Data & Knowledge Engineering 121 (2019) 1–17

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Learning process modeling phases from modeling interactions and eye tracking data

Andrea Burattin [a,*], Michael Kaiser [b], Manuel Neurauter [b], Barbara Weber [c,a]

[a] Technical University of Denmark, Denmark
[b] University of Innsbruck, Austria
[c] University of St. Gallen, Switzerland

ARTICLE INFO

Keywords:
Process of process modeling
Eye tracking
Interaction tracking
Automatic phase detection
Classification
Sequence labeling

ABSTRACT

The creation of a process model is a process consisting of five distinct phases, i.e., problem understanding, method finding, modeling, reconciliation, and validation. To enable a fine-grained analysis of process model creation based on phases or the development of phase-specific modeling support, an automatic approach to detect phases is needed. While approaches exist to automatically detect modeling and reconciliation phases based on user interactions, the detection of phases without user interactions (i.e., problem understanding, method finding, and validation) is still a problem. Exploiting a combination of user interactions and eye tracking data, this paper presents a two-step approach that is able to automatically detect the sequence of phases a modeler is engaged in during model creation. The evaluation of our approach shows promising results both in terms of quality as well as computation time demonstrating its feasibility.

1. Introduction

Process models play an important role in facilitating communication between different stakeholders and in documenting the organization's business processes. They are used for redesigning business processes as well as for automating them [1]. Process model development is an iterative and collaborative process which involves many stakeholders like domain specialists and system analysts [2]. It consists of two phases — elicitation and formalization. In elicitation phases, statements about the domain are generated and validated. In formalization phases, the extracted information is then used to create a formal process model [3].

The elicitation phase requires good communication between stakeholders and has been described in the literature as a negotiation process [4] in which different modeling alternatives are discussed. The formalization phase of a process model, also denoted as *process of process modeling (PPM)*, in turn, can be characterized as cognitive design activity during which a designer creates a formal process model from informal requirements descriptions [5]. This requires the designer to create a mental model of the domain and to externalize it as a formal process model [6]. During process model formalization the designer engages with the modeling platform (that provides a modeling notation as well as associated tool support) to improve the process being modeled. Research on model formalization has resulted in a description of the PPM and the identification of five distinct phases (i.e., problem understanding, method finding, modeling, reconciliation, and validation) [7]. The formalization of a process model is a flexible process during which the different phases are iteratively executed. A single instance of the PPM is called a *modeling session*.

* Corresponding author.
E-mail addresses: andbur@dtu.dk (A. Burattin), michael.kaiser@student.uibk.ac.at (M. Kaiser), manuel.neurauter@uibk.ac.at (M. Neurauter), barbara.weber@unisg.ch (B. Weber).

https://doi.org/10.1016/j.datak.2019.04.001
Received 17 July 2017; Received in revised form 25 February 2019; Accepted 7 April 2019
Available online 10 April 2019
0169-023X/© 2019 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Since each phase is characterized by different underlying cognitive processes, the factors determining a modeler's performance might vary for different phases. For example, while domain knowledge plays presumably an important role in problem understanding, process modeling knowledge and experience will be a relevant factor for modeling phases [7]. To enable a fine-grained analysis of the process of process modeling including its different modeling phases, the automatic detection of all five phases is required.

In [7,8], an algorithm for automatically detecting modeling and reconciliation phases from interactions with the modeling platform has been proposed. This approach exploits the interactions with the modeling platform to differentiate between modeling and reconciliation phases. Longer time periods characterized by an absence of interactions, i.e., problem understanding, method finding, and validation, cannot be differentiated and are labeled as *comprehension*. A comprehensive analysis of the process of process modeling, however, requires the detection of all five phases and not only those during which interactions with the modeling platform take place.

To overcome the limitations of the existing state of the art [7,8] this paper aims at the development of an automatic approach for phase detection in a single process modeling instance. The central research question of this paper can be formulated as follows: **"Is it possible to automatically and accurately detect the different modeling phases of single modeling instances?"**. To answer this research question we first developed a novel, machine-learning approach that exploits model interactions along with eye tracking data and slices the overall process modeling instance into a sequence of phases. We then validated our approach in two experiments. The validation of the approach against real data, referring to process modeling sessions, yielded promising results, both in terms of quality and computation time thus demonstrating the feasibility of our technique.

Our novel approach for automatic identification of phases makes a more fine-grained analysis of the process of process modeling possible and allows to take phase-specifics into account. An automated detection of such fine-grained phases within process modeling sessions is also a precondition for the development of a context-aware modeling platform that is able to detect the current context (i.e., the modeling phase the modeler is currently engaged in) and support the modeler in a phase-specific manner through recommendations, interventions, or even adaptations of the modeling platform (for example, the optimal tool support for validation is presumably different from tool support for problem understanding or method finding) [9].

The remainder of the paper is structured as follows. Section 2 reports some background information regarding the process of process modeling and the automatic phase detection problem. Section 3 formalizes the contribution of the paper and Section 4 evaluates the performance of the approach. Section 5 concludes the paper and sketches possible future work.

2. Background and related work

This section introduces the process of process modeling as typically presented in the literature including its five phases: problem understanding, method finding, modeling, reconciliation, and validation (cf. Section 2.1). Moreover, it introduces an existing approach for detecting modeling and reconciliation phases based on model interactions, which will serve as a baseline for the machine learning approach proposed in this paper (cf. Section 2.2). Finally, it discusses existing exploratory research on phase detection considering multi-modal data (i.e., model interactions as well as eye tracking data), which served as a starting point for developing our phase detection approach (cf. Section 2.3).

2.1. Process of process modeling

Existing research on the process of process modeling describes the act of creating a process model as a flexible process consisting of five distinct phases which are iteratively performed, i.e., they can be executed repeatedly and can be skipped for some iterations as needed [7].

Problem understanding. To develop a process model, modelers need to understand the problem (i.e., both the requirements and the process model created so far). During problem understanding, modelers build an internal representation (i.e., a mental model) of the problem to be modeled within their working memory [6].

Method finding. In method finding phases, modelers decompose the problem into smaller sub-problems and develop a solution that is independent of the concrete modeling notation. This can involve the hierarchically structuring of a process model, but also horizontally dividing the problem into sub-problems that can be mapped to workflow patterns [10] (e.g., embedding an activity into a conditional fragment for creating an optional activity).

Modeling. Once modelers have developed a solution for the problem, they can interact with the modeling platform to implement it by creating an external representation of the problem stored in their working memory. For instance, when using BPMN (Business Process Model and Notation) [11], modelers might insert an activity into the model and embed it into a conditional branch using gateways and sequence flows to implement an optional activity.

Reconciliation. Reconciliation phases are concerned with improving the understandability of the process model and facilitate subsequent phases. This includes changes to an activity label for resolving non-intention revealing naming of activities [12], but also relates to the secondary notation of process models [13,14].

Validation. In validation phases, modelers evaluate the quality of the externalized process model and assess if the model indeed provides a correct solution to the considered problem. In particular, in line with the SEQUAL framework [15] and grounded in semiotic theory, modelers might perform checks for identifying syntactical, semantical, and pragmatic quality issues in the process model [15].

As mentioned previously the creation of a process model is a flexible process during which the above described phases are iteratively executed. Fig. 1 depicts an example sequence of phases executed for one particular modeling session.

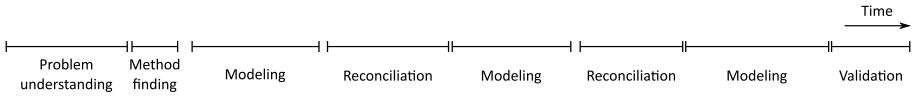


Fig. 1. Example sequence of phases of a process modeling session.

Table 1
List of possible interaction types.

#	Interaction type	Description
1	Create Node	Create an activity, gateway, or event
2	Delete Node	Delete an activity, gateway, or event
3	Create Edge	Create an edge connecting two nodes
4	Delete Edge	Delete an edge
5	Reconnect Edge	Reconnect an edge from one node to another
6	Select Tool	Select some tool from the toolbar
7	Select XOR	Select the XOR tool from the toolbox
8	Select AND	Select the AND tool from the toolbox
9	Select Activity	Select the Activity tool from the toolbox
10	Select Sequence Flow	Select the Sequence Flow tool from the toolbox
11	Select Start Event	Select the Start Event tool from the toolbox
12	Select End Event	Select the End Event tool from the toolbox
13	Rename	Rename an activity
14	Move Node	Move an activity, gateway, or event
15	Move Edge Label	Move the label of an edge condition
16	Create Bend Point	Create new bend point for an edge
17	Delete Bend Point	Delete new bend point for an edge
18	Move Bend Point	Update the routing of a bend point
19	Resize Node	Resize an activity, gateway, or event

2.2. Phase detection based on model interactions

In [7] a naïve approach to automatically detect modeling and reconciliation phases from a log of interactions obtained from the modeling platform Cheetah Experimental Platform (CEP) [16] is proposed. With respect to the modeling platform, the creation of a process model consists of a series of model interactions, e.g., adding activities and edges or moving elements for laying out the process model (cf. Table 1 for an overview of possible interaction types). The naïve approach maps the interactions with the modeling platform to the phases presented in Section 2.1. More specifically, interactions for creating model elements, deleting model elements, reconnecting edges, and adding/deleting edge conditions are classified as modeling actions. Interactions for laying out edges, moving model elements, renaming activities, and updating edge conditions are classified as reconciliation actions.[1] Identified actions are then aggregated to phases using the algorithm proposed in [8]; thresholds are used to avoid very short phases.

