Design Space Exploration of Adaptive User Interfaces

Sarah Bouzit1,2, Gaëlle Calvary2, Joëlle Coutaz2, Denis Chêne1, Eric Petit1, Jean Vanderdonckt3

1Orange Labs, 28 chemin du Vieux Chêne, F-38240 Meylan (France), {sarah.bouzit, denis.chene, eric.petit}@orange.com
2Université Grenoble Alpes, Laboratoire d’Informatique de Grenoble (LIG), F-38000 Grenoble (France),
3Université catholique de Louvain, Louvain School of Management, B-1348 Louvain-la-Neuve (Belgium),
{sarah.bouzit, gaelle.calvary, joelle.coutaz}@imag.fr, jean.vanderdonckt@uclouvain.be

Abstract— Extensive investigation of user interface adaptation has produced so many different adaptation techniques that comparing them becomes challenging, especially for adaptive user interfaces, thus preventing any systematic design space exploration. This paper presents a design space for engineering adaptive user interfaces throughout the user interface development life cycle in order to describe any adaptivity technique, to compare two or more techniques with respect to criteria, and to generate new, perhaps unprecedented, techniques. Grounded in the theory of psychological perception, this design space structures the adaptation life cycle into two regulation loops between the user and the system, thus enabling the definition of properties for assessing the quality of these loops. A method for systematic exploration of this design space of is then sketched, formalized, and instantiated on two advanced adaptive user interfaces: adaptive user interfaces based on machine learning and adaptive layouts. This exploration provides new insights for considering design options for designing adaptive user interfaces in general and adaptive menus in particular.

Keywords— Adaptation of user interfaces, adaptive user interfaces; design space exploration; quality properties; intelligent user interfaces.

1. INTRODUCTION

User Interface adaptation primarily consists in modifying parts or whole of a particular User Interface (UI) of any interactive system to satisfy specific needs required by one or many end users. Adaptation falls into two categories depending on who is controlling the adaptation [21,25]: adaptability refers to as the end user’s ability to adapt the UI [41], adaptivity refers to as the system’s ability to adapt the UI [13]. Mixed-initiative adaptation [7,8] occurs when both the end user and the system collaborate towards the adaptation. UI Adaptation has been extensively investigated [21], researched [35], developed [30], and tested [39] with the ultimate goal of optimizing the overall user experience with respect to her needs namely by: increasing user performance [11] and/or preference [1], reducing task completion time and error rate [5,16], improving the subjective user satisfaction and increasing the learning curve [27].

The challenge is to suggest the right adaptation at the right time at the right place for making it valuable to the user [25]. Otherwise, adaptation will be prone to several limitations that could become a major impediment to the expected benefits, if not thwart them [27]: risk of misfit (the end user’s needs are incorrectly captured or interpreted), user cognitive disruption (the end user is disrupted by the adaptation), lack of prediction (the end user does not know when and how the UI will be adapted by the system), lack of explanation (the system rarely provides the end user with some explanation on why this adaptation process took place), lack of user involvement (the end user is rarely given the opportunity to be involved in the adaptation process), and risk for privacy (the system maintains information related to the user’s needs). Whereas several adaptation techniques have emerged for desktop UIs, e.g., [2,15,18,37], they are not well suited to UI adaption for smartphones [5], which are more constrained by their computational power and available interaction techniques. Navigation through pages and lists like phone settings or address books can be cumbersome and time consuming especially when they get long. Selecting a target implies navigation and visual search times depending on the screen density. Hence, extrapolating adaptation techniques for new devices and computing platforms does not fit, which calls for inventing new adaptation designs, but which ones?

“The field of adaptive systems is infamous for its lack of standards, or even commonly accepted approaches” [34]. Surveys on UI adaptation have been published [6,13,19,30,35], attempting to summarize adaptation concepts, methods, and tools but they are often technologically driven, limited in scope, or surpassed by recent technological progress, which make them inappropriate for expressing the most recent adaptation techniques, especially for device adaptation and adaptivity, and for systematically exploring alternatives through a design space.

