Preprint of 2018 Conference on Designing Interactive Systems (DIS '18)

Measuring the Learnability of Interactive Systems Using a Petri Net Based Approach

Andrea Marrella
Sapienza University of Rome, Italy
marrella@diag.uniroma1.it

Tiziana Catarci
Sapienza University of Rome, Italy
catarci@diag.uniroma1.it

ABSTRACT

A learnable system allows a user to know how to perform correctly any task of the system after having executed it a few times in the past. In this paper, we propose an approach to measure the learnability of interactive systems during their daily use. We rely on recording in a user log the user actions that take place during a run of the system and on replaying them over the system interaction models, which describe the expected ways of executing system tasks. Our approach identifies deviations between the interaction models and the user log and assesses their weight through a fitness value. By measuring the rate of the fitness value for subsequent executions of the system we are able not only to understand if the system is learnable with respect to its tasks, but also to quantify its degree of learnability over time and to identify potential learning issues.

Author Keywords

Learnability; Usability; Quantitative Method; Interaction Design; Human-Computer Dialog; Petri nets; HCI Theory

INTRODUCTION

This paper is concerned with the use of a well-established technique for modelling *human-computer dialogs*, namely *Petri nets* [25], to objectively quantify the (extended) *learnability* of interactive systems during their daily use.

In the Human-Computer Interaction (HCI) community, *learnability* is generally recognized as one of the most relevant components of usability [26]. Learnability is addressed in ISO 9126-1 as "the capability of the software product to enable the user to learn its application" and applies to the nature of the performance change of a user when interacting with a system to accomplish a specific task. Basically, a highly learnable system should allow a user to automatically know *how to perform correctly* any relevant task of the system after having executed it a few times in the past.

Starting from this general idea of learnability, the work [12] surveys the existing learnability research spanning over the past four decades, and shows that there is *no consistent agreement* on how learnability of a system should be defined, evaluated and improved. Several accepted metrics exist for measuring learnability, but they are scattered across various research papers and limited to measure specific aspects of the interaction. A classification of those metrics is presented in [12]. Some of them suffer from *evaluation subjectivity*, as they rely on associating learnability to the "quality" of the interaction through a score given by an external evaluator or by analyzing the user feedbacks after the interaction has happened [22, 20, 11]. *Mental metrics* – though more abstract and complex to be evaluated – are used to understand which cognitive processes drive the user behavior during the interaction with the system, in order to make it more learnable [31, 29, 33]. Finally, there are also *quantitative* metrics that allow to measure in an *objective* way the performances of a user executing a relevant task through the system (such as completion times, error rates and percentage of functionality understood) [26, 23, 7].

Since all the above measurements are mainly performed in (controlled) lab environments under the guidance of an external evaluator, they have been proven to be particularly suitable for measuring the first time interaction with a system (i.e., the *initial learnability*). Although the assessment of initial learnability is considered as a crucial step to the study of a system, it represents (at best) a measure of the system's intuitiveness. Measuring intuitiveness is not necessarily an indicator of continued learning [26], and it does not provide the evaluator with information about the *extended learnability* of the system over a longer period of use. However, the major obstacle of evaluating extended learnability is that it traditionally requires expensive and time-consuming techniques for observing users over an extended period of time. Consequently, due to its prohibitive costs, evaluators have gradually refrained from performing extended learnability evaluation [31].

In this paper we present an approach to *objectively* and *automatically quantify* the *extended learnability* of a system during its daily use without the need of observing directly the interaction of the user with the system. Our approach is based on verifying whether the user's "observed" behavior, which is recorded in a specific *user log* and reflects the basic user actions performed during the interaction with the system while executing a relevant task, matches the "intended" behavior represented as a *model of the interaction* itself. To tackle this issue, we introduce the concept of *alignment*, which is a pairwise matching between user actions recorded in the log and actions allowed by the interaction model. A perfect alignment between the log and the model is not always possible, thus making deviations be highlighted. Such deviations may reflect mistakes or slips made by the user during the interaction with the system, or may simply represent different strategies (with respect to the ones expected) that a user follows to accomplish a task. The result of the alignment task is the so-called *fitness value*, i.e., how much the log adheres to a model of the interaction. The fitness value can vary from 0 to 1. A fitness value equals to 1 means a perfect matching between the log and the interaction model, i.e., the user was able to accom-

plish her/his task as foreseen in the corresponding interaction model. Viceversa, a fitness value between 0 and 1 pinpoints the presence of some deviations in the log with respect to the interaction model.

Our thesis is that in order to measure the extended learnability of an interactive system we can analyze the rate of the fitness values corresponding to subsequent executions of the system over time. An increasing rate will correspond to a system that is easy to be learnt with respect to its relevant tasks. Vice versa, given an initial low fitness value, a not-increasing or stable rate may indicate the presence of some learning issues that need to be investigated and fixed. Notably, our approach is able to take into account the *severity of deviations* when computing fitness. This allows us to evaluate the learnability of a relevant task of the system on the basis of the level of user's experience with the system, i.e., we can properly quantify learnability distinguishing between novice and experienced users.

Despite in the HCI literature many notations have been proposed for describing human-computer dialogs [28], to realize our technique we opted for *Petri nets* [25], which have a clear semantics and have proven to be adequate for modelling interaction models [28, 2, 27]. In order to perform the alignment task between user logs and interaction models represented as Petri nets, we customized an existing technique coming from the Business Process Management (BPM) context [1, 8], and we developed (on top of it) a proof-of-concept software tool that implements our approach to measure learnability.

The rest of the paper is organized as follows. In Section "Background", we first analyze the existing learnability definitions presented over the years in the HCI research literature. This allows us to clearly position our approach on the basis of the learnability features it enables to calculate. Then, we discuss prior works on modeling human-computer dialogs and we motivate the choice of Petri nets for designing interaction models. In Section "Preliminaries" we provide the relevant background on Petri nets and alignment necessary to understand the paper. In Section "General Approach" we describe our approach to quantify the extended learnability of interactive systems. Then, Section "Experiments" presents a preliminary longitudinal study performed with real users to assess the effectiveness and the feasibility of our approach. Finally, Section "Discussion and Conclusion" concludes the paper by providing a critical discussion about the general applicability of the approach and tracing future work.

BACKGROUND

Learnability Definitions

The interest in assessing learnability of interactive systems is acknowledged by the presence of several studies that are available (from more than three decades) in the HCI literature. Such studies have investigated the concept of learnability through different definitions and characterizations. While the use of the term learnability as an aspect of usability can be traced back to 1976 (cf. [19]), one of the first concrete attempts to define learnability was provided by Michelsen et al., which state that "a system should be easy to learn by the class of users for whom it is intended" [22]. While this definition is rather vague (the fact that a system should be "easy to learn" leaves room for many interpretations), it emphasizes the need to evaluate learnability considering the *level of user's experience* with a specific system.

According to [12], the majority of learnability definitions are categorized on the basis of their focus on: *(i)* the initial performance of a user interacting with a system (*initial learnability*), or *(ii)* the changes of such performances over time, after several interactions with the system (*extended learnability*).

On the one hand, there exists a bunch of definitions that only consider the initial learning experiences. For example, Nielsen advocates that an interactive system is highly learnable if it "allows users to reach a reasonable level of usage proficiency within a short time" [26]. Similarly, Holzinger defines learnability as "allowing users to rapidly begin to work with the system" [14] and Santos recognizes learnability as "the effort required for a typical user to be able to perform a set of tasks using an interactive system with a predefined level of proficiency" [31]. Finally, Shneiderman defines learnability as "the time it takes members of the user community to learn how to use the commands relevant to a set of tasks" [32].