While modeling and reconciliation phases can be detected by this naïve approach, all time periods characterized by the absence of interactions, i.e., problem understanding, method finding, and validation, cannot be differentiated, but are subsumed as *comprehension*. Moreover, this assumption makes the technique unreliable in many circumstances, e.g., when a short comprehension phase is surrounded by reconciliation or modeling phases, as documented in [17] (in this case the naïve approach incorporates the comprehension phase into the surrounding). Our work overcomes these limitations and presents an approach that can automatically detect all phases.

2.3. Phase detection based on multi-modal data

To address the limitations of the naïve approach introduced in Section 2.2, we propose an approach based on multi-modal data. More specifically, we explored the possibility of using eye tracking data (in addition to model interactions) to distinguish between the phases of problem understanding, method finding, and validation (cf. [18]).

Using eye tracking it is possible to identify fixations, i.e., points on the screen modelers focus their attention on. The sequence of fixations of a modeler is denoted as scan-path. Fig. 2 shows an example of a scan-path during a brief session, with fixations represented as black dots. Areas of Interest (AOI) are a tool that is frequently used for the analysis of eye tracking data. AOIs refer to sub-regions of the stimulus (in our context the modeling platform) [19]. In our case AOIs refer to the textual description of the task, the modeling area, and the toolbox, as represented in Fig. 2. Having AOIs allows the extraction of metrics specific for each AOI (e.g., amount of fixations on text) and the analysis of transitions between AOIs [19]. For example, Fig. 2 shows transitions from text to toolbox, from toolbox to model, from model to toolbox, and from toolbox to model.

As a first step in the development of a phase detection approach based on multi-modal data, we conducted an exploratory study where we collected data from 116 student modelers and analyzed the comprehension phases we obtained from applying the naïve

[1] Note that interactions regarding the selection of elements in the toolbox (interactions # 6–12 in Table 1) were recorded, but not used in [7].

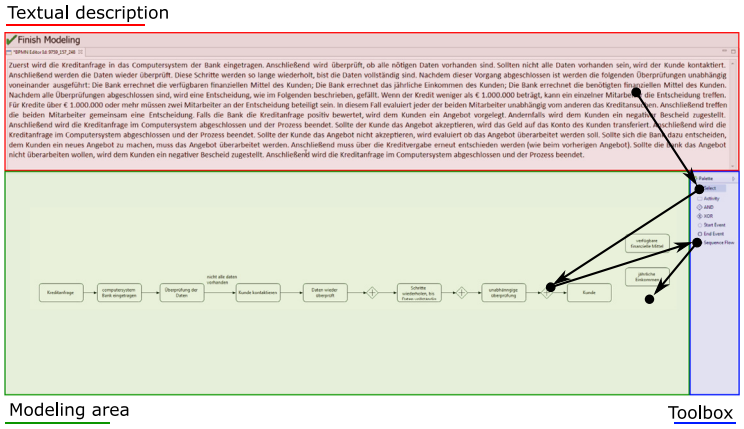


Fig. 2. BPMN modeling platform (i.e., Cheetah Experimental Platform) with three AOIs and the modeler's scan-path (i.e., sequence of fixations).

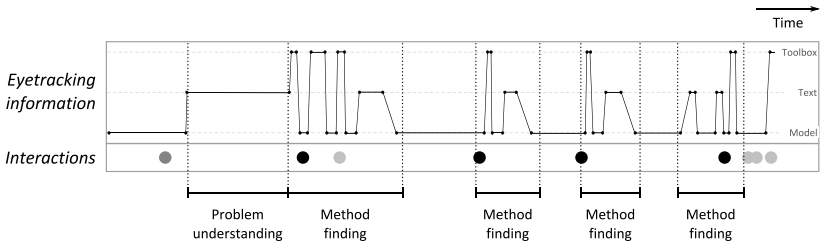


Fig. 3. Combined visualization of eye tracking information and model interactions.

phase detection algorithm outlined in Section 2.2 manually. Our preliminary findings from this exploratory study suggested that the fixation patterns we identified are related to the phases of problem understanding, method finding, and validation and can thus form the basis for automatically detecting the respective phases (cf. [18]). In particular, the way how the modelers' eyes transitioned between AOIs (i.e., model, text, toolbox) appeared promising. For example, our exploratory study showed that during problem understanding, the fixations were primarily on the textual description as modelers are building an internal representation of the problem. Similarly, during method finding, fixations were on parts of the textual description, followed by fixations on the modeling platform and the toolbox. Finally, during syntactic validation, fixations showed up on the modeling canvas and occasionally on the toolbox, while during semantic validation, the fixations occurred on the textual description and the modeling canvas with numerous switches between these two AOIs.

To further investigate fixation patterns that could form the basis of an automatic phase detection approach, we developed a visualization tool to represent transitions between AOIs together with the model interactions in an integrated manner [17]. An example of such visualization is reported in Fig. 3. It shows a problem understanding phase at the beginning with the attention of the user on the text. Afterward, several method finding phases occurred and all share a similar fixation pattern: the modeler first focused on the model or the text, then the focus briefly (and possibly repeatably) shifted to the toolbox (in order to understand the "tools" that can be used), and to the text (to map the requirements to the available tools). This further highlighted the potential of using eye tracking data as the basis for an automatic phase detection approach.

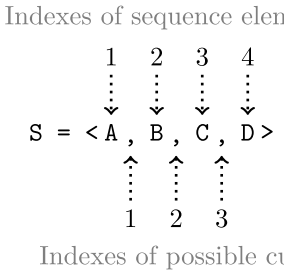
Additionally, Appendix A gives an intuition of how the multi-modal phase detection approach works in contrast to the state of the art approach: it shows a detailed example of a process modeling session including data on how the modeler interacted with the modeling platform.

3. Identification of phases using multi-modal data

This section introduces our approach for phase detection based on multi-modal data (i.e., model interactions combined with eye tracking data). We first provide preliminary definitions used in the remaining paper (cf. Section 3.1), then report a formal problem definition (cf. Section 3.2), before presenting our approach for phase detection (cf. Sections 3.3 and 3.4).

3.1. Preliminary definitions

The multi-modal data we are dealing with consists of each modality of an ordered sequence of events. These events can be interactions with the modeling tool or eye tracking data. In both cases, we formally describe these data as sequences. Just as an



(a) Indexes of sequence elements and possible cuts (i.e., $\{1, 2, 3\}$).

Cut	Partition
$\{\}$	<<A,B,C,D>>
$\{1\}$	<<A>,<B,C,D>>
$\{2\}$	<<A,B>,<C,D>>
$\{3\}$	<<A,B,C>,<D>>
$\{1, 2\}$	<<A>,<B>,<C,D>>
$\{1, 3\}$	<<A>,<B,C>,<D>>
$\{2, 3\}$	<<A,B>,<C>,<D>>
$\{1, 2, 3\}$	<<A>,<B>,<C>,<D>>

(b) Partitions for sequence <A,B,C,D> (induced by the power set of the set of cuts $\{1, 2, 3\}$).

Fig. 4. Indexes of all possible cuts of a sequence and the set of all possible partitions for the same sequence.

example, a sequence of interactions is graphically visualized in Fig. A.10a and a sequence of eye tracking data is represented in Fig. A.10b.

Given a set A, a *sequence* is a function $\sigma : \mathbb{N}^+ \to A$. We say that σ maps index values to the corresponding elements in A. For simplicity, in the rest of the paper, we will use the string interpretation of sequences: $\sigma = \langle s_1, \dots, s_n \rangle$.

In order to describe the basic operations we can do with our input data, we need some manipulation capabilities. Therefore, we assume that typical operators over sets and sequences are available and behave as commonly expected. Given two sequences the *concatenation* operator $\cdot$ creates a new sequence as the ordered combination of them: $\langle s_1^1, \dots, s_n^1 \rangle \cdot \langle s_1^2, \dots, s_m^2 \rangle = \langle s_1^1, \dots, s_n^1, s_1^2, \dots, s_m^2 \rangle$.

In all our cases, sequence elements contain a time component and we assume all sequences to be sorted temporally, in ascending order. We assume the availability of an operator which adds items to a sequence keeping it ordered. Moreover, we have access to the items of a sequence $S = \langle s_1, s_2, \dots, s_n \rangle$ using the *sequence indexing*: $s_i \in S$. Sometimes, the sequence elements are actually tuples, e.g., $t = (a, b)$ and to access their single components we assume a projection operator $\pi_a(t) = a$ and $\pi_b(t) = b$.

