An Approach to Identifying What Has Gone Wrong in a User Interaction

Andrea Marrella, Lauren S. Ferro, and Tiziana Catarci

DIAG, Sapienza University of Rome, Italy
{marrella,lsferro,catarci}@diag.uniroma1.it

Abstract. Nowadays, there is an increasing number of software applications offering task-based interactions through mobile devices or (directly) via the surrounding technological environment. Such interactions, which are difficult to assess with traditional user evaluation techniques due to their volatility, are usually recorded in dedicated *interaction logs*, which are then sent back to the software developers who must make sense of them. To date, *log studies* are mainly used to extract user behaviours from interaction logs for profiling purposes, or to compare such behaviors across different system variants. In this paper, we present a novel approach based on a declarative specification of *interaction models* that exploits logs for identifying exactly *what has gone wrong during a user interaction*, detecting which user actions may have caused usability issues and suggesting reparative actions for solving them.

1 Introduction

The modern revolution in Information Technology (IT) has allowed us to interact in advanced ways with a plethora of mobile devices and technological artefacts embedded in the surrounding environment. This has led to new shapes of user interfaces (UIs) and styles of interaction. Today's UIs range from simple mobile input devices with touchscreens and clickable symbols to complex artifacts without there being any visible surface presenting controls or displays of any kind. While the general feeling is that such increased interactivity is a positive feature, associated with being flexible and in control [18], the volatility of modern interactions has made more complex their in-depth analysis.

In the Human-Computer Interaction (HCI) field, *usability* is the key feature to capture the *quality of an interaction* with a UI in terms of measurable parameters such as time taken to (and learn how to) perform relevant tasks and number of errors made [3]. To measure the usability of a UI, the literature proposes several *user evaluation* techniques (the work [13] identified 95 techniques in 2003), which belong to three categories: *lab studies*, *field studies* and *log studies*.

In *lab studies*, participants are brought into a laboratory and asked to perform certain tasks of interest. Analysts can learn a lot about how participants interact with a UI, but the observed behavior happens in a controlled and artificial setting and may not be representative of what would be observed "in the wild" [20]. Alternatively, *field studies* collect data from participants conducting their own activities in their natural environments. Data collected in this way tends to be more authentic than in lab studies, but the presence of an evaluator observing what participants are doing may interfere with the natural flow of the interaction [16]. Both techniques are expensive in terms of the time that they require to collect the data. This can limit the number of user tests that can be performed.

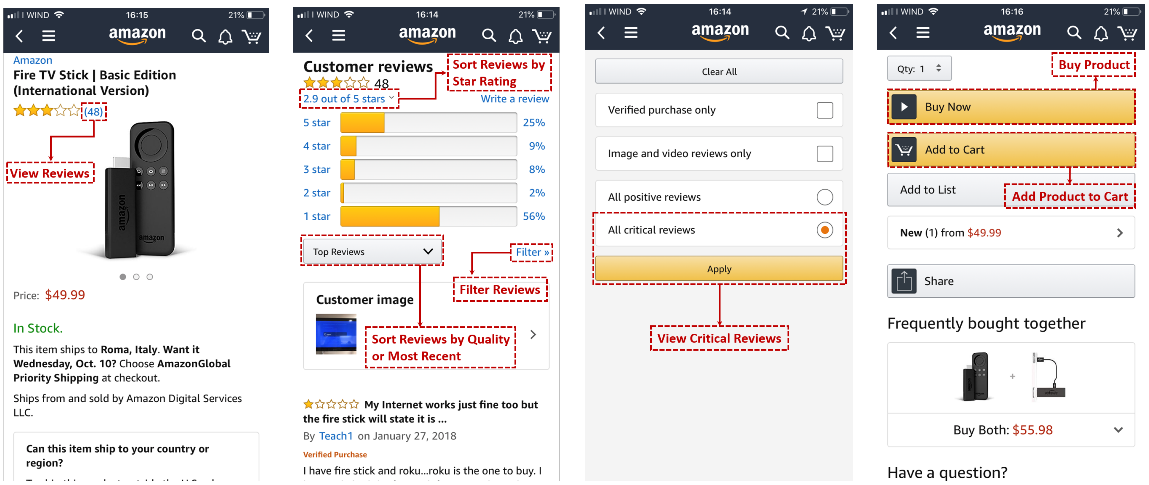
To mitigate the above limitations, it became common practice to capture the interactions with a UI during daily use and save them into log files for later analysis employing dedicated *log studies* [11]. *Interaction logs* include the user actions (from low-level keystrokes to content shared via social media) recorded "in situ" as people interact with UIs of software applications, uninfluenced by external observers. Interaction logs have the benefit of being easy to capture at scale (they can include data from tens to hundreds of millions of people), enabling to observe even small differences that exist between populations, including unusual behaviour that is hard to capture with the other studies. For this reason, today major software companies employ expert analysts to reveal user insights from interaction logs. Log studies can have an *observational* nature [21,34], when they are targeted to profile the end users of an application for marketing purposes, or an *experimental* nature [2,22,35], which is oriented to enable comparisons between two or more UIs (e.g., A/B testing).

