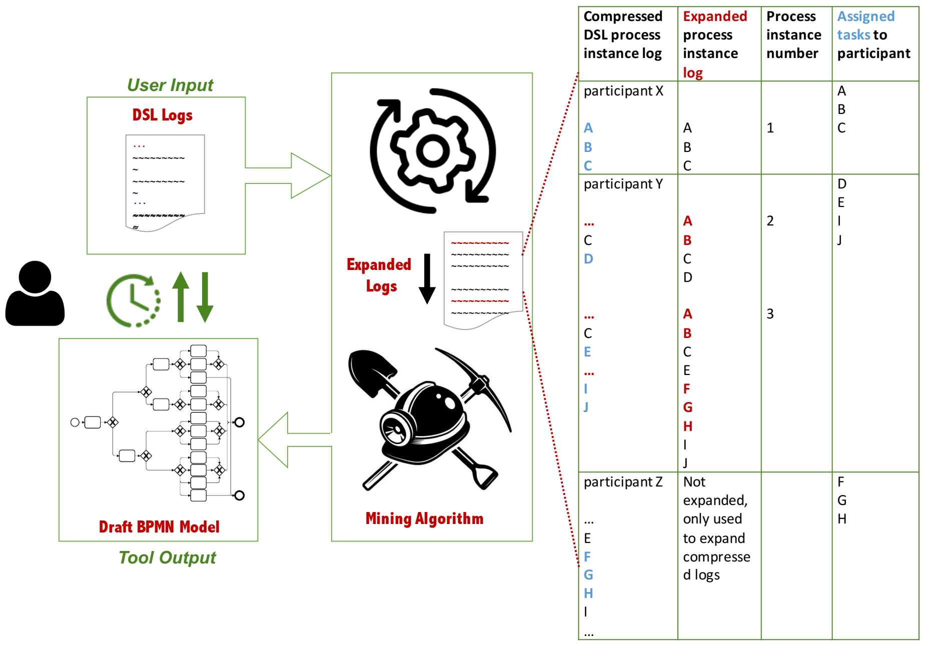
The tool first expands the traces written in the DSL to obtain the complete traces of all the possible paths that can be taken in the process. Namely, the DSL user input is parsed and each time the “...” symbol is encountered, depending on its position, the algorithm performs one of the following actions:

– If the “...” symbol is at the start of the process instance trace the algorithm searches for the first activity which is after the “...” symbol in all the other expanded process instance traces. When it finds it, it takes all the activities which precede it, thus creating one or more missing process fragments, which are then used to expand the initial process instance trace;

– If the “...” symbol is at the end of the process instance trace the algorithm searches for the last activity which is before the “...” symbol in all the other expanded process instance traces. When it finds it, it takes all the activities which succeed it, thus creating one or more missing process fragments, which are then used to expand the initial process instance trace;

– If the “...” symbol is in the middle of the process instance trace the algorithm searches for the first activity which is before the “...” symbol and the first activity which is after the “...” symbol in all the other expanded process instance traces. When it finds both these activities in the correct order, it takes all the activities which are between them, thus creating one or more missing process fragments, which are then used to expand the initial process instance trace.

As the trace expansion algorithm acts recursively and searches all expanded traces, if different sequences are identified as a match in different process instance traces, then they will all be used to expand the analyzed trace resulting with



Compressed DSL process instance log	Expanded process instance log	Process instance number	Assigned tasks to participant
participant X A B C	 A B C	 1	A B C
participant Y ... C D ... C E ... I J	 A B C D A B C E F G H I J	 2 3	D E I J
participant Z ... E F G H I ...	Not expanded, only used to expand compressed logs		F G H

Fig. 2. Architecture of the Sketch Miner

multiple expanded traces per one compressed trace. A simple example of how compressed traces are expanded by the Sketch Miner before passing them to the mining algorithm is provided in Fig. 2. As evident from the example, the assumption used by the expansion algorithm is the existence of at least one common activity between the compressed trace and the other traces, as the algorithm uses the common activity to identify the missing part of the path.