A sequence S can be divided into a sequence of (sub-)sequences $\langle S_1, S_2, \dots, S_m \rangle$, called *partition*, such that $S_1 \cdot S_2 \cdot \dots \cdot S_m = S$ (i.e., the partition *covers* the sequence). Specifically, given a sequence S with $|S| = n$, it is possible to identify $n-1$ indexes where to *cut* S, as depicted in Fig. 4a. Each cut splits the sequence in two additional parts. The power set[2] of the set of possible cuts identifies all possible partitions that can be generated starting from a sequence, as exemplified in Fig. 4b. In this case, the set of possible cuts is $\{1, 2, 3\}$ and its power set is $\{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. Therefore, given a sequence of length n, the number of possible partitions is the cardinality of the power set of the set of cuts, which is 2^n.

3.2. Problem formalization

In this paper we address the problem of dividing a modeling session into phases (i.e., find the partition in the session) introduced in Section 2.1 considering the user's interactions with the modeling platform as well as eye tracking data.

Definition 3.1 (*Interaction*)**.** An *interaction* $i = (t, intType)$ with the modeling tool is a pair containing the timestamp t – when the interaction took place – and the interaction type $intType$ which refers to one of those listed in Table 1.

Definition 3.2 (*Log of Interactions*)**.** A *log of interactions* $L_I = \langle i_1, i_2, \dots, i_n \rangle$ is a sequence of interactions with the modeling tool, progressively ordered according to their timestamps.

In our context, logs of interactions are collected with the Cheetah Experimental Platform. Considering the example depicted in Fig. A.10a, the corresponding log of interaction is:

$$\langle (t_1, Select\ Activity), (t_2, Create\ Node), (t_3, Create\ Node),$$
$$(t_4, Select\ Sequence\ Flow), (t_5, Create\ Edge), (t_6, Rename), (t_7, Rename) \rangle.$$

As a second source of input, we consider eye tracking data as detailed in the following, more specifically fixations and transitions.

Definition 3.3 (*Fixations*)**.** A *fixation* represents a period of time in which the gaze was fixed on one point (up to some approximations) [19]. Formally, in this work, a fixation $f = (t_s, t_e, aoi)$ is a tuple where t_s indicates the time the fixation started, t_e the time the fixation ended, and the area of interest of the fixation aoi, with $aoi \in \{text, model, toolbox\}$ as described in Section 2.3.

Definition 3.4 (*Transition*)**.** In our context, a *transition* represents the gaze movement from one area of interest to another one [19]. Formally, a transition $tr = (t_s, t_e, aoi_s, aoi_t)$ is a tuple reporting the beginning time of the transition t_s and the end of it t_e as well as the starting area of interest aoi_s and the target area of interest aoi_t. The possible values for the areas of interests are those reported in Section 2.3: $aoi_s, aoi_t \in \{text, model, toolbox\}$.

[2] The *power set* of set A is the set of all subsets of A, including the empty set and A itself.

Fixations and transitions of an eye tracking session can be calculated from the raw recordings of the eye tracker and are stored in temporal order in a log of fixations and transitions respectively.

Definition 3.5 (*Log of Fixations*). A *log of fixations* $L_F = \langle f_1, f_2, \dots, f_k \rangle$ is a sequence of fixations ordered progressively according to their start timestamp.

Considering the example of Fig. A.10b, it is possible to identify the following log of fixations:

$$\langle (t_{1_s}, t_{1_e}, text), (t_{2_s}, t_{2_e}, toolbox), (t_{3_s}, t_{3_e}, model), (t_{4_s}, t_{4_e}, toolbox) \rangle.$$

Definition 3.6 (*Log of Transitions*). A *log of Transitions* $L_T = \langle tr_1, tr_2, \dots, tr_m \rangle$ is a sequence of transitions progressively ordered according to their start timestamp.

Considering the example of Fig. A.10b, it is possible to identify the following log of transitions:

$$\langle (t_{1_s}, t_{1_e}, text, toolbox), (t_{2_s}, t_{2_e}, toolbox, model), \\ (t_{3_s}, t_{3_e}, model, toolbox), (t_{4_s}, t_{4_e}, toolbox, model) \rangle.$$

Using interactions, fixations, and transitions our goal is to automatically detect the phases introduced in Section 2.1. We can formalize this problem as finding a sequence of phases from a log of interactions, a log of fixations, and a log of transitions. More specifically, we aim to partition the logs such that each partition corresponds to one phase (where the events belonging to this partition are in line with the phase description outlined in Section 2.1) and adjacent partitions are of different phase types.

We approach the problem by decomposing it into a hierarchy of sub-problems. The general algorithm describing the overall approach is reported in Algorithm 1. It expects the logs of interactions, transitions and fixations as well as the minimum duration of a phase. Please note that we expect the logs to have synchronized time (i.e., they capture different angles of the same modeling session). In a first step, similarly to the state of the art, the algorithm detects a set of partitions with just high-level phases, i.e., modeling, reconciliation and comprehension phases (cf. line 1, Alg. 1; for details see Section 3.3).

Definition 3.7 (*High-Level Phase*). A *high-level phase* $p = (t_s, t_e, type)$ is a time interval, specified by a start time t_s and an end time t_e, with a type $type$ associated. The possible types are $type \in \{comprehension, modeling, reconciliation\}$.

Considering the example in Fig. A.10c, the result of the first line of the algorithm is the following sequence of high-level phases (wrt the figure, phases p_1 and p_2 are merged into a single comprehension phase):

$$\langle (t_{1_s}, t_{1_e}, comprehension), (t_{2_s}, t_{2_e}, modeling), (t_{3_s}, t_{3_e}, reconciliation) \rangle.$$

In a second step, the algorithm selects only the comprehension phases (line 2, Alg. 1) and replaces each of them with a sequence of low-level phases (i.e., problem understanding, method finding, syntactic and semantic validation) by applying one of the low-level phase detection algorithms (line 4, Alg. 1; for details see Section 3.4).

Definition 3.8 (*Low-Level Phase*). A *low-level phase* $p = (t_s, t_e, type)$ is a time interval, specified by a start time t_s and an end time t_e, with a type $type$ associated. The possible types are $type \in \{problem\ understanding, method\ finding, validation, modeling, reconciliation\}$.

Considering again the example in Fig. A.10c, the final result of the algorithm is the following sequence of phases:

$$\langle (t'_{1_s}, t'_{1_e}, problem\ understanding), (t''_{1_s}, t''_{1_e}, method\ finding), \\ (t_{2_s}, t_{2_e}, modeling), (t_{3_s}, t_{3_e}, reconciliation) \rangle.$$

3.3. Identification of high-level phases using multi-modal data

In order to extract the high-level phases out of a modeling session, we translate the problem at hand into a series of classification problems. Figs. 5a and 5b highlight the overall idea of our approach for high-level phase detection which can be divided into 4 steps. In the first step, the algorithm identifies a sliding window (cf. ① in Fig. 5a). In a second step, different features are extracted for the identified sliding window (cf. ② in Fig. 5a). The sliding window is then classified as any of the high-level phases using a pre-trained classifier (cf. ③ in Fig. 5a). This whole procedure is repeated for the entire session, resulting in a sequence of candidate phases. In a subsequent step, adjacent phases with the same phase type are merged (cf. ④ in Fig. 5b). Alg. 2 provides a more formal description of the high-level phase detection and is detailed in the following.

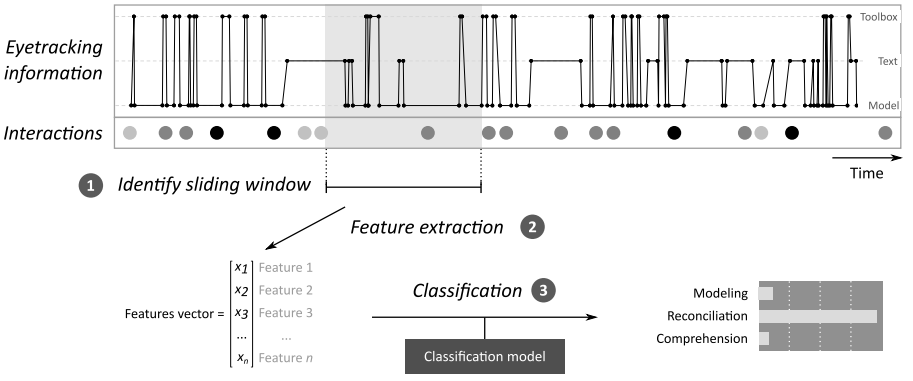
To obtain candidate high-level phases our algorithm uses a sliding window approach which allows to capture the optimal separation of partitions among events[3] and to obtain maximum flexibility in the selection of the classification algorithms and the features to use. In Alg. 2 the iteration over all windows is reported in line 5. The selection of the sliding window size plays an

[3] We might have two events (i.e., two candidate partitions) separated by a long time period. A sliding window approach allows to fine-tune the time where the separation takes place.