On the other hand, there are definitions that consider learnability not just in the very first interactions with a system, but as a results of several interactions in a longer time frame. For example, in [10] authors define learnability as: "the ease at which new users can begin effective interaction and achieve maximal performance". Butler defines learnability as "Initial user performance based on self instruction [allowing] experienced users to select an alternate model that involved fewer screens or keystrokes" [5], while Rieman as "minimally useful with no formal training, and should be possible to master the software" [30]. Bevan and Macleod state that learnability is the "quality of use for users over time" [4].

The above definitions confirm that there is no unified specification of the concept of learnability; nonetheless, some common features that are inherently related to it can be identified:

- learnability is a *function of user's experience*, i.e., it depends on the type of user for which the learning occurs;
- learnability can be evaluated on a single usage period (*initial learnability*) or after several usages (*extended learnability*);
- learnability measures the *performances* of a user interaction, in terms of completion times, error and success rates or percentage of functionality understood.

While the objective of this paper is not to provide a complete survey of how learnability is evaluated or to review the particular strategies that people use to learn (to this aim, interested readers can refer to other works such as [30, 12]), according to the above features the ability of our approach is to *objectively quantify the extended learnability of the relevant tasks of a system by measuring the error and success rates (and, implicitly, the percentage of functionality understood) during the user interactions on the basis of the level of user's experience.*

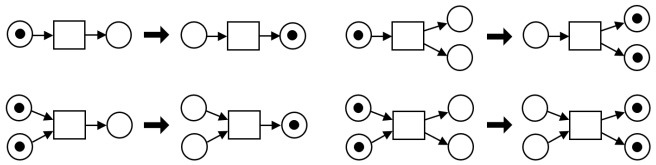


Figure 1. Petri net operational semantics.

Prior Works on Modeling Human-Computer Dialogs

The HCI literature is rich of notations for expressing human-computer dialogs that allow to see at a glance the structure of a dialog. Existing notations can be basically categorized in two main classes: *diagrammatic* and *textual*. Diagrammatic notations include (among the others[1]) various forms of state transition networks (STNs) [38], Petri nets [25], Harel state charts [13], flow charts [2], JSD diagrams [34] and ConcurTaskTrees (CTT) [24]. Textual notations include regular expressions [35], Communicating Sequential Processes (CSPs) [9], GOMS [18], BNF and production rules [17].

If, on the one hand, some notations are essentially equivalent (for example, JSD diagrams and regular expressions can describe exactly the same dialogs), on the other hand there are major differences in expressive power between different notations. In particular, according to [2], one important difference in expressive power is whether or not the formalism can handle concurrent dialogs. For example, STNs support sequential dialogs, production rules favor concurrent dialogs, but few could handle both (the exceptions being Harel state charts, CSPs, CTT and Petri nets). However, increased expressive power is not always desirable as it may mean a more complex and less easily understood description, i.e., the dialog of a user interface can become unmanageable.

We chose to represent interaction models as Petri nets because we believe they represent the right trade-off between expressive power and understandability of the models. Petri nets may contain exclusive choices, parallel branches and loops, allowing the representation of extremely complex behaviours in a very compact way. For example, simple Petri nets like the one in Figure 2 (we will thoroughly discuss it in the next section) allows to capture an infinite number of different behaviours (due to the presence of a loop), which can potentially all lead to a perfect execution of a relevant task. The definition of such complex behaviours (in particular, in the case of "undo-redo" actions, which require the presence of loops) would be simply impossible with other notations. Last but not least, Petri nets provide a formal semantics, which allow to interpret the meaning of an interaction model unambiguously.

PRELIMINARIES

In this section, we provide some preliminary concepts used throughout the paper. First of all, we introduce the Petri Net modeling language. Then, we describe the problem of computing alignments between user logs and interaction models represented as Petri nets.

[1]Interested readers can refer to [28, 2] for a detailed description of the existing notations for modeling human-computer dialogs.

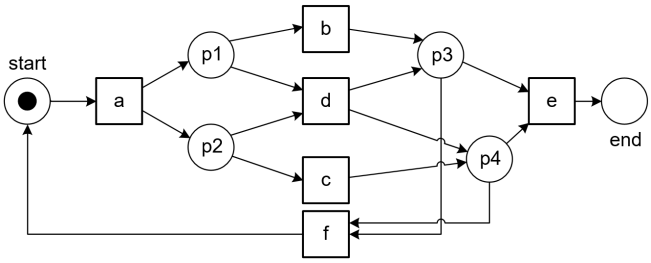


Figure 2. The Petri net used as running example.

Petri Nets

A Petri net [25] is a directed graph with two node types called *places* and *transitions*. Places are represented by circles and transitions by rectangles. Connections between two nodes of the same type are not allowed.

DEFINITION 1 (PETRI NET). *A Petri net is a tuple* (P,T,F) *where*

- P *is a finite set of places;*
- T *is a finite set of transitions;*
- $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation between places and transitions (and between transitions and places).*

Given a transition $t \in T$, ${}^\bullet t$ is used to indicate the set of *input places* of t, which are the places p with a directed arc from p to t (i.e., such that $(p,t) \in F$). Similarly, $t^\bullet$ indicates the set of *output places*, namely the places p with a direct arc from t to p. At any time, a place can contain zero or more *tokens*, drawn as black dots. The state of a Petri net, also known as *marking*, is determined by the number of tokens in places. Therefore, a marking m is a function $m : P \to \mathbb{N}$.

In any run of a Petri net, its marking may change according to the following enablement and firing rules:

- A transition t is **enabled** at a marking m iff each input place contains at least one token: $\forall\, p \in {}^\bullet t,\ m(p) > 0$.
- A transition t can **fire** at a marking m iff it is enabled. As result of firing a transition t, one token is "consumed" from each input place and one is "produced" in each output place. Hence, firing a transition t at marking m leads to a marking m', and this is denoted as $m \xrightarrow{t} m'$.

Figure 1 illustrates the act of firing for various Petri net configurations. It is assumed that the firing of a transition is an atomic action that occurs instantaneously and can not be interrupted. Within the salient features of Petri nets is the fact that several enabled transitions can not fire simultaneously, and that an enabled transition is not forced to fire immediately but can do so at a time of its choosing. These features make Petri nets particularly suitable for modeling concurrent executions such as those that occur in *interaction models* [2]. In the remainder, given a sequence of transition firing $\sigma = \langle t_1, \ldots, t_n \rangle \in T^*$, $m_0 \xrightarrow{\sigma} m_n$ is used to indicate $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$.

When Petri nets are used to represent interaction models, transitions are associated with the *user actions* (e.g., windows

opened, system commands executed, check boxes clicked, text entered/edited, etc.) required to accomplish a *relevant task* (e.g., copy and paste a document) of an *interactive system* (e.g., a text editor). Therefore, we can define as many interaction models as are the relevant tasks of the system. Since the executions of a relevant task have a *start* and a *end*, Petri nets need to be associated with an *initial* and *final marking*.