In order to address this challenge, the remainder of this paper is structured as follows: the next section reports on the theoretical background related to UI adaptation with a focus on surveys and design spaces. Then, a design space for UI adaptation is motivated, introduced, and defined based on the theory of perception coming from cognitive psychology. While this design space is valid for adaptable, adaptive, and mixed-initiative UIs, a special focus will be devoted to adaptivity since it has received less attention and is more challenging that adaptable UIs [27]. In addition, we would like to overcome the expressiveness of existing design spaces only tailored to adaptivity [9,19]. A method for systematic exploration of this design space is presented by defining a series of quality properties based on adaptation cycles and instantiated on two advanced adaptive UIs developed for this purpose and that are therefore considered as reference examples: an adaptive UI with widget selection based on machine learning and an adaptive UI with adaptive layout based on previous action sequences. The quality properties introduced based on the design space are also related to existing definitions of similar quality properties, which are scattered throughout the literature and suffer sometimes from ontological confusion and inconsistent definitions. By doing so, this paper is also aimed at integrating these quality properties into a coherent expression instead of a myriad of independent properties. This design space is expected to serve three virtues: descriptive, comparative, and generative. Finally, a conclusion delivers the main points of this research and presents some future avenues based on the design space.
II. BACKGROUND ON USER INTERFACE ADAPTATION

Pioneering work started with Browne et al. [6], who used Command Language Grammar (CLG) for specifying an adaptive UI, to conclude that the major strength was a way to support the Principle of Separation of Concerns in UI adaptation, starting with a conceptual model and enforcing it throughout the UI development life cycle. Although this enforcing was made explicit, it was not obvious though to easily propagate all specifications aspects contained in the CLG specifications into the final code. They indicated that CLG has very limited facilities for expressing UI presentation and behaviour, thus raising the need for improving the expressiveness in this respect.

Dieterich et al. taxonomy [13] has long been considered as a seminal reference for classifying different types of adaptation configurations and techniques. More than 200 papers dealing with various adaptation forms were sorted and summarized into four stages: initiative (which entities involved in the interaction expresses its intention to perform an adaptation), proposal (if a need for adaptation arises, it requires making proposals of adaptations that could be applied successfully given the current situation), decision (as several proposal could come up from the previous stage, which adaptation proposal best fit the end user’s needs detected should be decided, and execution (the adaptation technique previously chosen is finally enacted). Given these four stages, the authors classified every system with adaptation capabilities according to the actors involved at each stage. Fig. 1 presents an adaptation configuration in which the system recognizes the need for adaption, proposes some design alternatives, the end user is then picking the design alternative that she prefers to be finally executed by the system.

McKinley’s taxonomy [30] addresses compositional adaptation, which results in the exchange of algorithmic or structural parts of the system with ones that improve a program’s fit to its current context of use. This kind of adaptation is based on the separation of concerns between the functional behaviour and the cross-cutting concerns, the computational reflection that provides a vehicle to query the different aspects of a system, the component-based design practices that enable the development of the different parts of a system separately and the middleware that usually provides the compositional capabilities. The taxonomy is structured along three dimensions:

1. How to adapt: the process can be carried out by different entities: a human (e.g., a developer, a system administrator), a component loader, a run-time system or a meta-object.
2. Where to adapt: it describes where in the system the adaptation code is inserted. The most common approach is to place the code in the middleware, although extensible operating systems have also been used.
3. When to adapt: the adaptation time (Fig. 2) is static, respectively dynamic, when it takes place at design, prototyping, development, compile, link, or load time, respectively at run time. Moreover, it is said to be hardwired (when the UI adaptation is embedded in the code of the interactive application, in the UI code), customizable (when the UI adaptation enables some pre-computed freedom), configurable (when the UI adaptation technique could be configured before executing it), tunable (when the UI adaptation technique could fine-tune the UI at run time without modifying its code), or mutable (when the UI adaptation technique subsumes the run time code modification of the interactive application, such as in generative programming [37]).