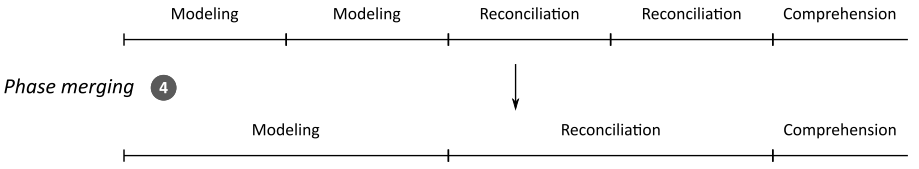
Algorithm 1: General algorithm to detect low- and high-level phases

Input: L_I: log of interactions
L_T: log of transitions
L_F: log of fixations
w_l: minimum duration of a phase
Output: P: sequence of phases discovered

1 $P \leftarrow$ HighLevelPhases(L_I, L_T, L_F, w_l) ▷ See Alg. 2
2 **foreach** $p = (t_s, t_e, type) \in P$ *with type = comprehension* **do**
3 Remove p from P ▷ Comprehension phases are refined
4 Add LowLevelPhases(L_T, t_s, t_e) to P ▷ See Alg. 3
5 **end**
6 **return** P



(a) Classification of a time window.



(b) Classification of the modeling session.

Fig. 5. Approach for high-level phase detection for multi-modal data.

important role: since each sliding window is classified as a single phase, its length represents the minimum duration of a high-level phase (or partition). Choosing a too small window size might lead to phases not being recognized (because of a lack of representative features). If the window size, in turn, is too long then phases might not be properly separated. For each sliding window, a set of features is extracted (line 6, Alg. 2) and the phase type is detected (line 7, Alg. 2) using a standard classification technique available in machine learning. The classifiers we tested in this paper are: Naïve Bayes [20,21]; Multilayer Perceptron [21,22]; Support Vector Machines (SVMs) [21,23] (with ANOVA – ANalysis Of VAriance – kernel [24]); Random Forests and Extra Trees [21,23,25]. At the end of each iteration, the algorithm appends the new phase to the sequence of phases (line 8). This procedure is repeated until the end of the modeling session is reached (line 5). Once the algorithm for high-level phase detection has provided a classification for the entire modeling session, there can be contiguous phases that were classified with the same type which are merged (line 10 in Alg. 2).

In order to identify the features to use for the classification (cf., line 6 in Alg. 2), we went through a feature engineering process. Data referring to user's interactions with the modeling tool represents the primary source of information. For example, the creation of a new task clearly represents a modeling phase, while the moving of an edge is part of a reconciliation phase. Eye tracking data (i.e., fixations and transitions between areas of interest) is also considered since it allows to refine the classification and in particular can help to identify short comprehension phases (cf. [17]). We selected features to capture all these notions and we grouped them into classes. Given a log of interactions L_I, a log of transitions L_T, a log of fixations L_F and a time window (i.e., a start sw_s and end time sw_e), the first feature class refers to features describing the user's interactions with the modeling tool. The primary purpose of these features is to capture the high-level phases by analyzing how the user performed the actual modeling:

Algorithm 2: Algorithm to detect high-level phases (HighLevelPhases)

Input: L_I: log of interactions
L_T: log of transitions
L_F: log of fixations
ω_l: minimum duration of a phase
Output: P: sequence of high-level phases discovered

$P \leftarrow \langle\rangle$ ▷ Sequence of high-level phases
$start \leftarrow \pi_{t_s}\left(L_{T1}\right)$ ▷ Timestamp of first interaction in L_T
$end \leftarrow \pi_{t_e}\left(L_{T|L_T|}\right)$ ▷ Timestamp of last interaction in L_T
$C \leftarrow$ trained classification model ▷ Obtain a trained classifier
▷ Construct all possible sliding windows and classify each of them as proper phase
foreach *window* (sw_s, sw_e) *in between start and end* **do**
  $\mathbf{F} \leftarrow$ ExtractFeatures$(L_I, L_T, L_F, sw_s, sw_e)$ ▷ Extract the feature vector for the current window
  $type \leftarrow C(\mathbf{F})$ ▷ Get most likely type for the window
  $P \leftarrow P \cdot \langle(sw_s, sw_e, type)\rangle$ ▷ Append the new phase
end
▷ In P there can be contiguous phases with the same type
Merge phases of P that are contiguous and with the same type
return P

- The number of interactions $\#(T)$ of a particular type T taking place within the time frame of the sliding window:

$$\#(T) = \left|\left\{(t, intType) \in L_I \mid t \geq sw_s \wedge\ t < sw_e \wedge\ intType \in T\right\}\right|.$$

Interactions with the modeling tool can refer to different interaction types (e.g., Create Node, Move Node). Specifically, as reported in Table 1, 19 different interaction types can be differentiated. This class of features counts the number of interactions of a particular type within a particular sliding window resulting in 19 features. Additionally, interaction types can be classified either as *modeling interactions* (i.e., types 1–12 in Table 1) or *reconciliation interactions* (i.e., types 13–19 in Table 1). Thus, we consider the number of modeling interactions and the number of reconciliation interactions as two additional features. This set of features is useful to provide information about the actual operations performed during the given sliding window.

- Time distance to the closest interactions of a particular type, both occurring before and after the current sliding window:

$$\text{before}(T) = \begin{cases} sw_s - \max\{t \mid (t, intType) \in L_I \\ \quad \wedge\ t < sw_s \wedge intType \in T\} & \text{if } \exists (t, intType) \in L_I : \\ & \quad t < sw_s \wedge intType \in T \\ \infty & \text{otherwise} \end{cases}$$

$$\text{after}(T) = \begin{cases} \min\{t \mid (t, intType) \in L_I \wedge t > sw_e \\ \quad \wedge\ intType \in T\} - sw_e & \text{if } \exists (t, intType) \in L_I : \\ & \quad t > sw_e \wedge intType \in T \\ \infty & \text{otherwise} \end{cases}$$

This class of features considers the time of the closest interactions of a particular type before and after the sliding window resulting into 38 features, i.e., 19 for before and 19 for after. Moreover, we consider 4 additional features capturing the time of the closest modeling interaction (i.e., T contains activities 1–12 in Table 1) and reconciliation interaction (i.e., T contains activities 13–19 in Table 1) before and after the sliding window. This set of features is useful to characterize the "neighborhood" of the sliding window under examination and therefore to improve the classification of borderline cases (e.g., windows with no interactions, but just before a dense cluster of modeling interactions which suggests a modeling phase about to start).

The second group of features refers to the information coming from eye tracking:

- The sum of the durations of all fixations in one area of interest:

$$\text{fixations}(area) = \sum_{(t_s, t_e, aoi) \in L_F} (t_e - t_s)\left[t_s \geq sw_s \wedge t_e < sw_e \wedge aoi = area\right].$$

This measure is computed for each area of interest. Since we have 3 areas of interest (c.f. Section 2.3), we have 3 additional features. Depending on the phase, different areas of interest might be in the focus. For example, during modeling phases, we expect most of the time spent on the model canvas whereas, during comprehension phases, we expect most of the time spent

Algorithm 3: Algorithm to detect low-level phases with HMM or CRF

Input: L_T: log of transitions
start: start time of window
end: end time of window
Output: sequence of low-level phases discovered

▷ Filtering of transitions, to consider just the given time interval
1 $T \leftarrow \langle (t_s, t_e, aoi_s, aoi_t) \in L_T \mid t_s > start \wedge t_e \leq end \rangle$
2 $F \leftarrow$ convert T into a sequence of features ▷ See Section 3.4
3 $S \leftarrow \mathsf{SequenceLabeler}(F)$ ▷ Standard HMM or CRF procedure
▷ In S we have a sequence (same length as T) where each component is classified as a low-level phase
4 Merge contiguous components of S with the same classification
5 **return** S

on the text and on the toolbox. This set of features is relevant in order to describe how much time the user spent looking at the different areas of interest, to reinforce the phase classification and therefore to help in disambiguating borderline cases [18].

- The number of transitions from one area of interest to another:

$$\text{transitions}(area_s, area_t) = |\{(t_s, t_e, aoi_s, aoi_t) \in L_T \mid t_s \geq sw_s \wedge t_e < esw_e \wedge aoi_s = area_s \wedge aoi_t = area_t\}|.$$

This measure is computed for each combination of transitions. As we have 3 areas of interest and it is possible to have transitions within the same area of interest, we introduce $3^2 = 9$ additional features. As for the previous group of features, this set of features is relevant in order to disambiguate borderline situations. An example of such case is reported in [17], where a phase was classified as reconciliation even though the number of transitions clearly showed a comprehension pattern (flipping between text and model).

One additional feature is included: the number of features with value set to 0. This feature can be seen as a way of quantifying the *absence* of interactions. Considering all the features described each sliding window extracts from the logs a feature vector with 76 numerical components.

3.4. Identification of low-level phases

As a result of the high-level phase detection algorithm, we obtain a sequence of phases classified as either modeling, reconciliation, or comprehension. In this section, we introduce our approach for low-level phase detection which further refines comprehension phases into *problem understanding*, *method finding* and *validation*. Since, by definition, comprehension phases are characterized by no interaction with the modeling tool, to detect low-level phases we can rely only on the eye tracking data and, in particular, we focused on the log of transitions. To detect the low-level phases we devised two approaches, graphically depicted in Fig. 6b. We rephrased our problem as sequence labeling and applied either (HMM) [23,26–28] or Conditional Random Fields (CRFs) [29,30] to solve it. The low-level phases identification approach starts by filtering the log of transitions for only those that occurred within the comprehension phase under examination (cf. Fig. 6a). Each sub-sequence of transition is converted into a sequence of features (cf. ① in Fig. 6b) and a sequence classifier attaches a possible low-level phase type to each transition (cf. ② in Fig. 6b). Finally, these points are merged into proper phases (cf. ③ in Fig. 6b).