Under the umbrella of log studies, this paper presents a novel approach that exploits interaction logs for a different yet little-explored challenge, namely the *automated identification of what has gone wrong during a user interaction.* Our approach is based on the concept of *alignment.* It verifies whether the user's "observed" behavior, which is recorded in a specific interaction log and reflects the basic user actions performed during the interaction with a UI while executing a relevant task, matches the "intended" behavior represented as a model of the interaction itself. A perfect alignment between the log and the model is not always possible, thus making *deviations* be highlighted. The approach can identify which user actions are responsible to cause such deviations, thus detecting potential usability issues with respect to the interaction model, and suggests reparative actions for solving them. Despite in the HCI literature many notations have been proposed for describing human-computer dialogs as interaction models [28], to realize our approach we opted for the DECLARE language [29], which enjoys formal semantics grounded in Linear Temporal Logic (LTL) [30] that has been proven to be adequate for designing interaction models [10].

The rest of the paper is organized as follows. Section 2 introduces a running example that will be used to explain our approach. In Section 3, we provide the relevant background necessary to understand the paper. Then, Section 4 presents an overview of our approach, while Section 5 concludes the paper with a critical discussion about its general applicability and tracing future work.

2 Running Example

As a running example, let us consider a situation where Amazon wants to check if its shopping mobile app for iPhone has the potential for improvements with



This item ships to **Roma, Italy**. **Want it Wednesday, Oct. 10?** Choose **AmazonGlobal Priority Shipping** at checkout.

Ships from and sold by Amazon Digital Services LLC.

Can this item ship to your country or region?

(a) Reading reviews.

My Internet works just fine too but the fire stick will state it is ...

By Teach1 on January 27, 2018

Verified Purchase

I have fire stick and roku...roku is the one to buy. I

(b) Filtering reviews.

(c) Critical reviews.

(d) Adding to cart.

Fig. 1. Some screenshots of the UI of Amazon shopping mobile app

regards to its UI design[1], with a specific focus on the feature that allows the users to *read the critical reviews* of a product before buying it. As shown in Fig. 1(a), to access the reviews of a product (in our case, a "Fire TV stick"), a user can select the link at the right of the review stars that shows how many reviews are associated to the product. Then, a new screen enables us to view reviews by stars rating, to sort them by their quality or by most recent, and to filter them, cf. the dedicated drop-down menu and links in Fig. 1(b), in particular the "Filter ≫" link. This third option takes users to a different screen, cf. Fig. 1(c), which allows to select only the critical reviews of the product, i.e., those reviews with 1, 2 or 3 stars. Finally, Fig. 1(d) shows two buttons that allow a user to add the product to the cart or to directly buy it, respectively.