In order to enable automatic deduction of the activity role mapping, while keeping the DSL as simple as possible, we have introduced the constraint that the participant starting the process, when writing the traces, can only write down the activities (s)he is responsible for, using the "..." symbol when necessary. That way the algorithm can assign the activity to the first participant that mentions it. The other participants in the process, when using the "..." symbol, should state one activity preceding and/or following their activities depending on the position of the "..." symbol (beginning, end or middle of a process instance trace). In Fig. 2 there is a simple example showing which of the activities are assigned to which participant following the above mentioned constraints. Thus, we use the minimal assumption that process participants know all the activities they are responsible for as well as at least one activity preceding/following their activities so that log fragments of different participants can be connected. Nonetheless, participants may be aware of additional activities and they are free to state them.

The requirements for embedding a process mining algorithm within the Sketch Miner architecture involve the existence of an API to automatically feed the mining algorithm with the expanded traces. Additionally, users typing the traces should not have to wait to see the resulting model, but the model should be updated *live* as new entries of the traces are added. This could be achieved with an incremental mining approach [16].

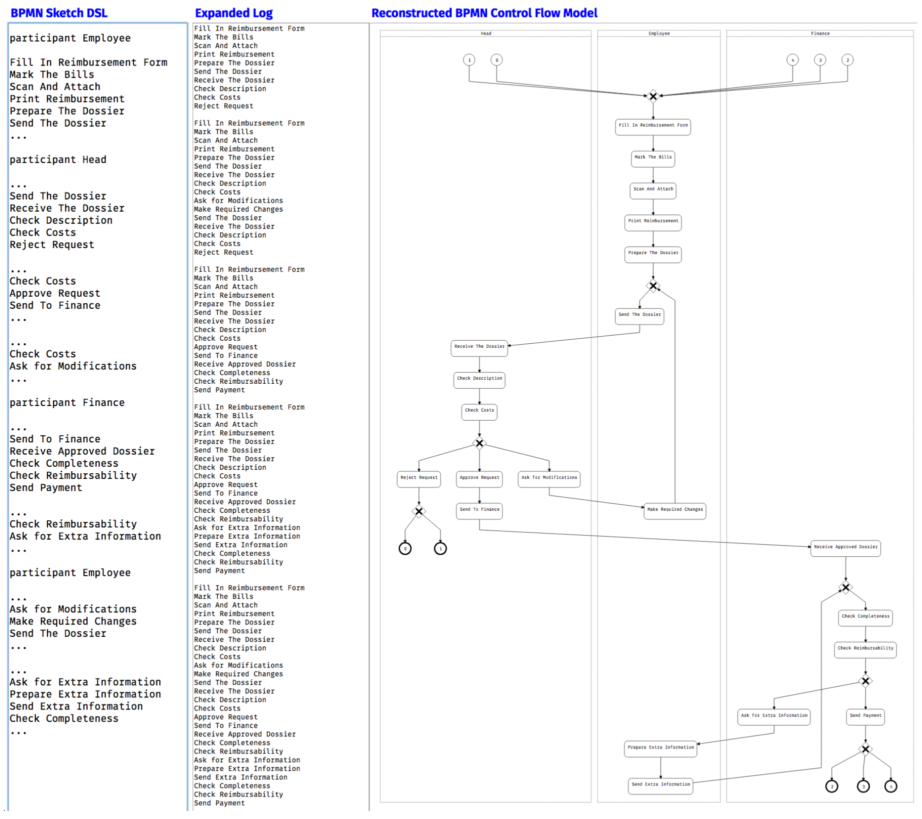
The current version of the Sketch Miner uses the Alpha algorithm for mining the expended traces and produces as output a BPMN diagram, serialized as an SVG image using the dagre-d3 library[6], so that it can be immediately displayed in a Web browser. Aiming at a proof of concept of this novel modeling approach, the validation of different mining algorithms for its implementation was not in the scope of this work. Furthermore, the approach of using mining for process model sketching is not intended to depend on the target language and can be potentially applied to other process modelling languages, such as Petri Nets, or proprietary flowcharting languages. The Sketch Miner is available at Github[7].