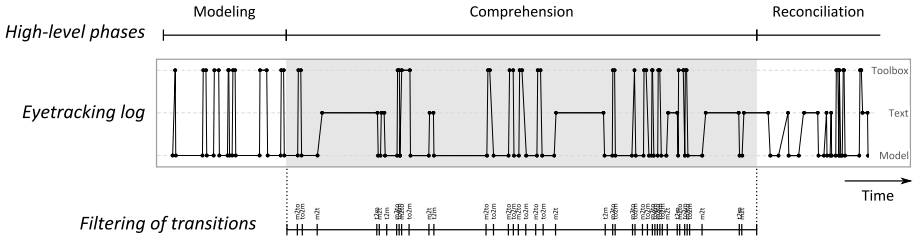
A formalization of this approach is reported in Algorithm 3: given a log of transitions (line 1, Alg. 3), it first converts it into a sequence of feature vectors (line 2, Alg. 3). The sequence of features is fed to a sequence labeler which associates each element with a class. In our case, the labels correspond to the low-level phases we are interested to detect (line 3, Alg. 3). Such a sequence of labels is post-processed by merging equivalent contiguous labels, thus creating time intervals describing the actual low-level phases (line 4, Alg. 3).

As described in the previous section also for the low-level phases identification we went through a feature engineering process. The set of features includes the source and the target AOIs of the current transition as well as the source and the target AOIs for the 2 previous and the 2 following transitions. Additionally, we added 2 features to indicate if the given transition is the first or the last within the comprehension phase.

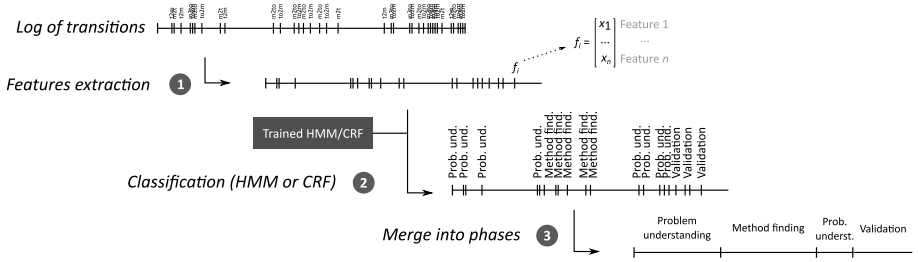
4. Experimental evaluation

This section describes the evaluation of the algorithms for the classification task of the high- and low-level phases. With this evaluation we want to answer the following two research questions:

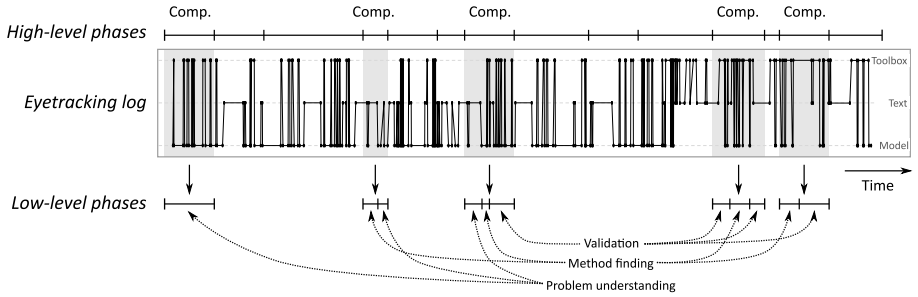
Q1 How well does the machine learning approach classify high-level phases in comparison with the state of the art?
Q2 How well does the machine learning approach segment a modeling session into low-level phases?



(a) Filtering of the transitions referring to a single comprehension phase.



(b) Labeling with HMM or CRF.



(c) Low-level phases detected for all comprehension phases.

Fig. 6. Different steps involved in the identification of low-level phases using HMM or CRF.

4.1. Data-set and ground truth

The data set used for our evaluation stems from a modeling session experiment where we collected model interactions and eye tracking data from novice modelers. This experiment took place in 2015 at the University of Innsbruck (Austria) and 116 psychology students (participation was voluntary) were asked to model a mortgage process (after receiving proper training). The log of interactions was directly obtained from CEP after the modeling session (cf. Section 2.2); eye tracking data was collected using a Tobii TX300 eye tracker and the log of fixations (i.e., time-series of fixations including coordinates) was obtained with the help of Tobii Studio. The fixations were mapped onto the AOI (cf. Fig. 2) using the coordinates of the fixation provided by Tobii Studio, thus obtaining a representation as described in Definition 3.5. The log of transitions was computed based on the log of fixations.

To extract our ground truth knowledge, i.e., the "gold standard" for our dataset, we started from video recordings of different subjects. For this experiment we use data generated by 5 subjects randomly selected among the participants. The video recordings were manually classified by two experts independently and discrepancies between the two classifications were resolved by a consensus building process. For the manual labeling, we used the descriptions of the phases introduced in Section 2.1, i.e., problem understanding, method finding, modeling, reconciliation, and validation. The data-set manually enriched represents the gold standard used for computing the qualitative performance of our approaches. Table 2 depicts the distribution of high-level phases and the distribution of low-level phases among subjects.

Please note that in order to answer the two research questions we will use only parts of the dataset. Specifically, to answer Q1 we will consider only the high-level phases, whereas to answer Q2 we will operate on the low-level phases.

Table 2
Absolute frequency of phase types in the ground truth and duration of modeling sessions for each subject.

	Subjects					Total
	s_1	s_2	s_3	s_4	s_5	
Modeling	40	41	47	22	49	199
Comprehension	35	25	36	22	38	156
Reconciliation	24	26	35	20	21	126
Sem. validation	15	6	15	7	17	60
Syn. validation	0	4	16	5	12	37
Method finding	31	18	27	16	16	108
Prob. under.	19	17	21	6	18	81
Duration	00:23:57	00:39:50	00:31:09	00:21:33	00:40:17	02:36:47

Table 3
Time needed for training and prediction of low-level classifiers.

	Training time	Prediction time
Markovian (baseline)	620.0 ms	> h
CRF	216.0 ms	2 ms
HMM	0.5 ms	0.8 ms

4.2. Question Q1: Comparison to the state of the art

To answer research question Q1, we use the state of the art automated phase detection approach by [7, Sec. 6.3.1] introduced in Section 2.2, which is based on user interactions only, and compare its results with those obtained by the approach presented in this paper. For the actual comparison, we adopted the *accuracy* measure, i.e., the ratio of correctly classified phases over all phases.

In order to extensively analyze the possible factors affecting the quality scores, we tested the different classifiers varying the window size and the set of features used. Fig. 7 depicts the corresponding accuracy values. Moreover, the measures for the state of the art approach are depicted for reference. As a further reference, No Information Rate (NIR) curves are reported, indicating the accuracy of selecting a random class. Our results show that the Extra Trees classifier achieves the highest accuracy with a value of 86% when a window of 5000 ms is used and both eye tracking and interaction features are exploited. 4 of the tested techniques have configurations that outperform the state of the art and only Naïve Bayes reports very poor performance.

Besides accuracy, we also evaluate the time needed for training and prediction. For both, the training and prediction times, we used the *wall-clock* as the measure, i.e., the time difference between the time at which the task finished and the time the task started. We repeat the training procedure 5 times and calculate the average. Additionally, we measure the wall-clock time needed by classifiers to predict the phases of a subject not used for training. We repeat this procedure 5 times and the average wall-clock time is computed. Fig. 8 shows the time performance for training the classifiers and for predicting the phases (except for Naïve Bayes due to lack of interest given its accuracy performance). With increasing window sizes the time needed for training and prediction decreases since it becomes easier to classify the time window (due to the presence of more observations).

In conclusion, our results showed that question Q1 can be answered positively, i.e., the new approach outperforms the state of the art approach.

4.3. Question Q2: Performance of segmenting into low-level phases

In contrast to the high-level phases classification, the low-level phases classification cannot be performed with the state of the art approach since the low-level phase classification operates only during *comprehension* phases (i.e., when no interaction takes place). Therefore, to define a baseline, we defined a "brute-force" algorithm called *Markovian* which tries all possible partitions and selects the most likely one, assuming Markov property (i.e., the future state only depends on the current one), as described in Appendix B.

For evaluation, we use the accuracy and the wall-clock time as measurements on the comprehension phases of the ground truth (for the same subjects evaluated in the previous section). Fig. 9 depicts the accuracy values for the detection of low-level phases. The highest accuracy is achieved by CRF with 83%, whereas the baseline stops at 76%.

For evaluating the time performance, we trained on the data of 4 subjects and used the fifth for the prediction. The training and prediction procedure is repeated 5 times, and the average is taken. Table 3 depicts the results. The prediction for the Markovian (baseline) approach is very slow, because it employs a brute-force method. CRF represents a very good trade-off: even though it is not as fast as HMM, both the training and the prediction times are very good.