Although this taxonomy is applicable to an entire interactive application, the three dimensions (i.e., how, where, when) are particularly constructive for a design space: ‘how to adapt’ grossly corresponds to the proposal stage in Dieterich’s taxonomy, ‘where to adapt’ is an implicit part of the execution stage, ‘when to adapt’ was uncovered. Therefore, McKinley’s taxonomy complement the Dieterich’s taxonomy, but both together do not detail the adaptation life cycle, a major reason why the ISATINE framework [29] has been introduced: to decompose the adaptation life cycle into seven stages according to Norman’s mental model of action [33], classified according to the part of the adaptation process where they occur. Two parts are identified (Fig. 3): the gulf of adaptation execution, located on the left part, covers all the stages in the adaptation process required to finally execute the adaptation and the gulf of adaptation evaluation, located on the right part, including those stages assessing the output issued from the execution gulf. As in Norman’s mental model [33], the process starts by stating adaptation goals which may trigger an initiative for adaptation to be specified and applied. Once the adaptation has been com-
pleted by the system, a transition may take place until interpretation of the adaptation will be reached, and evaluated with respect to the initial goals. Depending on this feedback is interpreted positively (i.e., the adaptation goal has been fulfilled) or negatively (i.e., the adaptation goals has not been achieved), another adaptation loop is created. Any stage can be carried out by any combination of various agents: end user, system, stakeholder, external third party (e.g., a moderator, a broker).

While more expressivity is gained, the ISATINE framework still misses important recent aspects: it only details the stages with respect to the end user’s viewpoint and does not provide any insight from the system viewpoint, it does not separate the part related to the main interaction from the adaptation itself, it does not express how adaptivity is achieved, and it does not allow expressing adaptation quality properties on it.

III. DESIGN SPACE FOR ADAPTIVE USER INTERFACES

A. Requirements for the Design Space

Any design space for adaptive UIs, as any other design space or model, should serve three virtues [3]:

1. A descriptive virtue: any adaptive UI should be described completely, consistently, and unequivocally.

2. A comparative virtue: any set of adaptive UIs should be made comparable according to the same criteria defined in the design space. This supports a sound comparative analysis of techniques and a rigorous benchmarking of them. When two adaptive UIs are compared on the design space, their coverage could be highlighted, thus enabling an identification of their respective strengths and weaknesses, which is referred to as commensurability.

3. An exploratory virtue: all dimensions steps of the design space could be systematically explored in order to identify where existing techniques are located, where they are strong or weak, where new opportunities emerge, and where are underexplored portions remain to be investigated, if not discovered yet. In particular, uncovered steps and dimensions of adaptive UIs could be analysed from both a theoretical and technological support viewpoints.

Beyond these three virtues, any design space should materialize the phenomenon of co-evolution between the user and the system when adaptivity occurs. As well as the user adapts herself while interacting with the system, the system also adapts itself for anticipating user actions and facilitating user interaction. Therefore, we need to identify two loops: one for the main UI interaction and one for the adaptation of this UI.

Cognitive psychology [23,33], as well as human decision theory, argues that three activities, i.e., perception, decision, and action are essential activities to make sense of and interact with our context of use [10]. Any interactive task thus involves three activities: perception when the end user has to perceive the basic properties of the context of use, incling the UI, decision when the end user has to make a choice between competing options based on her own interpretation, and action when a particular action should take place has a decision has been made. The end user follows a series of Perception-Decision-Action (PDA) cycles (Fig. 4a) until the task is completed. While some researchers claim that these three types of actions are intertwined [8,14] and could be linked by bidirectional arrows indicating the non-consecutive interaction [21], we prefer to keep the conclusion of the initial PDA cycle because it is equally applicable to the UI: the system has to perceive what the user is actually doing (through sensors, user-generated events), to decide what next actions will be undertaken (e.g., through a dialog controller), and to execute required actions. Hence, the user-system interaction could be represented as a series of 2 PDA cycles (Fig. 4b).

On top of the two PDA cycles for interaction, another sequence of two cycles could be defined for the UI adaptation that are made up of three activities: learning, prediction, and adaptation, defined as a LPA cycle. On the user side, the end user after carrying out an action, gains experience from the previous PDA cycle and learns, which facilitates prediction of what to do for adapting herself to the system. On the system side, the system also learns from what the end user did, creates and maintains this knowledge for predicting what the end user’s future actions could be offered, and for applying any adaptation technique accordingly. As well as the user adapts herself while interacting, the system also adapts itself for anticipating user actions and facilitating user interaction. Adaptation cycles cease when either the user or the system decides to suspend the adaptation, otherwise there is a risk of infinite adaptation which does not converge towards any stable state.