5 Travel Reimbursement Use-case

To show the use of the DSL proposed in Sec. 3 we will use the example of a travel costs reimbursement process. In this process, after returning from a business trip, Employees need to fill in the reimbursement request form, mark the bills with the expense number, scan and attach them to the form, print the form, and

[6] https://github.com/dagrejs/dagre-d3

[7] https://github.com/USI-INF-Software/RESTfulConversationMining



BPMN Sketch DSL

```
participant Employee

Fill In Reimbursement Form
Mark The Bills
Scan And Attach
Print Reimbursement
Prepare The Dossier
Send The Dossier
...

participant Head

...
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Reject Request

...
Check Costs
Approve Request
Send To Finance
...

...
Check Costs
Ask for Modifications
...

participant Finance

...
Send To Finance
Receive Approved Dossier
Check Completeness
Check Reimbursability
Send Payment

...
Check Reimbursability
Ask for Extra Information
...

participant Employee

...
Ask for Modifications
Make Required Changes
Send The Dossier
...

...
Ask for Extra Information
Prepare Extra Information
Send Extra Information
Check Completeness
...
```

Expanded Log

```
Fill In Reimbursement Form
Mark The Bills
Scan And Attach
Print Reimbursement
Prepare The Dossier
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Reject Request

Fill In Reimbursement Form
Mark The Bills
Scan And Attach
Print Reimbursement
Prepare The Dossier
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Ask for Modifications
Make Required Changes
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Reject Request

Fill In Reimbursement Form
Mark The Bills
Scan And Attach
Print Reimbursement
Prepare The Dossier
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Approve Request
Send To Finance
Receive Approved Dossier
Check Completeness
Check Reimbursability
Send Payment

Fill In Reimbursement Form
Mark The Bills
Scan And Attach
Print Reimbursement
Prepare The Dossier
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Approve Request
Send To Finance
Receive Approved Dossier
Check Completeness
Check Reimbursability
Ask for Extra Information
Prepare Extra Information
Send Extra Information
Check Completeness
Check Reimbursability
Send Payment

Fill In Reimbursement Form
Mark The Bills
Scan And Attach
Print Reimbursement
Prepare The Dossier
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Ask for Modifications
Make Required Changes
Send The Dossier
Receive The Dossier
Check Description
Check Costs
Approve Request
Send To Finance
Receive Approved Dossier
Check Completeness
Check Reimbursability
Ask for Extra Information
Prepare Extra Information
Send Extra Information
Check Completeness
Check Reimbursability
Send Payment
```

Fig. 3. Model Sketching by Mining in the Travel Reimbursement Use-case

together with the bills send it to their Head of Department for approval. After receiving the dossier the Head of Department checks the travel description and the stated costs, and can reject the request, or where needed ask the Employee to perform changes and resend the dossier. When the dossier is approved the Head of Department sends it to the Finance Department. After receiving it the Finance Department checks whether the submitted dossier is complete and whether the reported costs are reimbursable. If all the controls are passed successfully, the Finance Department sends the payment to the Employee. Otherwise they contact the Employee inquiring for additional information/documentation and re-performing the checks.

In the above described use-case there are three participants, the Employee, the Head of Department and the Finance Department. Probably none of them would know in detail what the other participants are doing as part of the described process, but we can assume that they all know at least the activity preceding/following the activities they are involved in. For instance, as can be seen in the left-hand side of Fig. 3, the Employees might not know the detailed steps that are taken by the Head of Department or the Finance Department, and thus they simply mark them with the "..." symbol. They know that some checks happen after they send the documents and that they can be asked to modify the request or to provide additional information, but they might not know the precise activities. On the other hand, the Head of Department can decide between three different paths after checking the documents, which is why (s)he specifies three different process instance traces. As two of the possible paths do not lead to an end of the process, they are placed in between the "..." symbols. On the other hand, the use of the "..." symbol at the beginning of the process instance traces allows the Head of Department to only state the new activities in the last two process instances, without repeating the activities already stated in the previous instance traces. Last but not least, the Finance Department, after performing the checks, can choose between asking for more information or sending the reimbursement payment, thus it only states two process instance traces. There are five possible paths assembled by the DSL log expansion, as evident in the central part of Fig. 3, and they are all passed to the mining algorithm in order to obtain the draft model presented in the right-hand side of Fig. 3.