In conclusions, we can answer question Q2 by showing the performance (both in terms of time requirements and quality) of the different techniques we devised. The CRF approach achieves the best accuracy while showing a good time performance.

4.4. Limitations

The main limitation of the approaches presented stems from the assumption that all phase types can be induced by interactions with the modeling platform and eye movements on the screen. While this assumption is realistic in some settings (e.g., teaching

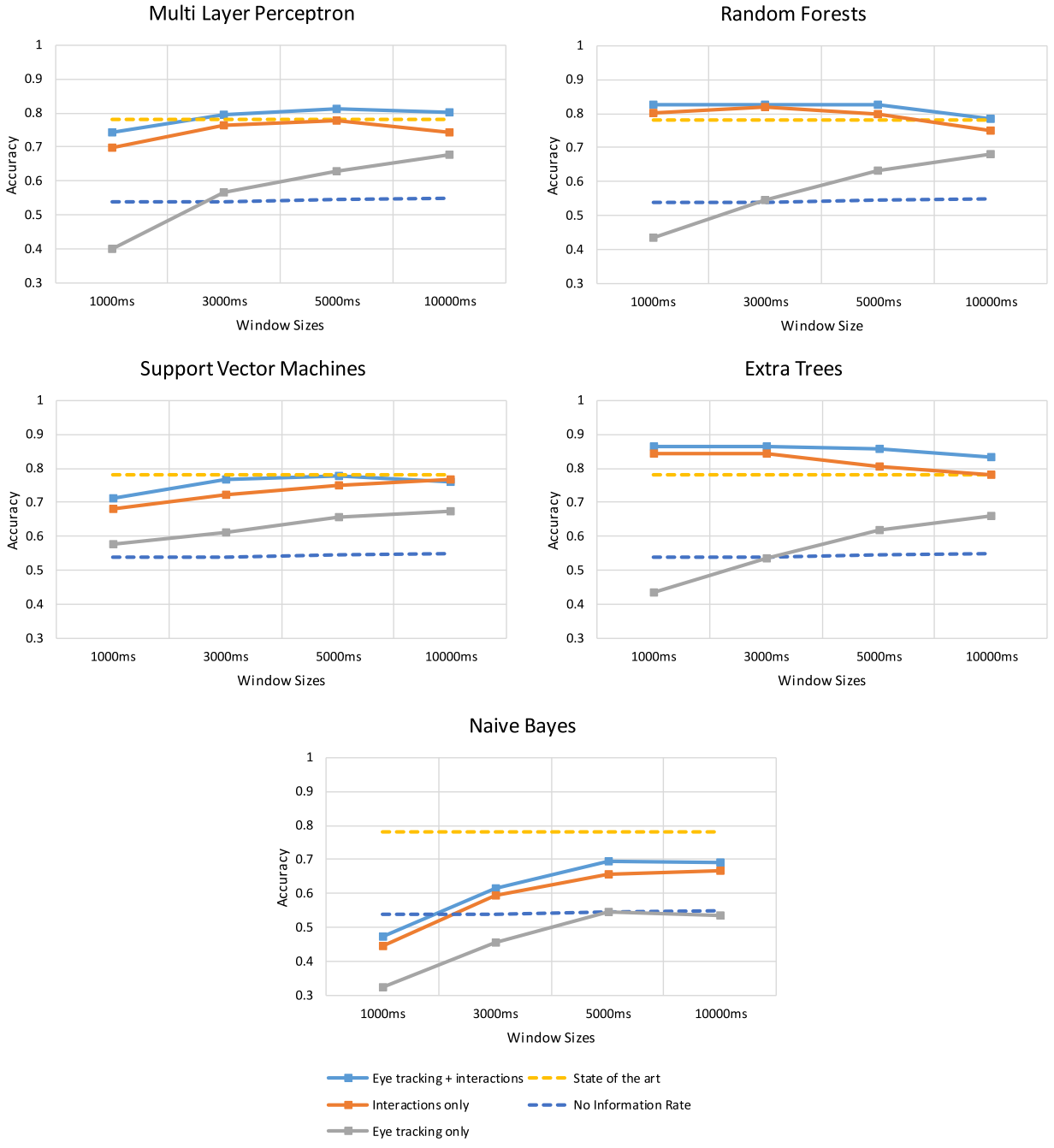


Fig. 7. Accuracy for detection of high-level phases with different approaches and different window sizes. The legend is shared by all charts.

exercises), it might not hold in the general case. However, eye tracking technologies are getting increasingly powerful enabling the mapping of eye tracking data onto real-world objects as well. Thus, it is realistic to assume that our approach can be applied in more dynamic settings in the future. A technical limitation of the approaches is the potential impact of the window size on the identification of high-level phases: a bad decision on this regard can lead to poor results (also for low-level phases).

Finally, it is important to mention that the manual labeling of the gold standard represents a limitation regarding the generalizability of our experimental results. Such labeling might be affected by human perception since it is a manual process. To mitigate this problem we asked two experts to independently label the instances and reach consensus.

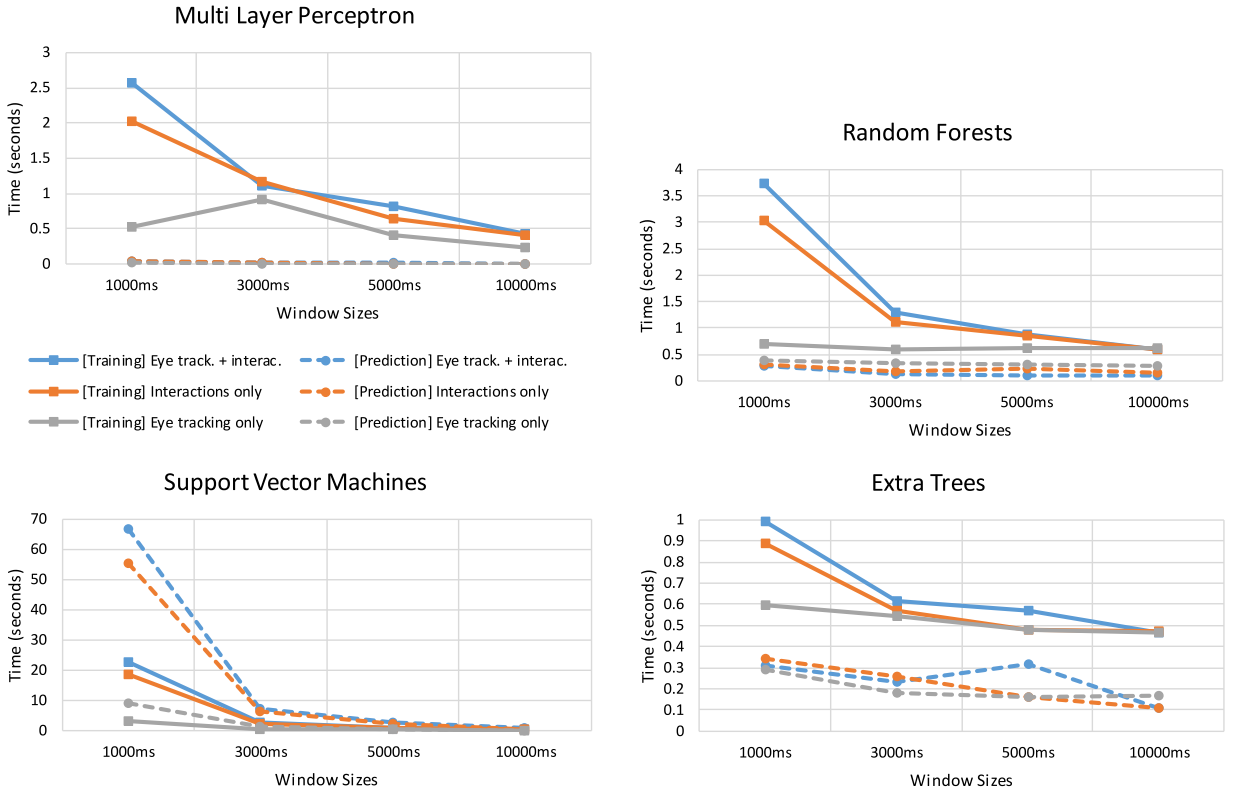


Fig. 8. Wall-clock time for training and prediction with different window sizes.

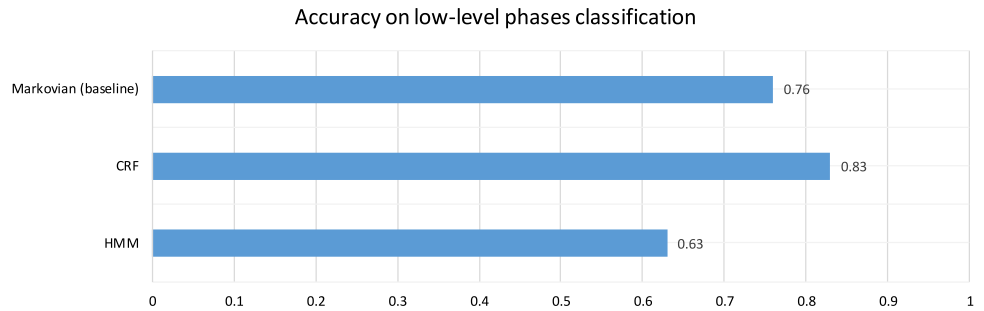


Fig. 9. Comparison of the accuracy for low-level classification for the different techniques devised as well as for the baseline (Markovian).