B. Design Space Definition and Exploration

The design space for UI adaptation resulting from these requirements is illustrated in Fig. 5: the end user side, graphically depicted in blue, is made up of two cycles (one PDA cycle for the interaction and one LPA cycle for human adaptation) as well as the system, graphically represented in green (one PDA cycle for controlling the system according to the business logics and one LPA cycle for system adaptation). The resulting PDA-LPA design space for UI adaptation therefore makes explicit in a concise manner the activities from both sides (user and system) while differentiating the main interaction process from the adaptation process. When no system adaptation exists, the LPA-system sequence disappears. When no user adaptation is achieved, the LPA-user sequence disappears. An adaptable UI could be represented equally without the LPA-system sequence since the end user is adapting the system. An adaptive UI should go through these 4 sequences.
C. Quality Properties

Any adaptation could be described in the terms of the PDA-LPA design space as follows: which technique is promoted in the interactive system for supporting perception, decision, action, learning, prediction, and adaptation when it occurs. This should respect the aforementioned descriptive virtue. In order to respect the comparative virtue, quality properties could be defined on top of the design space, thus revisiting and enriching existing quality properties for interactive systems [22] and simultaneously introducing new abilities [42] of the adaptation. Quality properties in our design space fall into two categories: system (software) quality properties when the property could be defined as an ability ensured by the system in a user-independent way, or user quality properties when the property could be defined as an ability offered to the user, which is therefore user-specific. A quality property is said to be shared when both the end user and the system collaborate to ensure the property. Fig. 6 depicts a first series of global properties (i.e., between user and system considered as a whole):

- Observability: refers to the system’s ability to make perceivable its state in a relevant way for the end user. This subsumes the interaction (system observability) and/or the adaptation (adaptation observability). Adaptation observability is key for the end user to perceive what the system is doing to support adaptivity: in Learning (to perceive what the system is currently learning or has learnt from the user, such as user events, actions), in Prediction (to perceive that the system is able to predict an adaptation), and in Adaptation itself (to render that an adaptation is taking place, such as with transition between a UI before adaptation and after [12]). Observability is a generalization of immediate visibility since several techniques could ensure observability [22]: traceability (step-by-step perceivability) [31] or browsability (visibility by navigation on-demand): what is not immediately perceivable could be made browsable on demand.

- Intelligibility: refers to the system’s ability to communicate to the end user how the interaction and/or the adaptation processes are conducted in a meaningful and representative way [4]. As intelligibility subsumes observability [40] (before making the adaptation intelligible, it should be made observable), it could be similarly decomposed into interaction intelligibility and adaptation intelligibility. Intelligibility is key for the end user to understand the system and the adaptation engine, while they are running, like explaining the reasoning of an expert system is key in artificial intelligence. For a context-aware system, intelligibility refers to the system’s ability to present itself in a convenient manner, the way the context is perceived and its behaviour depending on significant changes over time [23]. Intelligibility could be ensured by different ways [4]: explainability (the adaptation is explained), continuity (the adaptation process is continuously rendered) [20], honesty [22], or transparency (which each adaptation is rendered univocally to the end user to avoid the aforementioned limitations of adaptivity) [9]. The end user can understand all the better that she is able to observe the system thanks to observability. Honesty refers to the system’s ability to achieve two aims: to make the real system state observable to the end user (which is challenging due to latency and lag) and to make this state accurately [20] interpreted by the end user.

- Controllability: refers to the user’s ability to control the system interaction (interaction controllability), the system adaptation (adaptation controllability), or both depending on which cycles are concerned: the system PDA cycle, the system LPA cycle, or both. Controllability could cover any interactive UI aspect in principle. When the end user has no control, the interaction and/or the adaptation cycles are entirely initiated and controlled by the system through self-controlling, self-regulation, or self-adaptation [9,34].

Browsability refers to the system’s ability for the user to explore the system state by the way of articulatory tasks (i.e., tasks that do not modify the state of the functional core, such as scrolling, zooming in, zooming out, navigation). Various forms of observability are graphically depicted in Fig. 7: learning observability from the learning activity to the end user, prediction observability from the prediction activity to the end user, and adaptation observability for the final adaptation executed. Each activity could be subject to observability independently of the others: it is not because the adaptation is observable that its learning and its prediction should be also observable. On the opposite, these three activities are rarely subject to observability, apart from showing the results of the final adaptation to the user.
Adaptation controllability is essential to enable the end user to be actively involved in any adaptation activity: in Learning (to specify to the system what is allowed to capture, interpret, and learn from the user for ensuring privacy), in Prediction (to control the parameters used for predicting the adaptation, e.g., via machine learning), and in Adaptation (to assess adaptation proposals, to accept a relevant adaptation—true positives— or reject irrelevant adaptations resulting from wrong predictions—false positives—).