6 Discussion

In real world scenarios, a written natural language description of the process is not always available, or when it is available it is not always up to date. In the travel reimbursement use-case mentioned in Sec. 5, with the traditional process modeling approach, such situation would require the business analyst to interview the three process participants in order to make the first draft model of the process. Then (s)he would need to identify the relevant activities at a meaningful level of granularity and name them. Bear in mind that in this toy example process there are already 20 activities, while in real world processes there can be many more. Then the business analyst, who is required to have prior knowledge of the visualization and the semantics of different BPMN constructs, or another process modelling language, should identify the divergence/convergence in the control flow and any possible loops. In the use-case model there are four exclusive gateways connected by two loops. Identifying them requires a cognitive effort, that as can be seen in Tab. 1, the business analyst is not spared of when using existing textual process modeling languages (described in Sec. 2). However, with our Model Sketching by Mining approach that part of the work is done by the mining algorithm in order to get the first draft of the process model. With this approach the active role of the business analyst can be postponed to after the first draft of the model, when (s)he would need to refine the draft model based on the discussion and the feedback from the process participants.

The Model Sketching by Mining approach enables the process participants to become directly involved in the modeling effort by writing the traces, regardless of the fact that they might lack the process modelling language knowledge. The DSL encourages them to think in terms of sequences of work units that they perform, and how they should compose them into activities and name those activities in a manner that is meaningful to them. For instance, in the travel

Table 1. Textual process modelling languages comparison

	Target users	**Users identify and name activities**	**Users identify the control flow**	**Modelling language support**	**Output format type**	**Model visualized**
PlantUML	Developers	Yes	Yes	Full	PNG, SVG	On request
plantBPMN	Developers	Yes	Yes	High	BPMN	On save
BPELight	Business analysts	Yes	Yes	Full	BPEL	N/A
Model Sketching by Mining	Process participants	Yes	No	Basic	SVG	Real time

reimbursement process described in our use-case, for a business analyst the "Scan and attach the bills" step might not seem important as (s)he might not be aware of the average number of bills per request. Thus, (s)he might compose it together with the previous step "Mark the bills with the expense number" into a single activity. However, for the employees it might be important to single out this step as a separate activity as it is time consuming and they might want to keep track of the time they spend scanning. Providing meaningful names to the activities is also not an easy task for a business analyst, but it might come more natural for the actual participants executing the activities.

6.1 Usability vs. Expressiveness Trade-off

The trade-off between usability and expressiveness is not inherent only to the domain of business process modelling [10]. The fundamental reason for this trade-off is the fact that greater expressiveness requires more language constructs, which hinders the usability as the users need more time and effort to learn the language. In existing textual process modeling languages the user typically needs to write text which mimics the graphical constructs of the modelling language (e.g., "`->`" to denote an edge in plantBPMN) or use the terminology of the modelling language (e.g., pool, task etc. in plantBPMN, fork in PlantUML) in order to obtain the visual model. This requires the user to have prior knowledge of the visual process modelling language. Usability, and fast learnability as part of it, is particularly important in our approach as we are primarily targeting process participants who are not frequently exposed to the visual process modelling language in their daily job. Thus, we do not aim at providing full support for all BPMN constructs. BPMN is a very expressive and rather complex language with over 100 constructs. As such providing a full support in our DSL would be likely to increase the complexity of the syntax, and drift the DSL design away from the original concept of using mining to infer the structure of the process as much as possible, as opposed to using the DSL to give a textual representation of the same, as in the case of PlantUML or plantBPMN. This is an important trade-off, which we have resolved favouring simplicity to enable process participants who have no knowledge of process modeling to state their user stories [14]. Furthermore, in order to both facilitate the learning of the basic BPMN constructs, and to provide fast feedback to the users, we have opted for generating the visual output at real time, as the users are typing their stories using the DSL. As evident in Tab. 1, this is not the case with existing textual process modeling languages. Namely, in PlantUML the textual model gets synchronized with the visual model only upon request, while plantBPMN generates a file in a BPMN XML format which then needs to be imported into a dedicated tool, such as Signavio[8] or an Eclipse plug-in, for visualization and further editing.