Given the above scenario, we assume that the analysts of Amazon want to assess the usability of the UI of the mobile app with respect to the task: "*Read only the critical reviews associated to a Fire TV stick, and then buy it*". To this end, the common practice would be to employ a lab or a field study, which requires to involve several users that must be observed by external evaluators over an extended period of time during their interaction with the UI. The major obstacle is that the cost and time required to conduct lab and field testing against a stable release of a software application are often too high [14]. Consequently, in the "after-release" stage, companies tend to fix usability/learning issues only when such issues are reported by the end-users in form of complaints [11].

[1] The authors are not affiliated with Amazon in any way. The running example is just an exploration of some possible design issues of Amazon shopping mobile app.

In this paper, we exploit the above scenario to present an approach that can identify potential usability issues in a UI by directly employing the "knowledge" stored into interaction logs. This knowledge, presented in the form of execution traces, reflects the concrete user interactions happened with a UI.

3 Background

3.1 Interaction Logs

In HCI research, interaction logs arise from the user actions recorded when people interact with UIs of software applications. Such actions include all the steps (e.g., windows opened, system commands executed, check boxes clicked, text entered/edited, gestures, ect.) required to accomplish a relevant task (e.g., copy and paste a document) using the UI of a software application (e.g., a text editor). In the running example, we can recognize a universe of user actions of interest $Z = \{b, c, f, n, r, s, v\}$, such that: v = View Reviews, s = Sort Reviews by Quality or Most Recent, r = Sort Reviews by Stars Rating, f = Filter Reviews, n = View Critical Reviews, b = Buy Product, c = Add Product to Cart.

From a technical point of view, an interaction log is a multi-set of *execution traces*. Each trace consists of a sequence of *user actions* related to the *single execution* of a specific relevant task. Multiple executions of the same task may consist of the same sequence of actions executed and, hence, result in the same trace. This motivates the definition of an interaction log as a multi-set. If we consider our running example, the following $L_1 = [\langle v, f, n, b\rangle, \langle v, c\rangle, \langle v, r, b\rangle, \langle v, s, f, n, b\rangle, \langle v, f, n, b\rangle, \langle v, s, r, b\rangle, \langle n, b\rangle]$ is an example of interaction log consisting of 7 traces.

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. Interaction logs can be collected on a client machine or on remote servers. Client-side logging can be included in operating systems, applications such as browsers or e-readers, or special purpose logging software or hardware. Server-side logging is commonly used by service providers such as web search engines or e-commerce sites.

In this paper, we assume that the user's actions associated to a relevant task are already clustered in execution traces that refer to single enactments of the task itself. In a nutshell, our approach leverages on interaction logs containing only such kinds of execution traces. The identification and the extraction of relevant tasks from raw log data is out of the scope of this paper. Interested readers can refer to [25,31,8] for an insightful discussion of this topic.

3.2 Modeling Human-Computer Dialogs as Declare models

The HCI literature is rich of notations for expressing human-computer dialogs that allow to see at a glance the structure of a dialog [28,10]. Existing notations can be categorized in two main classes: *diagrammatic* and *textual*. Diagrammatic notations include (among the others) various forms of state transition networks (STNs) [37], Petri nets [33], Harel state charts [15], flow charts [10], JSD diagrams [32] and ConcurTaskTrees (CTT) [24]. Textual notations include regular expressions [36], LTL [30], Communicating Sequential Processes (CSPs) [9], GOMS [19], modal action logic [4], BNF and production rules [12].

While there are major differences in expressive power between different notations, an increased expressive power is not always desirable as it may suggest a harder to understand description, i.e., the dialog of a UI can become unmanageable. To guarantee a good trade-off between expressive power and understandability of the models, we decided to use DECLARE [29] for their specification. DECLARE is a declarative modeling language that allows us to describe a set of (temporally extended) constraints that must be satisfied by a user interaction. In DECLARE, the orderings of user actions are implicitly specified by constraints and any interaction that does not violate them is considered legal. The semantics of DECLARE is grounded in LTL. Compared with procedural approaches (e.g., CTT or Petri Nets), where all allowed behaviours must be explicitly represented in interaction models, DECLARE constraints are more suitable to describe interactions including many possible behaviours. Being based on LTL, where *all what is not explicitly specified is allowed*, few DECLARE constraints can specify many possible behaviors at once. Notably, the adoption of DECLARE constraints makes the definition of interaction models independent of the formalization in LTL.