5. Conclusion and future work

The process underlying the creation of process models has recently received a lot of attention. Specifically, the importance of process models, for communication and documentation purposes, drives the need for improving their correctness and understandability.

Historically, the analysis of the process of process modeling relied just on the recording of user's interactions with the modeling tools. Exploiting this data source, the literature reports techniques to gather information regarding the different modeling phases taking place (i.e., comprehension, modeling, reconciliation). With the contribution presented in this paper, we showed how to process the data in a completely new fashion, also incorporating a new source of information, i.e. eye tracking, resulting in a twofold improvement of the state of the art. On the one hand, we are now able to extract phase types that were not identifiable with state of the art techniques (i.e., problem understanding, method finding, semantic and syntactic validation), thus delivering additional useful information to the analysis. On the other hand, the new analysis technique and the exploitation of the new data source resulted in superior accuracy in identifying the phase types with respect to the state of the art.

Leveraging the technique reported in this paper allows the automatic detection of the different phases a user is performing during a modeling task. The automatic phase detection will allow a more fine-grained analysis of modeling sessions. Moreover, this can

be used to provide better modeling experience through phase-specific modeling support which, in turn, will result in better process models. In particular, the automatic phase detection can provide important contextual information to be able to identify how to best support the modeler in her specific situation through feedback, interventions, or adaptations. This represents the main long-term goal and future working direction of the paper. At the same time, we plan to investigate different possibilities to improve the overall quality of the phase detection, as well as the introduction of new techniques, resulting in even more accurate inferences.

Acknowledgment

This work is funded by the Austrian Science Fund (FWF) project "The Modeling Mind: Behavior Patterns in Process Modeling" (P26609).

Appendix A. Data example with expected results

Fig. A.10 presents an example of a process modeling session including data on how the modeler interacted with the modeling platform and how her attention was distributed. Moreover, it depicts the phases that should be detected when applying the multi-modal phase detection approach presented in this paper.

In particular, Fig. A.10a provides the sequence of interactions with the modeling platform that occurred during the modeling session. Each interaction is numbered $i_1, \ldots, i_7$ and the respective interaction type is shown on top (for the full list of possible types see Table 1). Fig. A.10a also shows how the process model (as a result of the model interactions) evolved over time. With the first interaction, i.e., i_1, the modeler selected the tool for adding activities from the toolbox. This was followed by the creation of two nodes (i.e., i_2, i_3). After that, a different tool from the toolbox was selected (i_4) and an edge was created (i_5). Finally, the two activities were renamed (i_6, i_7). Please note that not all interactions changed the structure of the model: i_6 and i_7 are reconciliation actions since they consisted of renaming activities already in the model. Additionally, i_1 and i_4 did not cause any change on the artifact since they just involved the selection of a tool from the modeling platform toolbox.

Fig. A.10b, in turn, focuses on the eye tracking input stream highlighting the AOIs the modeler focused on as well as the transitions between AOIs. In the beginning, the user focused on the text (aoi_1), followed by the toolbox (aoi_2) and the modeling canvas (aoi_3). In the end, after switching the focus to the toolbox to select a different tool (aoi_2), the user looked again at the modeling canvas (aoi_3). Please note that this sequence of transitions between AOIs is also visible in the scan-path shown in Fig. 2.

Given the above data, the application of the state of the art technique described in Section 2.2 would (based on model interactions only) result in the detection of only 3 phases (i.e., "comprehension", "modeling", "reconciliation").

The aim of this paper, instead, is to combine the sequence of interactions with the eye tracking data in order to infer modeling phases with a finer granularity (i.e., where "comprehension" is split into "problem understanding", "method finding", and "validation") as reported in Fig. A.10c. In this case, during the first time period, no interactions were observed (cf. Fig. A.10a) and the user spent all time focusing on the text (as indicated in Fig. A.10b) and therefore our system should infer a "problem understanding" phase (p_1). This is followed by "method finding" (p_2) as the user selected a new tool from the toolbox (as indicated both in Figs. A.10a and A.10b). Then, while the user was focusing on the model and creating activities and the edge, a "modeling" phase took place (p_3). Finally, when the user was spending time on the model without changing its structure, but renaming activities, a "reconciliation" phase should be detected (p_4).

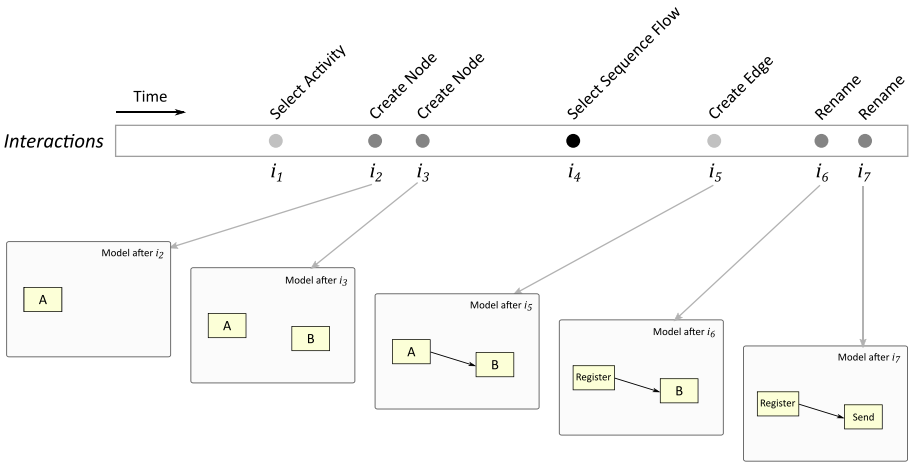
Appendix B. Segmentation with Markov Chain

The "brute-force approach" used as baseline to detect low-level phases is based on Markov Chains (MC) [28,31]. Markov Chains provide a probabilistic framework to describe a system whose current status only depends on the previous one.

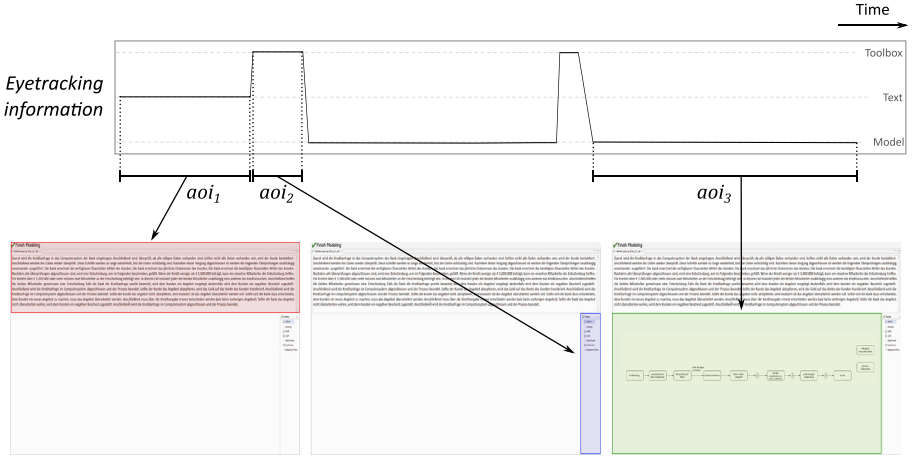
We can use this notion to *construct* a Markov Chain for each specific low-level phase we want to detect. To achieve that, we feed the learning algorithm with the sequence of transitions observed for a specific phase (i.e., the gold standard) to obtain a MC describing the low-level phase under examination. Then, given an unseen sequence of transitions, we can use our MC to obtain the probability that the sequence has been generated by the given low-level phase.

A formalization of the entire approach is reported in Algorithm 4. First of all, the algorithm filters the log of transitions, to consider only those referring to the considered time frame (line 1, Alg. 4). Comprehension phases are a generalized phase type, each of them can consist of one or more sub-phases, i.e., problem understanding, method finding, syntactic, and semantic validation. However, since a Markov Chains can only provide the probability of a sequence constituting a particular phase type, we have to generate all possible partitions of the sequence (line 4, Alg. 4) and calculate probabilities for each of them. Since a partition is basically a sequence of sub-sequences, the approach iterates through the partition itself and assigns a score to each sub-sequence, which represents a candidate low-level phase (line 9, Alg. 4). To do that, the sequence is checked against all MCs for each possible low-level phase type, and the best performing (i.e., the one with the highest probability) is returned. The scores for each sequence of the partition are summed up (line 10, Alg. 4) and the candidate sequence of low-level phases is built (line 11, Alg. 4). The algorithm keeps only the best performing partition (lines 13–16, Alg. 4), which is eventually returned as final output (line 18, Alg. 4).