Even though we favour the DSL simplicity to expressiveness, we still aim to empower the process participants to tell their story by writing activity traces from their perspective. To that end, we have introduced the "..." symbol as a placeholder for parts of the process performed by others, and thus unknown to whoever is writing the story. The use of the "..." symbol is also meant to make the writing down of traces more time efficient, as it allows the users to avoid repeating sequences of activities which they have already written down. This is especially handy when there are alternative flows. For instance, in our travel reimbursement use-case, the Head of Department needs to make a decision whether to reject the request, ask for modification or send it to the Finance Department. Before making such decision the Head of Department would need to receive the reimbursement request dossier, check the travel description and check the stated costs. Writing down this sequence of activities three times would be too repetitive, so the Head of Department, when using the DSL, can avoid doing so by simply stating the above mentioned sequence in the first process instance trace and then starting a new instance trace using the "..." symbol followed by the "check the state costs" activity (see Fig. 3).

6.2 Potential Improvement of Modeling Efficiency

Wasana et.al. [2] have identified the *modelling methodology* and the *modelling tool* as two factors impacting the success of a process modelling project. While the methodology refers to the modelling approach being followed (e.g., how is the requirements and information gathering phase performed), the modelling tool refers to the software being used for the design of the models. Each of these factors has its related costs in terms of efficiency and cognitive load. When modeling business processes, we can differentiate between the cognitive load for 1) identifying and naming the process activities, and 2) identifying the correct topology of the control flow graph. The intrinsic cognitive load of people depends on their prior knowledge and the complexity of the task that they need to perform [21]. It has been shown that, when it comes to understanding visual models, readers first identify smaller submodels and later connect everything together [13]. Our DSL design takes advantage of this by allowing participants to specify sub-models they have first-hand knowledge about. These are then assembled by the mining algorithm to build the end-to-end process model. We expect this approach should decrease the cognitive load of the business analysts thanks to the use of a mining algorithm for reconstructing the control flow graph, which is an especially complex task in larger process models. There are various techniques for measuring the cognitive load, such as self-reported scales about the

[8] https://www.signavio.com

mental effort, or the difficulty of the tasks, as well as through response time [6], or eye movement tracking and pupillary response [4]. We plan to conduct such experiments in the future.

The type of ***modelling tool*** on the other hand, in addition to the cognitive load, can also impact the ***time efficiency*** of the modelling task per se. In a graphical editor a modeler would need to select the correct constructs from the available palette, place them in the modeling space, frequently using the drag&drop functionality, connect them in the correct order, and type the name of the activities. In the Sketch Miner, the user still needs to type the name of the activities as when working with the graphical editor. However, there is no drag&drop involved, thus there is no repositioning of the cursor within the activity shapes as all names are written on different lines of the same text editor. In other words, for equally experienced users we expect our textual entry to be more efficient than drawing graphical models, as argued in [12, 11]. In the future, we plan to conduct controlled experiments where one group is asked to use the Sketch Miner and another group is asked to use a graphical editor to construct the same model with only the core BPMN constructs. By limiting the use of the BPMN constructs in the graphical editor we ensure that any potential overhead is not caused by the complexity of BPMN as a modeling language.

6.3 Limitations

One limitation of our current tool is that it uses the assumption that the participants know at least the last activity preceding/following their involvement in the process. This assumption requires that different participants use the same name to identify such activities. However, one of the main disadvantages of the natural language is its ambiguity. To deal with this limitation in future versions of the Sketch Miner we can leverage on work done in the creation of domain ontology and semantic annotation of process models expressed in BPMN. For instance, in [7] the authors propose using natural language parsing combined with information content similarity for generating suggestions for the semantic annotation of business process elements. We can use similar approach for suggestions or auto-completion of activity names. Another complementary solution is to enable collaborative editing of the traces so that participants can input them at the same time and resolve conflicts as soon as they appear.

As supporting parallel flows is only planned for future extensions of the Sketch Miner, we currently do not introduce the explicit notion of an instance id, which could be an approach for dealing with situations when work done by two different process participants is done in parallel.