- **Predictability**: refers to the user’s ability to predict future system actions for supporting the interaction, the adaptation, or both based on past corresponding actions [20]. Regarding the prediction predictability (see Fig. 8 for various forms of this property), the user should to some extent predict the behaviour of the adaptivity algorithm based on past adaptivity actions, which may subsume controllability: the end user can predict all the better that she is under control of the system thanks to controllability.

Fig. 8 graphically depicts a second series of local (from one step of the user/system directly to the other entity) properties:

- **Awareness**: refers to the user’s ability to perceive (hence, the perception activity is concerned) how the interaction, the adaptation, or both are occurring in the system. Awareness could be supported in several ways, such as context-awareness [21] (what are the contextual conditions that are estimated significant enough to trigger a change of context), action awareness (what are the actions undertaken by the system), decision awareness (how the system decides which adaptation), and perception awareness (how the system perceive the context of use). Awareness is positively influenced by corresponding system properties, such as observability and honesty: the more the system is observable and honest in what is observed, the more the user is capable of being aware of the system state, information, and actions.

- **Decidability**: refers to the user’s ability to decide (hence, the decision activity is concerned) what to do (in the interaction cycle) and/or how to adapt herself (in the adaptation cycle). In case of a system with internal control, there is no possibility for the user to decide anything; on the opposite side of the continuum, the user may decide of everything in case of a full external locus of control; in the middle, mixed-initiative [8] prompts the user with several options on what actions to undertake next, on which adaptation could take place so that the user may deciding while knowing the potentially positive and negative consequences of it.

- **Triggerability**: refers to the user’s ability to trigger (hence, the action activity is concerned) the actions she wants (in the interaction cycle) and/or the options needed for an appropriate adaptation. In adaptability for instance, the user is given the opportunity to adapt some UI features (e.g., icons, menu shortcuts, toolbar contents and position) at any time. For instance, some adaptation may be allowed or forbidden depending on the context of use or because the consequences will be beneficial or fatal for the end user. Deactivation of actions and adaptations is a typical example for revealing or hiding (un)triggerable actions.

Fig. 10 graphically depicts a third series of local (from one entity’s activity entity to another one in particular) properties:

- **Capacity**: refers to the system’s ability to execute (hence, the action activity is concerned) either domain actions (belonging to the business domain of activity) or adaptation actions or both [42]. The capacity for adaptation reveals the power of the adaptation techniques offered to the end user in terms of applicability. The end user may want to trigger some adaptation (in the triggerability), but the system has no capacity for carrying out the required adaptation. This may represent a mismatch between the adaptation goals and the system capacity to achieve them.

- **Autonomy**: refers to the system’s ability to decide (hence, the decision activity is concerned) either domain actions or adaptation actions or both. Autonomy may be governed by the locus of control: if adaptation actions are made available, the system may have the permission to decide which one to apply with or without the consent of the end user. When the system is given the full autonomy, the locus of control is completely internal. When the system has no autonomy, the locus of control is completely external. Agent technology is recognized for having autonomous agents that carry out tasks for the end user almost automatically, without the direct intervention of end users or others, sometimes even without any observability and controllability.
- **Perceptibility**: refers to the system’s ability to perceive (hence, the perception activity is concerned) domains and/or adaptation actions achieved by the end user. These actions could be interactive tasks in the course of the interactive system itself or manual tasks that are outside the system, but yet perceivable (e.g., thanks to camera-based computer vision, situation and activity detection).

Capacity, autonomy, and perceptibility are three local quality properties expressing system abilities. In order to refine these properties in terms of potential benefits for the end user, further sub-properties could be introduced:

- **Accuracy**: refers to the degree to which perceptibility is achieved. When an adaptive system accurately perceives the user and her context of use, it provides end users with some comfort and trust.

- **Adequacy**: refers to the degree to which autonomy is achieved. When an adaptive system adequately decides an adaptation that is considered suitable for the end user, it provides end users with some subjective satisfaction.

- **Stability**: refers to the degree to which capacity is achieved. When an adaptive system has to ensure a high degree of stability in the interface adaptation, it means that it should be capable of ensuring consistent adaptation actions that are subject to a smooth transition from the status before adaptation to the status after adaptation.