DEFINITION 2 (LABELLED PETRI NET). *A Labelled Petri net is a tuple* (P,T,F,A,ℓ,m_i,m_f) *where*

- (P,T,F) *is a Petri net;*
- A *is the universe of user actions allowed by the system;*
- $\ell : T \to A$ *is a function that associates a transition in* T *with a user action in* A*;*
- m_i *and* m_f *are the initial and final marking.*

Figure 2 shows an example of a Labelled Petri net, i.e., of an interaction model representing a fictitious relevant task. The markings with respectively one token in place *start* or in place *end* are the initial and final marking (and no tokens in any other place). Transitions *a*, *b*, *c*, *d*, *e* and *f* are associated with labels, i.e., user actions, of the same name.

In the remainder of this paper, we assume all Petri nets to be labelled and 1-bounded, also known as *safe*. A Petri net is 1-bounded if in any reachable marking from the initial marking m_i, no place ever contains more than 1 token. One-boundness is not a large limitation as the behavior allowed by interaction models can be represented as 1-bounded Petri nets [2].

Alignment between User Logs and Petri Nets

User logs are the starting point to quantify the learnability of interaction models. A user log is a multi-set of *traces*. Each trace consists of a sequence of *user actions* related to the *single execution* of a specific relevant task.

DEFINITION 3 (USER LOG). *Let* N *be a Labelled Petri net and* A *be the universe of user actions. A trace* $\sigma_L \in A^*$ *is a finite sequence of user actions. A user log* L *over* N *consists of a multi-set* $L \in \mathbb{B}(A^*)$[2] *of traces.*

Multiple executions of a relevant task may consist of the same sequence of transition firings and, hence, result in the same trace. This motivates the definition of a user log as a multi-set. The following $[\langle a,b,c,e\rangle, \langle a,b,c,e\rangle, \langle a,b,c,f,a,d,e\rangle, \langle a,d\rangle, \langle a,d\rangle, \langle a,d,e\rangle]$ is an example of user log, assuming a universe of user actions $A = \{a,b,c,d,e,f\}$. The concept of time is usually explicitly modeled in a way that user actions in a trace are sorted according to the timestamp of their occurrence.

As mentioned in Section "Introduction", the starting point for measuring learnability is to construct an *alignment* of user log L and interaction model N, which allows us to exactly pinpoint where deviations occur. On this aim, the actions in the user log need to be related to transitions in the model, and vice versa. In [1], an approach to perform the alignment task is realized by relating "moves" in the log to "moves" in the model. However, it may be that some of the moves in the log can not be matched by the model and vice versa. We explicitly denote such "no moves" by $*$.

[2]Given a set X, $\mathbb{B}(X)$ is the set of all multisets constituted by elements of X.

$$\gamma_1 = \begin{array}{|c|c|c|c|c|}\hline a & f & b & c & * \\ \hline a & * & b & c & e \\ \hline \end{array} \qquad \gamma_2 = \begin{array}{|c|c|c|c|c|c|}\hline a & * & f & b & c & * \\ \hline a & d & * & * & * & e \\ \hline \end{array}$$

Figure 3. Alignments of trace $\langle a,f,b,c\rangle$ and the interaction model in Figure 2.

DEFINITION 4 (LEGAL ALIGNMENT MOVES). *Let* $N = (P,T,F,A,\ell,m_i,m_f)$ *be a Labelled Petri net and* L *a user log. A* legal alignment move *for* N *and* L *is represented by a pair* $(s_L,s_M) \in (A \cup \{*\} \times T \cup \{*\}) \setminus \{(*,*)\}$ *such that:*

- (s_L,s_M) *is a move in log if* $s_L \in A$ *and* $s_M = *$*,*
- (s_L,s_M) *is a move in model if* $s_L = *$ *and* $s_M \in T$*,*
- (s_L,s_M) *is a synchronous move if* $s_L \in A$*,* $s_M \in T$ *and* $s_L = \ell(s_M)$

An *alignment* is a sequence of alignment moves:

DEFINITION 5 (ALIGNMENT). *Let* L *be a user log and* $N = (P,T,F,A,\ell,m_i,m_f)$ *a Labelled Petri net. Let* Γ_N *be the universe of all legal alignment moves for* N *and* L*. Let* $\sigma_L \in L$ *be a log trace. Sequence* $\gamma \in \Gamma_N^*$ *is an* alignment *of* N *and* σ_L *if, ignoring all occurrences of* $*$*, the projection on the first element yields* σ_L *and the projection on the second yields a sequence* $\sigma'' \in T^*$ *such that* $m_i \xrightarrow{\sigma''} m_f$*.*

Basically, an alignment consists of replaying an input trace against the interaction model (one action at a time) to find the *closest corresponding trace* that can be parsed by the model. The output is a pair of aligned traces together with the points of divergence between these two traces. Specifically, a pair shows a trace in the log that does not match exactly a trace in the model, together with the corresponding closest trace produced by the model. Many alignments are possible for the same trace. For example, Figure 3 shows two possible alignments for a trace $\sigma_1 = \langle a,f,b,c\rangle$. Note how moves are represented vertically. For example, as shown in Figure 3, the first move of γ_1 is (a,a), i.e., a synchronous move of a, while the the second and fifth move of γ_1 are a move in log and model, respectively.

In order to find an alignment of σ_L and N with minimal deviation cost, it is possible to define the *severity of a deviation* by introducing a *cost function* on legal alignment moves.

DEFINITION 6 (COST FUNCTION). *Let* $N = (P,T,F,A,\ell,m_i,m_f)$ *be a Labelled Petri net and* $\sigma_L \in A^*$ *a log trace, respectively. Assuming* Γ_N *as the set of all legal alignment moves, a* cost function κ *assigns a non-negative cost to each legal move:* $\Gamma_N \to \mathbb{N}_0^+$*. The* cost of an alignment $\gamma \in \Gamma_N$ *between* σ_L *and* N *is computed as the sum of the cost of all constituent moves:* $\mathscr{K}(\gamma) = \sum_{(s_L,s_M)\in\gamma} \kappa(s_L,s_M)$*.*

The definition of a cost function depends by the specific relevant task of interest and can be used to evaluate the severity

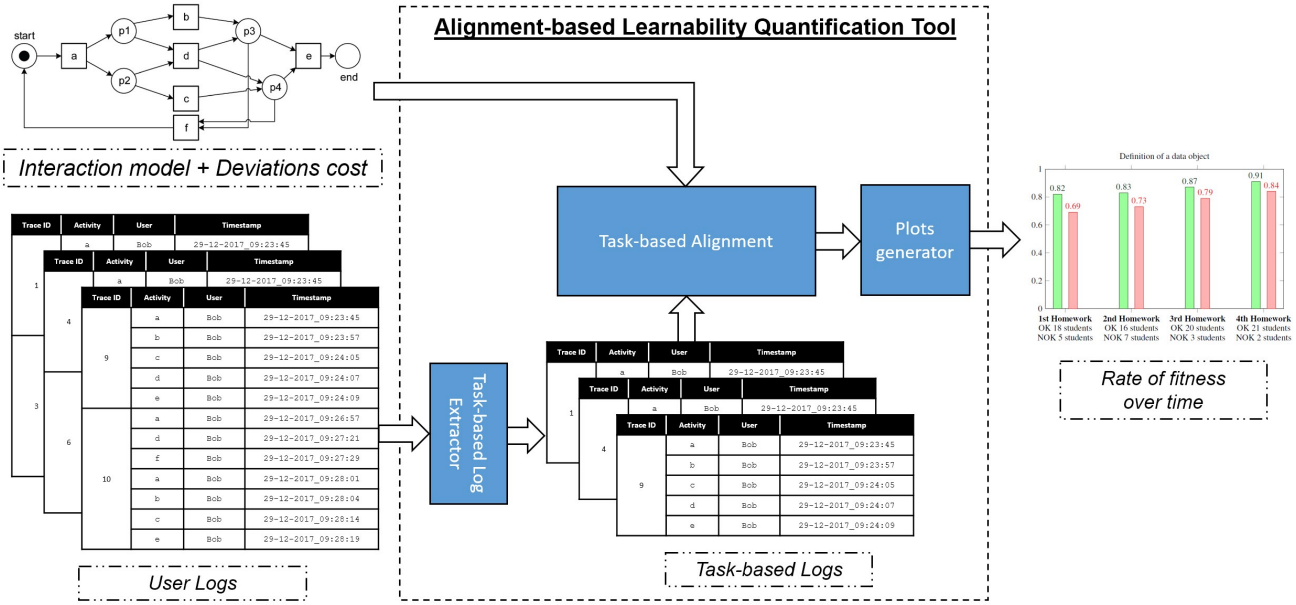


Figure 4. An overview of the general approach to alignment-based learnability quantification.