Our current approach can lead to over-fitting, i.e., the automatically derived model might allow for process instances which are not described in the DSL. To deal with this limitation in the future we plan to generate the traces of such over-fitting instances and present them to the user so that the process participants can decide if some of those traces should be excluded from the model.

7 Related Work

Visual vs. textual modelling of processes has been long studied. Damij [5] studies the appropriateness of flowcharts vs activity tables for capturing the reality of a process using two case-study processes. Her work advises business analysts to start with the activity table and then transform the table into a flowchart. Ottensooser et.al. [18] use an experimental study with different types of participants to evaluate the impact of modeling with BPMN vs modelling with written use-cases on the understandability of the process. As Damij, Ottensooser et.al. also advise to start with the textual technique. Namely, they show that the process understanding of all participants benefited from reading the textual model, while only BPMN trained participants benefited from the visual model. However, they also observed that untrained participants who read the BPMN model after reading the textual model did improve their understanding based on the BPMN model. These studies have inspired our approach of using a textual DSL to automatically generate a visual BPMN model. Effektif[9] started with an idea similar to ours, i.e., hiding complexity from the users by allowing them to create task lists by simply naming tasks while the tool would create the corresponding visual BPMN task constructs. Users would then use Signavio as a standard graphical editor to modify the control flow topology, e.g., by connecting the tasks and introducing the appropriate gateways. Their goal was to facilitate the automation of simple processes so that it can be done by any process participant, even without technical background. In our work we aim at simplifying and speeding up the creation of the initial model sketch by also deducing and drawing the control flow constructs, and not only the task constructs as in the case of Effektif. However, as in Effektiv the draft model generated with our approach may also need to be refined using a traditional graphical editor.

When it comes to related work in using process traces, in Test Driven Modelling (TDM) [23] for declarative process models, traces are used for creating test cases. A test case is a complete trace of a process instance that takes the form of a list of a sequence of activities that has to be supported by the defined process model. The completeness of traces requirement is relaxed in the recent work in [20]. However, in TDM traces are used for validation of an existing process model, while our primary goal is the sketching of the process model itself. In the past, scenarios and process fragments, called example runs [3] or oclets [9], have been used to create process models. While we propose a DSL for stating such process fragments, in [3] labelled partial orders and in [9] Petri Nets oclets are used. For the composition of the fragments we use a mining algorithm, while in the mentioned works the domain experts need to do the composition using composition operators (sequence, alternative, iteration, concurrency) [3] or combining actions enabled for extension [9].

Instead of the traditional process mining approach, whose output is automatically discovered process model, recent approaches tend to include the domain experts in the discovery of the process models, by allowing the user full control

[9] https://www.signavio.com/post/introducing-effektif/

over the creation of the process model and simply providing suggestions regarding next activities based on the probabilities discovered with process mining algorithms [8]. While our work also requires domain experts' input, it does not require the existence of real-world, system generated process execution logs, thus it can also be applied for processes which are still not supported by PAIS.

8 Conclusions and Future Work

Gathering requirements for the design of a new PAIS can be a time consuming task as it requires the cooperation of software technology experts with business domain experts. In a traditional approach, the requirements gathering phase starts with interviews with process participants conducted by a business analyst who, based on those interviews, needs to sketch a process model to be discussed and agreed upon with the process participants. To facilitate the work of the business analyst, in this paper we have proposed an approach for process model sketching by mining activity log written down directly by the process participants using a textual DSL we have designed for this purpose. The DSL lists traces with activity names on separate lines and it uses only one keyword to specify participant names and one symbol to indicate trace fragments. We have also presented the Sketch Miner, a proof of concept implementation of the approach, using BPMN for the model visualization. The novelty of this approach is that the control flow is deduced automatically by a mining algorithm, based on the participants' user stories, and the draft process model is rendered in real-time as the users type in the activity traces. In a traditional approach the discovery of the control flow is done mentally by the business analyst who then needs to use graphical tools to represent it.