Some classical properties could be represented on top of the design space, such as various forms of feedback (Fig. 11) [36] and self-management properties (Fig. 12) [34]:

- **Feedback**: refers to any entity’s ability to provide the other entity with any information in return to an action executed by the other entity, whether it is the user or the system [14]. When the end user executes an action (in her PDA cycle), the system needs to perceive it (in its PDA cycle) and the user should be provided with some immediate feedback (e.g., an information message, a process running), which is covered by system feedback since the system is responsible. When the system executes an action (in its PDA cycle), the end user should be notified as well, which is covered by system feedback by the end user (e.g., by a progress bar, by acknowledging a command, by accepting an adaptation proposal). Whereas the feedback always occurs after an action has been carried out, preferably immediately after termination, the next property should occur even before any action could be undertaken or while an action is being formulated by the end user. For instance, a pen-based gesture could display some feedback indicating the recognition results before initiating the corresponding command.

- **Feedforward**: refers to any entity’s ability to provide the other entity with any information before this last entity will execute any action [14]. **User feedforward** occurs when the system produces any action just before the end user will do her task, thus helping her to decide whether this task is indeed appropriate. For instance, the system shows possible pen-based gestures while the user is producing them, thus providing feedback before the final gesture is produced. On the contrary, **system feedforward** occurs when the end user produces any action just before the system will initiate a task. For instance, a wizard may be redirected by a user action while running, thus dispatching to another branch.

**Self-management** in general refers to a set of system abilities to deal itself with various issues without requesting any external operation from the end user or another entity [9,34]. These abilities could be again structured along the cycles:

- **Self-reconfiguration**: refers to the system’s ability to reconfigure itself by undertaking appropriate actions. A newly discovered service is added. Self-protection occurs when an adaptive UI is required to protect itself from attacks and end users who inadvertently make software changes and errors.

- **Self-decision**: refers to the system’s ability to decide itself to initiate any action. For instance, dynamic programming exhibits the capability of the system to generate new rules which, when triggered offer new decision opportunities.

- **Self-perception**: refers to the system’s ability to perceive its own functioning. For instance, fault-tolerant system may detect this it is no longer properly working and re-initialize.

- **Self-optimization**: refers to the system’s ability to learn itself from its own knowledge in order to optimize its functioning. An adaptive system must improve its learning and rules applied for adaptation for providing good predictions.

- **Self-prediction**: refers to the system’s ability to predict itself when a self-adaptation may occur. For instance, a context-aware UI could probe the context of use to deduce that an adaptation of its behaviour is likely to happen if the context of use is continuously and regularly accessed.

- **Self-adaptation**: refers to the system’s ability to adapt itself depending on new requirements, whether they are functional or non-functional requirements.

The relevance of adaptive UI strongly depends on the performance of the predictability. The challenge is to make the adaptation predictable to the user while maintain accuracy [20]. For this purpose, the system is required to be stable [19] in order to help users understanding the system and creating a mental model of the UI. In addition, the system should be transparent [22] by explaining the proposed prediction. Also, transparency is required about its learning as well as about the decision made and the chosen format of prediction presentation.
In order to support the six steps of the PDA-LPA cycle, FFUI follows the Intelligent Agile Runtime Adaptation framework [14]. It also provides the flexibility to develop up-to-date adaptive behaviour and compute adjustment with regard to real aspects. FFUI flexibility enables proactive and reactive management of adaptation. We assume that the consideration of hybrid strategy for widget selection allows systems to take advantage of a vast number of approaches potentials. Two directives are established with diverse strategies and perceptions to support FFUI adaptation process. FFUI takes advantage of a typical user-based directive that refers to end-user preferences for widgets selection enhancing the context-awareness of the process and advanced system adaptability through a score-based adaptation improving the system decisions. This mixture is aimed at enhancing a context-aware widget selection process through overlapping different prospect improving the decision-making. FFUI supports the explicit feedback by integrating a proactive-behavior involving end users in the adaptation decision-making process and supporting an intelligent user-centered widget selection. FFUI enables the user to take full control of the interface while assisting them by recommending appropriate widgets. Allowing user control is intended to be coherent with the whole interactive session in order to avoid overloading user and to do not interrupt the main user task. Those reflections are considered advantageous to encourage users to invest in personalizing their UI versus auto-personalization [41].