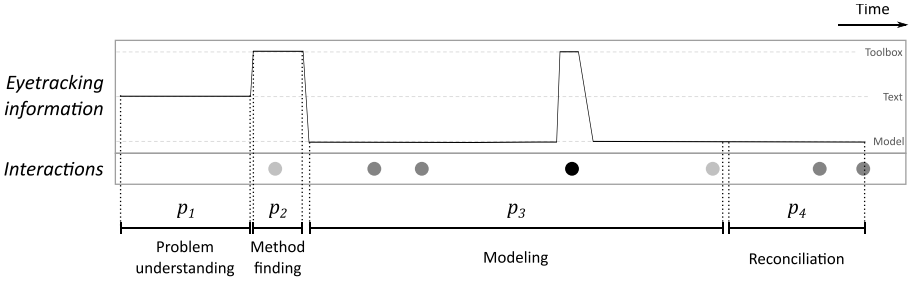
Despite the efficiency of computing the likelihood of a sequence, the most relevant drawback of this approach is the number of partitions it has to evaluate. Such a brute force mechanism makes the whole approach very inefficient and therefore hardly applicable in real scenarios. Still, it is relevant since it allows the identification of all partitions (including the correct one as well) and therefore can be used as baseline.



(a) Input: example of sequence of modeling interactions $i_1, \ldots, i_7$. The evolution of the artifact after each interaction is also represented.



(b) Input: eye tracking information. Areas of interest receiving attention during some time periods $(aoi_1, \ldots, aoi_3)$ are also visualized.



(c) Combined view of the two input streams and output with the sequence of phases $(p_1, \ldots, p_4)$ inferred from the given data.

Fig. A.10. Example of a process modeling session with model interaction and eye tracking data as well as the inferred phases.

Appendix C. Implementation details

The approach presented in this paper has been implemented using different programming languages and techniques.

Algorithm 4: Algorithm to detect low-level phases with Markov Chains

Input: L_T: log of transitions
start: start time of window
end: end time of window
Output: sequence of low-level phases discovered

▷ Filtering of transitions, to consider just the given time interval
$T \leftarrow \langle (t_s, t_e, aoi_s, aoi_t) \in L_T \mid t_s > start \wedge t_e \leq end \rangle$
▷ Define the best score and the best sequence
$score_{best} \leftarrow -\infty$
$seq_{best} \leftarrow \langle\rangle$
$P \leftarrow$ generate all possible partitions of the sequence T ▷ See Section 3.1
foreach *partition* $p \in P$ **do**
  $score \leftarrow 0$
  $seq \leftarrow \langle\rangle$
  ▷ Iterate through all sub-sequences of the current partition to check its quality
  foreach *sequence* $s \in p$ **do**
    $score_s, type \leftarrow$ highest likelihood that s is generated by one of the Markov Chains, and the phase type the M.C. refers to
    $score \leftarrow score + score_s$
    $seq \leftarrow seq \cdot \langle (\pi_{t_s}(s_0), \pi_{t_e}(s_{|s|}), type) \rangle$
  end
  ▷ If the current partition is the best so far, update the global values
  if $score > score_{best}$ **then**
    $score_{best} \leftarrow score$
    $seq_{best} \leftarrow seq$
  end
end
return seq_{best}

The high-level phase classification is completely coded in Python and the open-source *ecosystem* SciPy.[4] We extensively used the `scikit-learn` package [32] for training and validating the different classifiers.

For the low-level phase classification, Matlab[5] was used for the HMM. The R programming environment[6] has been adopted to program the Markov Chain algorithm, and Python and the `sklearn-crfsuite` (which is based on CRFsuite [33]) package was exploited for CRF.

References

[1] P. Fettke, How conceptual modeling is used, Commun. Assoc. Inf. Syst. 25 (2009).
[2] S. Hoppenbrouwers, H.A. Proper, T.P. van der Weide, Formal modelling as a grounded conversation, in: Proc. LAP'05, 2005, pp. 139–155.
[3] S. Hoppenbrouwers, H.A. Proper, T.P. van der Weide, A fundamental view on the process of conceptual modeling, in: Proc. ER'05, 2005, pp. 128–143.
[4] P. Rittgen, Negotiating models, in: Proc. CAiSE'07, 2007, pp. 561–573.
[5] J. Recker, N. Safrudin, M. Rosemann, How novices design business processes, Inf. Syst. 37 (6) (2012) 557–573.
[6] P. Soffer, M. Kaner, Y. Wand, Towards understanding the process of process modeling: Theoretical and empirical considerations, in: Proc. ER-BPM'11, 2012, pp. 357–369.
[7] J. Pinggera, The Process of Process Modeling (Ph.D. thesis), University of Innsbruck, Austria, 2014, URL http://bpm.q-e.at/wp-content/uploads/2014/10/thesis_jakob_final.pdf.
[8] J. Pinggera, S. Zugal, M. Weidlich, D. Fahland, B. Weber, J. Mendling, H.A. Reijers, Tracing the process of process modeling with modeling phase diagrams, in: Proc. ER-BPM'11, 2012, pp. 370–382.
[9] B. Weber, M. Neurauter, A. Burattin, J. Pinggera, C. Davis, Measuring and explaining cognitive load during design activities: A fine-grained approach, in: Proceedings Gmunden Retreat on NeuroIS (Preprint), 2017.
[10] W.M.P. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow patterns, Distrib. Parallel Databases (14) (2003) 5–51.
[11] OMG, Business Process Model and Notation (BPMN) - Version 2.0, Beta 1, 2009.
[12] B. Weber, M. Reichert, J. Mendling, H.A. Reijers, Refactoring large process model repositories, Comput. Ind. 62 (5) (2011) 467–486.
[13] M. Petre, Why looking isn't always seeing: Readership skills and graphical programming, Commun. ACM 38 (6) (1995) 33–44.
[14] J. Mendling, H.A. Reijers, J. Cardoso, What makes process models understandable? in: Proc. BPM'07, 2007, pp. 48–63.
[15] J. Krogstie, G. Sindre, H. Jørgensen, Process models representing knowledge for action: a revised quality framework, Eur. J. Inf. Syst. 15 (1) (2006) 91–102.

[4] See https://www.scipy.org/.
[5] See https://www.mathworks.com/products/matlab.html.
[6] See https://www.r-project.org/.

[16] J. Pinggera, S. Zugal, B. Weber, Investigating the process of process modeling with cheetah experimental platform, in: Proc. ER-POIS'10, 2010, pp. 13–18.
[17] A. Burattin, M. Kaiser, M. Neurauter, B. Weber, Eye tracking meets the process of process modeling: a visual analytic approach, in: Proc. TAProViz'16, 2016.
[18] B. Weber, J. Pinggera, M. Neurauter, S. Zugal, M. Martini, M. Furtner, P. Sachse, D. Schnitzer, Fixation patterns during process model creation: Initial steps toward neuro-adaptive process modeling environments, in: Proc. HICSS'16, IEEE, 2016, pp. 600–609.
[19] K. Holmqvist, M. Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, J. van de Weijer, Eye Tracking: A Comprehensive Guide to Methods and Measures, OUP Oxford, 2011.
[20] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, third ed., Prentice Hall, 2009, p. 1152.
[21] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, in: Springer Series in Statistics, Springer-Verlag New York, 2009.
[22] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.
[23] C.C. Aggarwal, Data Mining, Springer International Publishing, 2015, p. 734.
[24] T. Hofmann, B. Schölkopf, A.J. Smola, Kernel methods in machine learning, Ann. Statist. 36 (3) (2008) 1171–1220.
[25] Tin Kam Ho, Random decision forests, in: Proc. ICDAR'95, 1995, pp. 278–282.
[26] L. Rabiner, B. Juang, An introduction to Hidden Markov models, IEEE ASSP Mag. 3 (January) (1986) Appendix 3A.
[27] S.R. Eddy, Hidden Markov models, Curr. Opin. Struct. Biol. 6 (3) (1996) 361–365.
[28] J. Kacprzyk, W. Pedrycz (Eds.), Springer Handbook of Computational Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
[29] J. Lafferty, A. McCallum, F. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data, in: Proc. ICML'01, 2001, pp. 282–289.
[30] C. Sutton, A. Mccallum, An introduction to conditional random fields for relational learning, in: L. Getoor, B. Taskar (Eds.), Introduction to Statistical Relational Learning, Vol. 2, MIT Press, 2006, pp. 93–128.
[31] R. Serfozo, Basics of Applied Stochastic Processes, Springer Science & Business Media, 2009, p. 443.
[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[33] N. Okazaki, Crfsuite: a fast implementation of conditional random fields (crfs), 2007. URL: http://www.chokkan.org/software/crfsuite/.

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.

Michael Kaiser is a Computer Science Student at the University of Innsbruck (Austria). In 2017 he obtained his B.Sc. degree from the University of Innsbruck (Austria). His main research interests include game theory, consensus algorithms, blockchain technology and artificial intelligence. He is also involved, with different roles, in some companies with a particular focus on blockchain technology.

Manuel Neurauter is a Ph.D. candidate at the University of Innsbruck (Austria). Manuel received his M.Sc. degree from the Department of Psychology, University of Innsbruck in 2013. His main research interest is the influence of human cognition and cognitive load on the design process of process models. His research interests further include the influence of cognitive abilities in general and particularly working memory functions on business process quality. Manuel has published eight papers in international journals, conferences, and workshops.

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. In addition, she holds a part-time full professor position at the Department of Applied Mathematics and Computer Science with the Technical University of Denmark. 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.