In the following, we summarize some of the most relevant DECLARE constraints (the reader can refer to [29] for a full description of the language). Constraints *existence(A)* and *absence(A)* require that A occurs at least once and never occurs in every execution trace, respectively. The *response(A,B)* constraint specifies that when A occurs, then B should eventually occur after A. The *precedence(A,B)* constraint indicates that B can occur only if A has occurred before. The *succession(A,B)* constraint states that both response and precedence relations hold between A and B. The *chain succession(A,B)* constraint states that A and B must occur one immediately after the other. DECLARE also includes some negative constraints to explicitly forbid the execution of actions. According to the *not succession(A,B)* constraint, any occurrence of A can not be eventually followed by B. Finally, the *not chain succession(A,B)* constraint states that A and B can not occur one immediately after the other.

If we consider our running example, we can specify the interaction model that describes the expected behaviour underlying the relevant task: "*Read only the critical reviews associated to a Fire TV stick, and then buy it*" as the set consisting of the following DECLARE constraints:

- $absence(s)$ means that action $s =$ Sort Reviews by Quality or Most Recent can not ever be performed.
- $existence(b)$ means that action $b =$ Buy Product must be executed at a certain point of the interaction.
- $precedence(n, b)$ forces $n =$ View Critical Reviews to precede $b =$ Buy Product.

Formally speaking, a DECLARE model $D = (Z, \pi_D)$ consists of a universe of user actions Z involved in an interaction log and a set of DECLARE constraints π_D defined over such actions. With the above interaction model, an HCI designer may want to express that to correctly execute the task under observation a product must be eventually bought (i.e., *existence*(b)) after the user has read the critical reviews associated to it (i.e., *precedence*(n, b)). Sorting reviews by quality or by most recent is considered as a user mistake (i.e., *absence*(s)). The fact that some of the actions identified in Section 3.1 are not involved in any DECLARE constraint means that they are not considered as potential sources of usability issues. For example, no matter if a user sorts reviews of a product by stars rating or adds the product to the cart before buying it, since such actions do not violate any constraint in the DECLARE model. To be more specific, given an execution trace belonging to an interaction log, the problem we want to address in this paper is to identify *all potential violations* of DECLARE constraints representing an interaction model in the trace. Such violations, which reflect potential usability issues, can be detected and repaired by our approach.

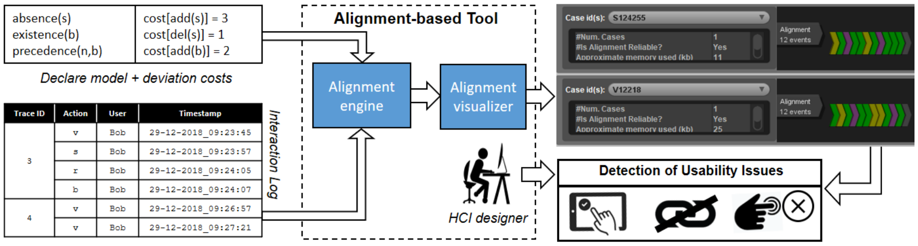
4 Approach

As shown in Fig. 2, our approach relies on 4 steps to be performed in sequence.

First, it is necessary to collect an interaction log L containing execution traces that describe concrete interactions performed by end users to achieve the objectives of a relevant task of interest. As already pointed out in Section 3.1, in this paper we do not focus on how such interaction logs are recorded. We assume that they can be generated through massive web-based user tests, which large companies often periodically conduct with thousands of end users.

Secondly, it is required to formalize the potential dialog between the user and the UI by employing a collection of DECLARE constraints (i.e., a DECLARE model) as interaction model. Using DECLARE, the model consists just of those constraints *that must not be violated* by a user interaction during the execution of a relevant task. Interestingly, an interaction model may allow different strategies to perform a relevant task. For example, if we consider the DECLARE model associated to the relevant task of the running example (see Section 3.2), the number of ways to properly achieve the objectives of the task are potentially unbounded. For instance, the execution traces $\tau_1 = \langle v, f, n, b\rangle$ and $\tau_2 = \langle n, b\rangle$ represent (both) good ways to execute the relevant task. This basically means that a same task can be completed through different traces of user actions in the UI with equivalent results, as may happen in real UIs.