In order to support the six steps of the PDA-LPA cycle, FFUI relies on a machine learning algorithm based on a scoring function determining the weight of each widget candidate. Initially, each widget candidate (e.g., an edit box vs an accumulator) in Fig. 14 is assigned to a default score according to a default scoring function which is defined as (Fig. 15):

$$\text{Default Score}_\text{(Widget)} = P*SC + D*SU$$

where $SC$ is the score of change which defines the additional weight assigned to a widget after being selected by a user or a designer for a specified form. Such score allows the promoting/demoting of widgets with regard to users and experts.

$SU$ is the score of unchanged which defines the interest accorded to the system choice in term of rewarding a well-behaved recommendation within a reinforcement-learning paradigm.

$P$ denotes the number of times that the widget is selected by the end user without being displayed by default ($W_{Selected} = W_{Default}$).

$D$ denotes the number of times that the widget selected by the end user is the displayed one ($W_{Selected} \neq W_{Default}$).

<table>
<thead>
<tr>
<th>System</th>
<th>Perception</th>
<th>Decision</th>
<th>Action</th>
<th>Learning</th>
<th>Predicting</th>
<th>Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-consciousness and behavior</td>
<td>Perception predictability and controllability</td>
<td>Decision predictability and controllability</td>
<td>Action predictability and controllability</td>
<td>Learning predictability and controllability</td>
<td>Predicting predictability and controllability</td>
<td>Adaptation predictability and controllability</td>
</tr>
<tr>
<td>System</td>
<td>Perception</td>
<td>Decision</td>
<td>Action</td>
<td>Learning</td>
<td>Predicting</td>
<td>Adaptation</td>
</tr>
<tr>
<td>Self-conscioussness and behavior</td>
<td>Perception predictability and controllability</td>
<td>Decision predictability and controllability</td>
<td>Action predictability and controllability</td>
<td>Learning predictability and controllability</td>
<td>Predicting predictability and controllability</td>
<td>Adaptation predictability and controllability</td>
</tr>
</tbody>
</table>

Fig. 13. Table of property mappings between user and system.

Fig. 14. Changing the widget selection for a form field.
In order to integrate all these choices, a global scoring function is defined for recommending any widget (Fig. 16) [31]:

$$\text{Score}_{\text{widget}} = P \times SC + D \times SU + T \times SG + f(w, SA)$$

where $f(w, SA)$ determines whether the selected widget matches the designer recommendation and $T$ is the total number of widget selections.

$SA$ is the designer’s score assigned to the widget in this context based on usability engineering and guidelines, such as graceful degradation rules [18].

$SG$ is the global score based on previous options.

The global scoring function is continuously computed while the end user is interacting with the form. Of course, the scoring function can be tailored to another formula (Fig. 16) whether a most accurate schema could be determined and the recommendation can be activated or deactivated. Each end user action is recorded in a log file that is attached to any particular widget of any particular form, thus making it unique for each context.

B. Adaptive GUI layout

We developed Task-Based Design Adaptive (TABADA), a software enabling end users to carry out interactive tasks as explicitly defined in a task model with run-time adaptive layout based on machine learning. Our approach follows existing model-based work specifying the UI at a higher level of abstraction [10,38]. We consider task-oriented language (task tree) and an Abstract UI specification (AUI) in order to remain model-based and to allow for a greater flexibility in generating UIs from abstract levels [10]. TABADA is distinctive by the use of machine learning techniques for runtime adaptation and their deployment together within a model based approaches. TABADA exploits the user behaviour prediction to improve the arrangement of abstract interaction units at the abstract user interface level. Then all data collected by implicit feedback are used into a module called user behaviour predictor. This module uses a machine learning technique based on statistics in order to predict the next action(s) that will be accomplished by the user given the previous ones he filled. The prediction is implemented via a `UserActionPrediction` class, and can be seen as an extension of the context of use where the data are processed to extract more useful data. This class needs an instance of `ActionMonitoringDB` as “raw material” and also takes as parameter the Markov order. The process of user behaviour predictor is based on Markov chains as follows:

- Generating and monitoring sequences of actions based on various parameters (Fig. 17).
- Learning an $n$ order Markov chain model (or all the order from one to $n$).
- Predict the next most probable action of the user thanks to its history of immediate action.