of the different sorts of nonconformity. For instance, in a graphical editing tool it is significantly more severe to cut a picture when not supposed than to push the "Help" button and ask for additional information. In case that a customized cost function can not be provided, one can use a *standard cost function*, which assigns cost 1 to every move in log or move in model and cost 0 to synchronous moves.

$$\kappa\big((s_L, s_M)\big) = \begin{cases} 1 & \text{if } s_M = *, \\ 1 & \text{if } s_L = *, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Let us consider the alignments in Figure 3. Using the standard cost function in Equation 1, the cost of alignment γ_1 is 2 (i.e., $\mathscr{K}(\gamma_1) = 2$) since it has two moves in model and/or log. Conversely, γ_2 takes on a cost 5 (i.e., $\mathscr{K}(\gamma_2) = 5$). So, γ_1 is a less expensive alignment. Since no alignment exists with cost less or equal than 1, γ_1 is among the least expensive alignments. Therefore, γ_1 is an *optimal alignment*, i.e., an alignment with minimal deviation cost, since for any alignment γ' of N and σ_L, we have that $\mathscr{K}(\gamma_1) \leq \mathscr{K}(\gamma')$.

GENERAL APPROACH

In this section, we describe the main ingredients of our approach to measure the learnability of an interactive system.

First of all, as shown in Figure 4, it is required to formalize the potential dialogs between the user and the system by employing dedicated Petri nets as interaction models. An interaction model represents the *expected way* to perform a relevant task of the system. Interestingly, an interaction model may allow *different strategies* to perform a relevant task. For example, if we consider the Petri net in Figure 2, the number of ways to properly reach the final marking are potentially unbounded (due to the presence of a loop in the model). For instance, the execution traces $[\langle a,b,c,e\rangle, \langle a,b,c,f,a,b,c,f,a,d,e\rangle]$ represent (both) good ways to execute the relevant task underlying the Petri net in Figure 2. This basically means that a same relevant task can be completed through different paths of user actions in the interface with equivalent results, as may happen in real user interfaces. In addition, we can customize the *severity of the deviations* based on the kind of user that is interacting with the system, i.e., we can assign specific *cost functions* on legal alignment moves to distinguish between novice and experienced users.

Secondly, the system to be evaluated must be able to capture all the user actions taking place during a single execution of the system and record them in a specific *user log*. As extensively discussed in the previous section, a user log consists of a sequence of user actions potentially related to the execution of several relevant tasks. However, in order to quantify the learnability of the system with respect to a specific relevant task, we need to extract from the user log only those user actions that are related to the execution of the relevant task under consideration. To identify such user actions, we detect the *entry/exit points* of the relevant task in the user log. According to that, a *trace* is a complete list of actions performed between an entry point and exit point of the relevant task. All the traces associated to a relevant task are clustered in a *task-based log*. In a nutshell, a task-based log will contain all the traces associated to a specific relevant task and extracted during a single run of the system.

As shown in Figure 5, to perform correctly the above extraction it is important that the actions in the user logs are related to transitions in the interaction model, and vice versa. For example, given the interaction model of Figure 2, a possible trace extracted from a user log can be $\gamma = \langle a,b,c,d,e\rangle$, being a and e the entry/exit points of the task. Notice that in the context of a single run of the system a relevant task can be executed several times, i.e., several traces related to the relevant task may be extracted from the user log. During the extraction,

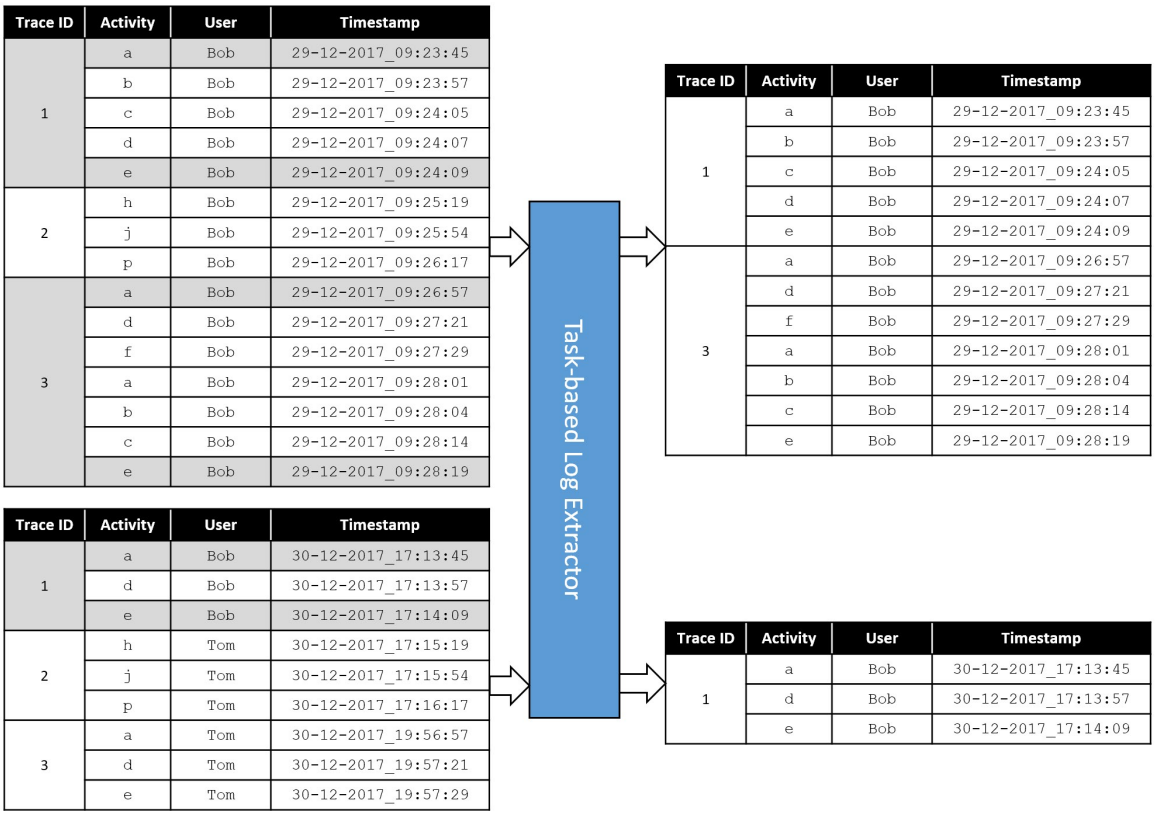


Figure 5. Extraction of task-based logs from user logs.

it is also possible to focus just on the interactions performed by a specific user. For example, in Figure 5 it is shown that only user interactions (i.e., execution traces) performed by user "Bob" and related to the interaction model of Figure 2 are extracted and injected into specific task-based logs.