While working on the DSL in the future we plan to investigate how far we can go with increasing the expressiveness of the language, without making it too complex to be learned and effectively used by process participants, while utilizing as much as possible the mining technique to deduce the topology of the control flow graph. We also plan to empirically validate the approach. Namely, we expect that this new model sketching approach can speed up the feedback cycles in the process design as a result of the automatic process model generation which could reduce the cognitive load of the business analyst. However, we will need to run controlled experiments to systematically validate and quantify these expected benefits.

References

1. Youseef Alotaibi and Fei Liu. Survey of business process management: challenges and solutions. *Enterprise Information Systems*, 11(8):1119–1153, 2017.
2. Wasana Bandara, Guy G Gable, and Michael Rosemann. Factors and measures of business process modelling: model building through a multiple case study. *European Journal of Information Systems*, 14(4):347–360, 2005.

3. Robin Bergenthum, Jörg Desel, Sebastian Mauser, and Robert Lorenz. Construction of process models from example runs. In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 243–259. Springer, 2009.
4. Ricardo Buettner. Analyzing mental workload states on the basis of the pupillary hippus. *NeuroIS*, 14:52, 2014.
5. Nadja Damij. Business process modelling using diagrammatic and tabular techniques. *Business process management journal*, 13(1):70–90, 2007.
6. Krista E DeLeeuw and Richard E Mayer. A comparison of three measures of cognitive load: Evidence for separable measures of intrinsic, extraneous, and germane load. *Journal of educational psychology*, 100(1):223, 2008.
7. Chiara Di Francescomarino et al. Supporting ontology-based semantic annotation of business processes with automated suggestions. In *Enterprise, Business-Process and Information Systems Modeling*, pages 211–223. Springer, 2009.
8. PM Dixit, HMW Verbeek, JCAM Buijs, and WMP van der Aalst. Interactive data-driven process model construction. In *International Conference on Conceptual Modeling*, pages 251–265. Springer, 2018.
9. Dirk Fahland. Oclets–scenario-based modeling with petri nets. In *Int'l Conference on Applications and Theory of Petri Nets*, pages 223–242. Springer, 2009.
10. Andre Freitas et al. Querying heterogeneous datasets on the linked data web: challenges, approaches, and trends. *IEEE Internet Computing*, 16(1):24–33, 2012.
11. Nicole Freund. Development of a Text-Based Representation of BPMN Models. Master's thesis, Leibniz Universität Hannover, Hannover, Germany, 2018.
12. Hans Grönninger, Holger Krahn, et al. Textbased modeling. In *Proc. of the 4th International Workshop on Software Language Engineering*, 2007.
13. Volker Gruhn and Ralf Laue. Reducing the cognitive complexity of business process models. In *2009 8th IEEE International Conference on Cognitive Informatics*, pages 339–345. IEEE, 2009.
14. Michael Havey. Keeping bpm simple for business users: power users beware. *BP-Trends (January 2006)*, 2006.
15. Jerry Luftman. Assessing it/business alignment. *Information Systems Management*, 20(4):9–15, 2003.
16. Florent Masseglia, Pascal Poncelet, et al. Incremental mining of sequential patterns in large databases. *Data & Knowledge Engineering*, 46(1):97–121, 2003.
17. Jörg Nitzsche, Tammo Van Lessen, et al. BPEL light. In *International Conference on Business Process Management*, pages 214–229. Springer, 2007.
18. Avner Ottensooser, Alan Fekete, et al. Making sense of business process descriptions: An experimental comparison of graphical and textual notations. *Journal of Systems and Software*, 85(3):596–606, 2012.
19. Stephan Seifermann and Henning Groenda. Survey on the applicability of textual notations for the unified modeling language. In *International Conference on Model-Driven Engineering and Software Development*, pages 3–24. Springer, 2016.
20. Tijs Slaats, Søren Debois, and Thomas Hildebrandt. Open to change: A theory for iterative test-driven modelling. In *International Conference on Business Process Management*, pages 31–47. Springer, 2018.
21. John Sweller. Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational psychology review*, 22(2):123–138, 2010.
22. Wil van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
23. Stefan Zugal, Jakob Pinggera, and Barbara Weber. Creating declarative process models using test driven modeling suite. In *International Conference on Advanced Information Systems Engineering*, pages 16–32. Springer, 2011.