Thirdly, given a DECLARE model of the relevant task of interest and an interaction log associated to it, we can construct the *alignment* between any of the traces extracted from the log and the model. The alignment activity consists of *replaying* any user action included in a trace over the interaction model. Sometimes, actions as recorded in the log cannot be matched to any of the actions allowed by the model. A *deviation* can manifest itself in *skipping* actions that have been executed in the log but are not allowed by the model (e.g., a user sorting the reviews by their quality or by most recent), or in *inserting* actions, namely, some actions that should have been executed (i.e., prescribed by the model) but are not observed in the log (e.g., reading the critical reviews of a product is needed before to buy it). If an alignment between a log trace and



Trace ID	Action	User	Timestamp
3	v	Bob	29-12-2018_09:23:45
	s	Bob	29-12-2018_09:23:57
	r	Bob	29-12-2018_09:24:05
	b	Bob	29-12-2018_09:24:07
4	v	Bob	29-12-2018_09:26:57
	v	Bob	29-12-2018_09:27:21

Case id(s): S124255
#Num. Cases 1
#Is Alignment Reliable? Yes
Approximate memory used (kb) 11

Case id(s): V12218
#Num. Cases 1
#Is Alignment Reliable? Yes
Approximate memory used (kb) 25

Fig. 2. An overview of the proposed approach

model contains at least a deviation, it means that the trace refers to a user interaction that is not compliant with the allowed behavior represented by the model. As a matter of fact, the alignment moves (i.e., skipping or inserting actions) indicate where the interaction is not conforming with the model by pinpointing the deviations that have caused this nonconformity. Pinpointing the actual reasons of nonconformity is crucial for identifying potential usability issues.

For example, given the trace $\tau_3 = \langle v, s, r, b \rangle$ taken by our running example and the interaction model described by the DECLARE constraints in Section 3.2, an alignment activity will identify that: *(i)* action s = Sort Reviews by Quality or Most Recent has been executed even if forbidden by the model, and *(ii)* action n = View Critical Reviews is required by the model (even if it does not appear in the trace), and must be executed before b = Buy Product. In a real interaction log, this same trace can appear hundreds of time. The alignment of trace τ_3 with the model will instruct to *skip* action s and *insert* action n before b, i.e., the aligned trace is $\hat{\tau}_3 = \langle v, del(s), r, add(n), b \rangle$. Recovery instructions are labeled with *add* and *del* to capture those wrong/missing actions that must be removed/inserted from/into the aligned trace to make it compliant with the model.

Our approach takes inspiration from conformance checking techniques in the Business Process Management field [5], where concrete tools exist [1,7,6] that allow us to *(i)* compute alignments between logs and declarative models, and to *(ii)* detect and visually locate deviations (in form of recovery instructions) on top of a trace, e.g., see the right part of Fig. 2. The analysis of the recovery instructions enables the last step of our approach, which is targeted to support an HCI designer to detect potential usability issues in the UI. For example, considering $\hat{\tau}_3 = \langle v, del(s), r, add(n), b \rangle$, it is possible to infer that the user was confused about how to filter critical reviews, probably due to the lack of visual cues in the UI of the mobile app, or could not find the feature. Based on this, a possible solution could be to update the UI by reducing the friction for sorting and filtering reviews. For example, both the “Filter By” and “Sort By” features can be organized as drop down menus and clearly labeled in the UI.

We finally notice that the alignment considers also the *severity* of a deviation, by assigning a non-zero cost for repairing a wrong/missing action. For instance, pushing the “Help” button to access to the help features of the UI should not be punished too hard: it might have been unnecessary but it does not imply any concrete violation of the interaction model. Using a dedicated severity function, 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 log can be associated with valid execution paths specified by the model. To be more specific, the fitness value can vary from 0 to 1 (*perfect fitness*), and quantifies the amount of deviations between a trace and a model. This value is particularly useful when HCI designers want to quickly assess if the amount of deviations of a UI decreases over time, update after update.