We notice that our approach requires an extra modelling effort to the design of the interaction models at the outset. While we think that this effort is comparable to the design and evaluation of traditional usability tests that measure learnability, being our approach based on Petri Nets, we can leverage on the huge literature on Petri Nets discovery (cf. [36]) to automatically derive interaction models from a set of user logs. However, since such discovered models reflect real task executions, they may contain/miss (un-)necessary user actions with respect to the task expected behaviour. Consequently, the discovered models are usually rough versions of the final interaction models and they should be refined by the evaluators to obtain the desired ones. Nonetheless, we believe that to automatically discovering a base structure for the interaction models allows to strongly mitigating the design effort.

Thirdly, given a relevant task for which we developed an interaction model and a task-based log associated to it, we can construct the *alignment* between any of the traces extracted from the log and the interaction model itself. As detailed in the previous section, the alignment activity consists of *replaying* the extracted traces over the interaction model. For each trace, we can derive the alignment with the lowest cost, i.e., the optimal alignment, which allows us to infer the *fitness value*.

The fitness is a metric that reflects the extent to which the traces of a task-based log can be associated with valid execution paths specified by the interaction model. To be more specific, the fitness value quantifies the amount of deviations between a trace and an interaction model. A deviation can manifest itself in skipping actions prescribed by the interaction model (i.e., move in model), or in inserting actions, namely, some actions that do not appear in the model can be executed and logged (i.e., move in log). Then, the fitness takes also into account the severity of a deviation. For instance, pushing the "Help" button during the execution of a relevant task to access to the help features of the system should not be punished too hard: it might have been unnecessary but it does not imply any concrete violation of the interaction model and may eventually improve the learnability of the system.

In the BPM research literature, there are several ways to compute such a fitness value. In this paper, we rely on the definition of fitness provided in [37]. Specifically, given a Petri net N and a task-based log L, to quantify the fitness of any trace $\sigma_L \in L$ with respect to N (whose optimal alignment is γ_{σ_L}), we can apply the cost function $\mathscr{K}$ as follows:

$$fitness(N,L) = 1 - \frac{\Sigma_{\sigma_L \in L} \mathscr{K}(\gamma_{\sigma_L})}{move_L(L)+|L|+move_M(N)} \qquad (2)$$

The purpose of Equation 2 is to normalize the cost of all optimal alignments of the traces in L (see numerator) with

the worst case scenario, which consists of just performing moves in model and moves in log and no synchronous move in the optimal alignment. This can be expressed through $move_L(L)$ and $move_M(N)$, which represent the total cost of moving through the whole log (or model, resp.) only. The outcome the fitness value is a number that may vary between 0 (very poor fitness) to 1 (perfect fitness).

At this point, given several task-based logs representing subsequent executions of a relevant task over time, we can compute the fitness values associated to any of these logs and calculate the *rate* of such values. *Our approach allows to quantify the extended learnability of an interactive system by analyzing the rate of the fitness values over time*. Specifically, given an initial fitness value, an increasing rate (or a stable rate if the initial fitness is high) corresponds to an interactive system that is *easy to be learnt* with respect to the relevant task under observation. Conversely, given an initial low fitness value, a not-increasing or stable rate may indicate the presence of some learning issues that need to be fixed. To this aim, the notion of alignment [1, 8] provides a robust technique to pinpoint the deviations and to quantify the percentage of functionality understood in a user's interaction with the system.

EXPERIMENTS

We have developed a proof-of-concept software tool that implements our approach to measure the extended learnability of interactive systems. The tool exploits an existing plugin of ProM[3] developed in [1], which allows to compute optimal alignments between Petri nets and log traces and to derive fitness values as expressed in Equation 2. Our tool takes in input an interaction model represented as a Petri Net and a set of user logs recording subsequent executions of the system over time. The tool supports the standards XES (eXtensible Event Stream) and PNML (Petri Net Markup Language) for respectively storing and analyzing user logs and Petri nets. The tool is able to extract from the user logs the task-based logs related to the interaction model (i.e., relevant task) of interest. Then, it leverages on the features implemented in [1] to calculate the fitness values associated to the single task-based logs. Finally, as shown in the right part of Figure 4, the tool automatically produces a plot showing the rate of the fitness values over time. In addition, the tool is able to exploit the graphical features of ProM to detect and visually locate deviations (e.g., overlaid on top of the interaction model) and explain their severity (e.g., user action x is executed multiple times in the user log, but this is not allowed according to the interaction model).

To test the feasibility and the effectiveness of our approach, we employed our tool to perform a preliminary *longitudinal study* with real users using the SmartPM system [21]. SmartPM is a BPM environment that provides a set of Artificial Intelligence (AI) techniques and tools to build recovery procedures on-the-fly for adapting running processes when unanticipated exceptions and exogenous events occur at run-time. In SmartPM, a GUI-based tool has been developed to support the process design activity. At design-time, a human process designer can enter knowledge on processes without being an expert of the internal working of the AI tools involved in the system. To this aim, the GUI of SmartPM provides several wizard-based editors that assist the process designer in the definition of the process knowledge (i.e., data objects, process variables, process activities with preconditions and effects, etc.), and a graphical editor to design the control flow of a business process using a relevant subset of BPMN[4]. A screenshot of the workspace of the GUI is provided in Figure 6(a). Once the design activity is completed, a process can be enacted with the execution engine of SmartPM, which provides automated adaptation features at run-time.

Our experiment consisted of *measuring the learnability of some relevant tasks* of the GUI of SmartPM against the complexity of modeling realistic BPM environments and processes with the tool. To this aim, we performed a longitudinal study with 23 Master students in Engineering in Computer Science during a university course held at our university. After a preliminary training session to describe the usage of the tool, we provided to the students 4 different homeworks of growing complexity in 4 consecutive weeks (one homework per week); each homework was targeted to model the domain and the control flow of a realistic process through the GUI of SmartPM. The students were requested to complete the homework assigned to them in a specific week within the end of the week itself. Before the assignment of a new homework, we shown to the students the minimal optimal solution to perform correctly the previous homework.

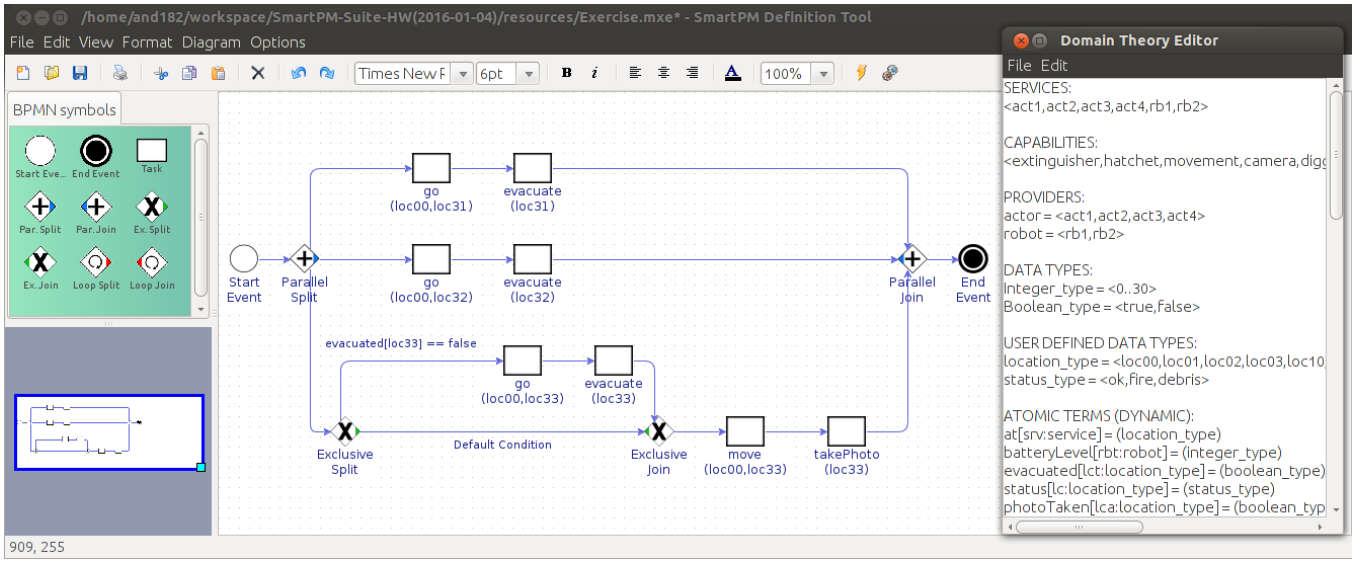
To apply our approach, we recorded in a specific log all the user actions performed by the students during their interaction with SmartPM. Then, to assess the learnability of the system, we replayed such logs over three *interaction models* of the system describing the expected ways to achieve three relevant tasks: *(i)* definition of a new data object, *(ii)* generation a new atomic term and *(iii)* creation of a new task specification.

For example, in Figure 6(b) it is shown one of the wizard-based editor provided by the GUI of SmartPM, specifically the one to create a new atomic term. In the dialect of SmartPM, an atomic term is a process variable. As depicted in Figure 6(b), after having defined the name and the type of the atomic term, there exist three possible expected pipelines to create a new atomic term in the best way possible:

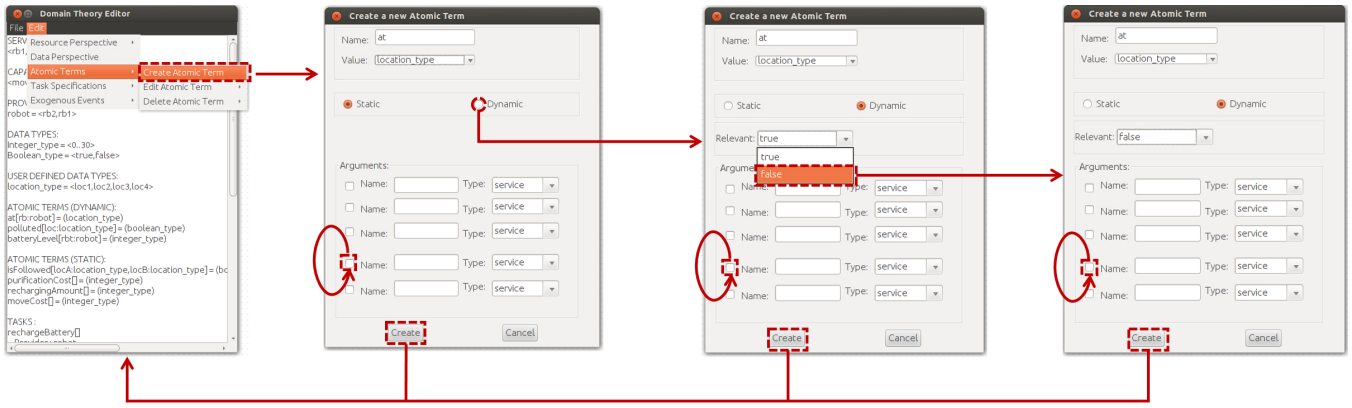
- if the atomic term to be created is *static* (i.e., it can not be changed during process execution), the user can simply creates it by pushing the dedicated button "Create";
- if the atomic term to be created is *dynamic* (i.e., its value may change during process execution), the user pushes the dedicated radio button to set the atomic term as dynamic and then creates it;
- if the atomic term to be created is *dynamic* and *relevant* (i.e., its value must be monitored at run-time to trigger process adaptation), the user selects the dedicated item from a combo box and then creates the atomic term.

[3]ProM (http://www.promtools.org/) is an is a open-source framework for implementing process mining tools and algorithms.

[4]The Business Process Modeling Notation (BPMN) is the standard language (ISO/IEC 19510:2013) for modeling business processes



(a) The workspace provided by the GUI of SmartPM.



(b) The wizard-based editor to build atomic terms.

Figure 6. Some screenshots of the GUI of SmartPM.

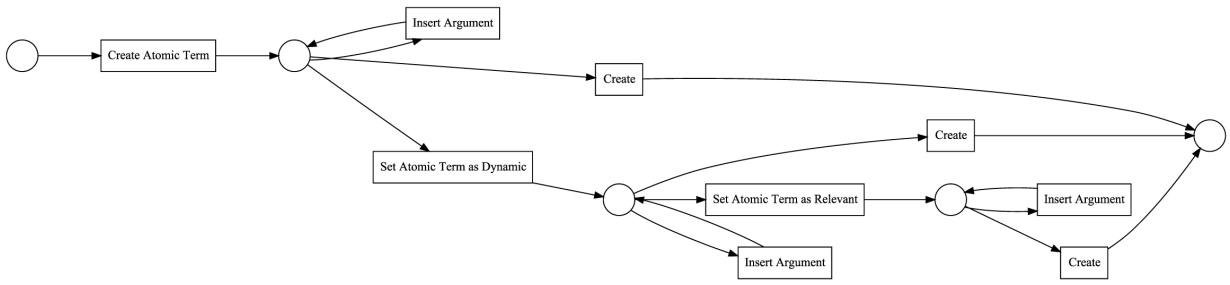


Figure 7. The Petri net that formalizes the expected behavior to build new atomic terms.

Notice that in any of the above three cases, a user can add new arguments to the atomic term. Such a behaviour is formally captured by the interaction model in Figure 7. The fact that some potential user actions are not captured by the interaction model (for example, the possibility of removing arguments from the atomic term) does not mean that such neglected actions are wrong, but just that they are irrelevant for a perfect creation of an atomic term. In fact, if we like to create an atomic term without arguments, if we remove an argument is because we previously added it erroneously. Therefore, the presence of a user action in a trace log that is associated to the deletion of an argument may lead to a non-perfect alignment between the trace and the interaction model, i.e., the fitness value will be lower than 1.