5 Discussion and Concluding Remarks

Interaction logs and log studies represent an important tool to collect insightful data about how users interact with a UI, capturing actual user behavior and not recalled behaviors or subjective impressions. In this paper, we have presented an alignment-based approach that exploits interaction logs for identifying what has gone wrong during a user interaction with a UI to detect potential usability issues. To tackle this challenge, our approach leverages explicit interaction models defined as DECLARE constraints, which is a novelty in the HCI literature.

We found in the literature three similar approaches that analyze interaction logs with support of explicit interaction models to identify mistakes during a user interaction [26,27,23]. The main difference of our approach with respect to the previous ones relies in the definition of the models. In fact, previous works employ CTT (cf. [26,27]) and Petri Nets (cf. [23]) for the definition of the models, forcing the designer to define *all potential behaviours* of a user interaction, which is unrealistic when the UI allows several paths to complete a task. Conversely, in our approach, the use of DECLARE models enables an HCI designer to define *just the behaviour of interest*, making easier the definition of the models.

One major objective of our approach is to show that the use of automated mechanisms to interpret logs may offer a compromise solution and an interesting complement to pre-release lab and field testing, reducing at the same time the efforts required to perform user evaluations of UIs of already-released applications. In addition, understanding exactly what happened during such interactions is crucial for developing a culture of *transparency* and *explainability* of UIs [17].

A weakness of our approach is that the logs must be already organized in execution traces that refer to the specific task under evaluation. This activity, which is out of the scope of this paper, is not trivial, and requires the use of large web-based user tests that enable to generate and organize the content of such interaction logs. Consequently, a first future work is to devise a methodology to understand how to collect proper interaction logs to use as input for our approach. Then, to validate our approach, we are working on designing a longitudinal study to be performed with real users. Finally, we aim at identifying some "thresholds" in the amount of deviations found that may help to *(i)* automatically coupling deviations to usability issues and *(ii)* classifying them according to their severity (e.g., "critical", "important" or not "important" issues).

References

1. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking Using Cost-Based Fitness Analysis. In: EDOC'11 (2011)
2. Balagtas-Fernandez, F., Hussmann, H.: A Methodology and Framework to Simplify Usability Analysis of Mobile Applications. In: ASE'09 (2009)
3. Benyon, D.: Designing Interactive Systems: A comprehensive guide to HCI, UX and interaction design, 3/E (2014)
4. Campos, J.C., Sousa, M., Alves, M.C.B., Harrison, M.D.: Formal verification of a space system's user interface with the ivy workbench. IEEE SMC **46**(2) (2016)
5. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer (2018)
6. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the Disruptive Effectiveness of Automated Planning for LTLf-Based Trace Alignment. In: AAAI'17 (2017)
7. De Giacomo, G., Maggi, F.M., Marrella, A., Sardiña, S.: Computing Trace Alignment against Declarative Process Models through Planning. In: ICAPS'16 (2016)
8. Dev, H., Liu, Z.: Identifying frequent user tasks from application logs. In: IUI'17. ACM (2017)
9. Dignum, M.: A model for organizational interaction: based on agents, founded in logic. SIKS (2004)
10. Dix, A., Finlay, J., Abowd, G., Beale, R.: Human-Computer Interaction. Pearson (2004)
11. Dumais, S., Jeffries, R., Russell, D.M., Tang, D., Teevan, J.: Understanding user behavior through log data and analysis. In: Ways of Knowing in HCI (2014)
12. Feary, M.S.: A toolset for supporting iterative human automation: Interaction in design. NASA Ames Research Center (2010)
13. Ferre, X., Juristo, N., Moreno, A.M.: Improving software engineering practice with HCI aspects. In: SERA'03. Springer (2003)
14. Ferre, X., Villalba, E., Julio, H., Zhu, H.: Extending mobile app analytics for usability test logging. In: INTERACT'17. Springer (2017)
15. Harel, D.: Statecharts: A visual formalism for complex systems. Science of computer programming **8**(3), 231–274 (1987)
16. Hartson, H.R., Andre, T.S., Williges, R.C.: Criteria for evaluating usability evaluation methods. Int. Journal of Human-Computer Interaction **13**(4), 373–410 (2001)
17. Holzinger, A., Biemann, C., Pattichis, C.S., Kell, D.B.: What do we need to build explainable ai systems for the medical domain? arXiv:1712.09923 (2017)
18. Janlert, L.E., Stolterman, E.: Things That Keep Us Busy: The Elements of Interaction. MIT Press (2017)
19. John, B.E., Kieras, D.E.: The goms family of user interface analysis techniques: Comparison and contrast. ACM TOCHI **3**(4) (1996)
20. Kjeldskov, J., Skov, M.B.: Was it worth the hassle?: Ten years of mobile HCI research discussions on lab and field evaluations. In: MobileHCI '14. ACM (2014)
21. Lau, T., Horvitz, E.: Patterns of search: analyzing and modeling web query refinement. In: UM99 User Modeling, pp. 119–128. Springer (1999)
22. Lettner, F., Holzmann, C.: Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications. In: 10th Int. Conf. on Advances in Mobile Computing & Multimedia. pp. 118–127. ACM (2012)
23. Marrella, A., Catarci, T.: Measuring the Learnability of Interactive Systems Using a Petri Net Based Approach. In: 2018 Designing Interactive Systems Conf.