The complete results of the experiments are provided in Figure 8. Collected data are organized in 3 diagrams related to

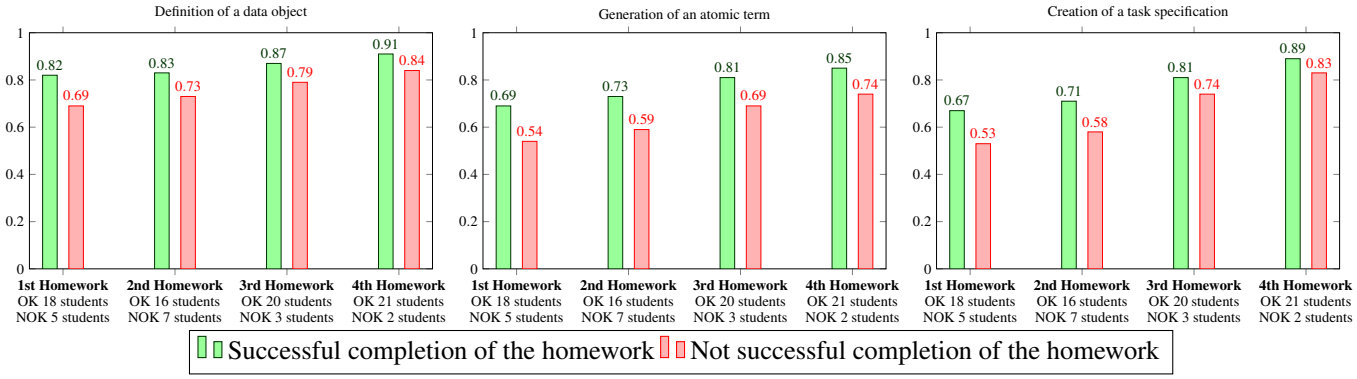


Figure 8. Analysis of the learnability of the GUI of SmartPM.

the completion of the three above relevant tasks. For any diagram, the x-axis indicates the specific homework considered, while the y-axis indicates the average fitness value obtained to complete the relevant task. Notice that for each homework we represent two bars for separating the students that performed correctly the whole homework from the students that were not able to complete it or to compute a correct solution.

The analysis of the performed experiments points out some interesting aspects. For example, let us consider the first diagram in Figure 8, that shows the fitness values related to the definition of a new data object. First of all, it is evident that both the rate of the fitness values and the number of students able to correctly complete an homework increase over time, i.e., homework after homework. In fact, while 6 students out of 23 were not able to complete the first homework (around 26% of failure), the third and the fourth homework were correctly completed by 20 and 21 students (percentage of success of 86.95% and 91.3%). It is interesting to notice a "discontinuity" of this trend between the first and the second homework, probably caused by the first relevant increase of complexity.

Concerning the fitness value, we can notice that it is always higher for those students that completed correctly the whole homework, even if this gap slightly decreases in the subsequent homeworks. The above considerations are valid also for the other diagrams, even if the fitness values emphasize that the definition of a data object is a simpler task if compared with the generation of an atomic term or the creation of a task specification. However, also in this case, this "complexity gap" seems to decrease over time.

To sum up, the results obtained allow to observe an increasing rate of the fitness value over time, which – according to our approach – suggests that the students performing the homeworks were able to efficiently *learn* the usage of the GUI of SmartPM, without any knowledge of the AI technologies employed in the system.

These results are also supported by a qualitative evaluation performed after the completion of each homework. Specifically, before the starting of a new homework, we organized a dedicated focus group by involving 3 students randomly selected from the ones that correctly completed the previous homework and 3 students that were not able to complete it (except in the case of the fourth homework, where the ratio was 4:2). Then, we explicitly asked them to replicate the homework (in a controlled environment with an external evaluator observing them) indicating the modeling issues found while interacting with the GUI of SmartPM. This further test allowed us to double-check and confirm the validity of the results obtained through our learnability approach.

DISCUSSION AND CONCLUSION

In this paper, we have presented an approach for objectively quantifying the extended learnability of interactive systems in an automated way. We used Petri nets to represent interaction models and user logs to record the concrete user interactions with a system. Then, we computed learnability by investigating the extent to which a recorded user interaction trace matches the interaction model. We believe that our approach is timely and can be particularly relevant for the HCI field, as nowadays there is an increasing number of task-based software applications offered through mobile devices [16, 6] or on the web that use to record interaction logs and send such logs back to the developers of the software, who must then make sense of them. Furthermore, understanding exactly what happened during such user interactions is crucial for transparency and explainability of interactive systems [15].

It is undoubtable that there are aspects of the interaction that can not be formalized through Petri nets, including the objects manipulated by user actions and the representation of the users roles. They can certainly be modeled by richer languages, e.g., the standard language CTT (ConcurTaskTrees [24]) for designing interaction sequences or the task modelling language Hamsters [3], which has been applied for evaluating the effectiveness of plane cockpits interfaces. However, in this paper we did not aim at demonstrating that Petri nets are suitable for modeling human-computer dialogs. In fact, despite of their simplicity, it has been already proven that Petri nets are sufficiently adequate to model crucial aspects of an interaction [28, 2, 27]. Conversely, the target of this paper was to devise a very clean and simple-to-manage framework that couples interaction models represented as Petri nets with the notion of alignment, in order to obtain a precise measurement of the extended learnability of an interactive system. To this respect, our approach generates a fitness value that corresponds exactly

to the success/error rate of user interactions measured with traditional learnability approaches.

The strengths of the approach range from the possibility to be enacted in the background as the user concretely uses the system in the real user context (e.g., in the user's working place) to the ability of associating different weights to the deviations identified during an alignment. Furthermore, for a specific relevant task, different interaction models can be developed to represent the different interaction strategies of novice and experienced users. Finally, the granularity of user actions in interaction models can be customized on the basis of the kind of user logs recorded by the system, i.e., the approach is *scalable*. All these aspects make our approach flexible and customizable for several settings and different types of users.

We also notice that our approach takes inspiration from conformance checking techniques in the BPM field [1, 8]. Such techniques are currently employed to detect deviations and frauds in the execution of real-world business processes, whose structure is extremely more complex than the one of relevant tasks in interactive systems. This further emphasizes the (potential) practical relevance of our approach in the HCI field.

The main weakness of the approach is that currently only few interactive systems provide a structured recording of user logs. A second limitation is that - to make the approach work - the relevant tasks have to be predetermined and known, since they require to be explicitly modeled through Petri nets. This limitation can be partially mitigated by employing algorithms for Petri nets discovery (cf. [36]), which allow to automatically derive the base structure of interaction models from a set of user logs and to identify (potential) new relevant tasks of the system. Finally, a third limitation relies on the assumption that forces to explicitly identify, for each relevant task to be analyzed, one or more entry/exit points in the user log for extracting the specific traces associated to the task. This means that, to date, our approach is particularly suitable for evaluating the learnability of wizard-based and structured tasks, such as the ones evaluated in the preliminary longitudinal study presented in Section "Experiments".

As a future work, we are focusing on relaxing the latter assumption, by investigating ways to design interaction models that are less constraining than Petri nets and allow our approach to be applied for less structured and "open-world" tasks. Then, we are also working on designing further robust longitudinal studies to be performed in longer time frames. The target is twofold: *(i)* we want to understand more about the implications on the fitness value when there is a disparity between what is captured in the interaction model and what is captured in the user log, i.e., how robust is our metric in a variety of different scenarios; and *(ii)* we aim at identifying some exact threshold values for the fitness that may help to precisely classify the learnability of an interactive system as "high" or "low". This aspect is crucial to enable our approach to calculate also the initial learnability of a system.

Finally, we like to clarify that we do not think that our approach will substitute the existing learnability metrics, but we rather aim it will be useful to complement them. In fact, on the one hand, our approach is thought to be employed during the daily use of a system, an aspect that makes it completely different from the traditional learnability approaches, which usually require a direct user involvement in controlled (lab) experiments. On the other hand, we allow to automatically quantify the extended learnability of an interactive system, which has proven to be complex and costly to be calculated.

ACKNOWLEDGMENTS

This work is partly supported by the Sapienza grants DAKIP ("Data-aware Adaptation of Knowledge-intensive Processes in Cyber-Physical Domains through Action-based Languages") and IT-SHIRT, and by the Italian projects Social Museum and Smart Tourism (CTN01_00034_23154), RoMA - Resilience of Metropolitan Areas (SCN_00064) and FilieraSicura.

REFERENCES

1. A. Adriansyah, B. F. v. Dongen, and N. Zannone. 2013. Controlling Break-the-Glass through Alignment. In *2013 International Conference on Social Computing*. 606–611.
2. Dix Alan, Finlay Janet, Abowd Gregory, and Beale Russell. 2004. Human-Computer Interaction. *Pearson* (2004).
3. Eric Barboni, Jean-Francois Ladry, David Navarre, Philippe Palanque, and Marco Winckler. 2010. Beyond modelling: an integrated environment supporting co-execution of tasks and systems models. In *Proc. of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 165–174.
4. Nigel Bevan and Miles Macleod. 1994. Usability measurement in context. *Behaviour & information technology* 13, 1-2 (1994), 132–145.
5. Keith A Butler. 1985. Connecting theory and practice: a case study of achieving usability goals. In *ACM SIGCHI Bulletin*, Vol. 16. ACM, 85–88.
6. F. Cossu, A. Marrella, M. Mecella, A. Russo, G. Bertazzoni, M. Suppa, and F. Grasso. 2012. Improving operational support in hospital wards through vocal interfaces and process-awareness. In *2012 25th IEEE Int. Symp. on Computer-Based Medical Systems (CBMS)*.
7. Sid Davis and Susan Wiedenbeck. 1998. The effect of interaction style and training method on end user learning of software packages. *Interacting with Computers* 11, 2 (1998), 147–172.
8. Massimiliano de Leoni and Andrea Marrella. 2017. Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. *Expert Systems with Applications* 82 (2017), 162 – 183.
9. MV Dignum. 2004. *A model for organizational interaction: based on agents, founded in logic*. SIKS.
10. Alan Dix. 1990. Design issues for reliable time-critical systems. *Interacting with Computers* 4, 3 (1990).
11. GJ Elliott, E Jones, and P Barker. 2002. A grounded theory approach to modelling learnability of hypermedia authoring tools. *Interacting with Computers* 14, 5 (2002).

12. Tovi Grossman, George Fitzmaurice, and Ramtin Attar. 2009. A Survey of Software Learnability: Metrics, Methodologies and Guidelines. In *Proc. SIGCHI Conf. Human Factors in Comp. Sys.* ACM.

13. David Harel. 1987. Statecharts: A visual formalism for complex systems. *Science of computer programming* 8, 3 (1987), 231–274.

14. Andreas Holzinger. 2005. Usability engineering methods for software developers. *Commun. ACM* 48, 1 (2005).

15. Andreas Holzinger, Chris Biemann, Constantinos S Pattichis, and Douglas B Kell. 2017. *What do we need to build explainable AI systems for the medical domain?* Technical Report. arXiv preprint arXiv:1712.09923.

16. Shah Rukh Humayoun, Tiziana Catarci, Massimiliano de Leoni, Andrea Marrella, Massimo Mecella, Manfred Bortenschlager, and Renate Steinmann. 2009. The WORKPAD User Interface and Methodology: Developing Smart and Effective Mobile Applications for Emergency Operators. In *Universal Access in Human-Computer Interaction (UACHI)*. Springer.

17. Robert JK Jacob. 1983. Using formal specifications in the design of a human-computer interface. *Commun. ACM* 26, 4 (1983), 259–264.

18. Bonnie E John and David E Kieras. 1996. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)* 3, 4 (1996), 320–351.

19. Joseph CR Licklider. 1976. User-oriented interactive computer graphics. In *Proc. of the ACM/SIGGRAPH workshop on User-oriented design of interactive graphics systems*. ACM, 89–96.

20. Han X Lin, Yee-Yin Choong, and Gavriel Salvendy. 1997. A proposed index of usability: a method for comparing the relative usability of different software systems. *Behaviour & information tech.* 16, 4-5 (1997), 267–277.

21. Andrea Marrella, Massimo Mecella, and Sebastian Sardina. 2016. Intelligent Process Adaptation in the SmartPM System. *ACM Trans. Intell. Syst. Technol.* 8, 2 (2016). DOI:http://dx.doi.org/10.1145/2948071

22. Christie D Michelsen, Wayne D Dominick, and Joseph E Urban. 1980. A methodology for the objective evaluation of the user/system interfaces of the MADAM system using software engineering principles. In *Proc. of the 18th annual Southeast regional conference*. ACM, 103–109.

23. Deborah A Mitta and Sherrill J Packebush. 1995. Improving interface quality: an investigation of human-computer interaction task learning. *Ergonomics* 38, 7 (1995), 1307–1325.

24. Giulio Mori, Fabio Paternò, and Carmen Santoro. 2002. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Trans. Software Eng.* 28, 8 (2002), 797–813.

25. T. Murata. 1989. Petri nets: Properties, analysis and applications. *Proc. of the IEEE* 77, 4 (1989).

26. Jakob Nielsen. 1994. *Usability Engineering*. Elsevier.

27. Philippe A Palanque and Remi Bastide. 1995. Petri Net Based Design of User-Driven Interfaces Using the Interactive Cooperative Objects Formalism. In *Interactive Systems: Design, Specification, and Verification*. Springer.

28. Fabio Paterno. 1999. *Model-Based Design and Evaluation of Interactive Applications* (1st ed.). Springer-Verlag.

29. Tim F Paymans, Jasper Lindenberg, and Mark Neerincx. 2004. Usability trade-offs for adaptive user interfaces: ease of use and learnability. In *Proc. of the 9th Int. Conf. on Intelligent User Interfaces*. ACM, 301–303.

30. John Rieman. 1996. A field study of exploratory learning strategies. *ACM Transactions on Computer-Human Interaction (TOCHI)* 3, 3 (1996), 189–218.

31. Paulo J Santos and Albert Badre. 1995. *Discount learnability evaluation*. Technical Report. Georgia Institute of Technology.

32. Ben Shneiderman. 2010. *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India.

33. Christian Stickel, Josef Fink, and Andreas Holzinger. 2007. Enhancing universal access–EEG based learnability assessment. *Universal Access in Human-Computer Interaction. Applications and Services* (2007), 813–822.

34. Alistair G. Sutcliffe and I Wang. 1991. Integrating human computer interaction with Jackson system development. *The computer journal* 34, 2 (1991), 132–142.

35. Jan Van Den Bos, Marinus J. Plasmeijer, and Pieter H. Hartel. 1983. Input–Output Tools: A Language Facility for Interactive and Real-Time Systems. *IEEE Transactions on Software Engineering* 3 (1983), 247–259.

36. Wil van der Aalst. 2016. *Data Science in Action*. Springer Berlin Heidelberg, 3–23.

37. Wil Van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. 2012. Replaying history on process models for conformance checking and performance analysis. *Wiley Int. Rev.: Data Mining and Know. Disc.* 2, 2 (2012), 182–192.

38. Anthony I. Wasserman. 1985. Extending state transition diagrams for the specification of human-computer interaction. *IEEE Trans. on Soft. Eng.* 8 (1985), 699–713.