24. Mori, G., Paternò, F., Santoro, C.: CTTE: support for developing and analyzing task models for interactive system design. IEEE Trans. Sof. Eng. **28**(8) (2002)
25. Oliver, N., Smith, G., Thakkar, C., Surendran, A.C.: Swish: semantic analysis of window titles and switching history. In: IUI'06. ACM (2006)
26. Paganelli, L., Paternò, F.: Tools for remote usability evaluation of web applications through browser logs and task models. Behavior Research Methods, Instruments, & Computers **35**(3), 369–378 (2003)
27. Parvin, P.: Real-time anomaly detection in elderly behavior. In: ACM SIGCHI Symposium on Engineering Interactive Computing Systems. EICS '18, ACM (2018)
28. Paterno, F.: Model-Based Design and Evaluation of Interactive Applications. Springer-Verlag, 1st edn. (1999)
29. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: EDOC. pp. 287–300 (2007)
30. Pnueli, A.: The temporal logic of programs. In: Found. of Computer Science (1977)
31. Shen, J., Irvine, J., Bao, X., Goodman, M., Kolibaba, S., Tran, A., Carl, F., Kirschner, B., Stumpf, S., Dietterich, T.G.: Detecting and correcting user activity switches: algorithms and interfaces. In: IUI'09. ACM (2009)
32. Sutcliffe, A.G., Wang, I.: Integrating human computer interaction with jackson system development. The computer journal **34**(2), 132–142 (1991)
33. Sy, O., Bastide, R., Palanque, P., Le, D., Navarre, D.: PetShop: a CASE tool for the Petri Net based specification and prototyping of CORBA systems. In: Petri nets. vol. 2000, p. 78 (2000)
34. Tyler, S.K., Teevan, J.: Large scale query log analysis of re-finding. In: Third ACM Int. Conf. on Web search and data mining. pp. 191–200. ACM (2010)
35. Valle, T., Prata, W., et al.: Automated usability tests for mobile devices through live emotions logging. In: 17th Int. Conf. on Human-Computer Interaction with Mobile Devices and Services Adjunct. pp. 636–643. ACM (2015)
36. Van Den Bos, J., Plasmeijer, M.J., Hartel, P.H.: Input–Output Tools: A Language Facility for Interactive and Real-Time Systems. IEEE Trans. Sof. Eng. (3) (1983)
37. Wasserman, A.I.: Extending state transition diagrams for the specification of human-computer interaction. IEEE Trans. on Soft. Eng. (8), 699–713 (1985)