When TABADA is executed, a first by-default GUI layout is generated based on a task model. All end users actions are then recorded (e.g., filling in a field, selecting a new tab, making a choice – Fig. 18) so as to feed the above algorithm. Based on used and unused parts of the task model and based on interaction traces, TABADA computes the most probable interaction paths. At any time, the end user can stop the system and ask for alternate adapted GUI layout that better suit her task (Fig. 17).
V. DESIGN SPACE EXPLORATION

Design space exploration [24,26] consists of systematically exploring all steps of a design space to determine to what extent two or more instantiations may be similar or different and to give rise to a complete consideration of all possibilities. In this section, we will primarily focus on the comparison of FFUI versus TABADA. Fig. 19, respectively Fig. 20, represents the PDA-LPA design space instantiated for FFUI, respectively for TABADA.

In FFUI, every user action \( (A_u) \) is recorded in a log file \( (P_s) \) that automatically triggers the computation \( (A_s) \) of a new score. Since this function is always computed for every widget affected by the action, the system decision \( (D_s) \) is bypassed. The end user may perceive the widgets candidates \( (P_u) \) by entering in FFUI design mode (like in Fig. 14) by pressing a control key, then decide whether a new widget should be selected \( (D_u) \). This decision in then translated into a new action \( (A_u) \).

This new user action enters in a new cycle, thus triggering a new score computation \( (A_s) \), which in turn reinforces the learning \( (L_s) \): the system learns that the end user has preferred to rely on another widget (SC is updated) or not (SU is updated), which is reflected in a new score computation \( (P_s) \). The system then performs the required adaptation \( (A_s) \) by saving the widget current state (e.g., its current value, its possible values), by substituting the old widget by the new one [18], and by restoring the current state into the newly selected widget. A new loop is then generated and so on. The end user does not have any genuine LPD cycle since it is very hard to predict which widget could be selected depending on all parameters. It is primarily an on-demand isolated selection.

In TABADA, every end user action (e.g., using a particular widget, navigating between views, changing the value of a field) is recorded by a sequence monitor \( (P_s) \) which records all sequences on top on an internally-maintained task model, which is unfortunately invisible to the end user. Similarly to FFUI, TABADA does not decide to perform any adaptation (no \( D_s \)) and always automatically generate a series of alternate layouts based on previously recorded sequences \( (A_s) \). These alternate layouts could be made observable on-demand by the end user \( (P_s) \), among which the end user may pick one layout or not \( (D_s) \), and requests \( (A_u) \) the system to switch to this alternate layout. This request is in turn recorded in the system \( (P_s) \), which refreshes the generation of sequences \( (A_s) \) and updates the Markov chain accordingly \( (L_s) \) that again identifies the most probable sequences \( (P_s) \). A new loop is generated and so on. Note that in this case, there is a minimal LPD for the end user since she was conscious of the new layout selected based on previous task sequences, thus learning \( (L_s) \). This may help her to predict how a new layout could be computed \( (P_s) \).
Table 1 provides an overview of how both FFUI and TABADA are addressing the quality properties defined on the design space. For both FFUI and TABADA, observability is assessed as medium since system and adaptation observability are ensured by browsability: the end user does not immediately see that there is an adaptivity process ongoing, but could access to the adaptivity control panel by clicking on a control key in FFUI or on an icon in TABADA. Some users even do not notice the control key and the icon. Nothing else informs the end user that some adaptivity is ensured by the system, thus suggesting that this property should be largely improved. Intelligibility is assessed as superior in TABADA than in FFUI because the alter-
nation an icon in TABADA. Some users even do not notice the con-
trol key and the icon. Nothing else informs the end user that
some adaptivity is ensured by the system, thus suggesting that
this property should be largely improved. Intelligibility is as-
essed as excellent in both cases since the new adapted lay-
out will be exactly the one built with the new adaptation deci-
sion. Adequacy cannot be estimated at this stage because it
should require a user experiment to determine whether the ad-
aptation proposals are adequate enough for the end user. What
could be captured however is the extent to which the system
proposes candidates that are accepted or rejected by the end
user. Stability is always ensured because the same algorithm
is always applied in both cases, although we could imagine that
different algorithms for adaptation (e.g., based on different ma-
chine learning algorithms) could be competing.

Self-management properties are almost not ensured since both
systems apply adaptation only after an end user decision. How-
ever, both systems are able to perceive their own state (self-perception) and are still able to make predictions at any time
(self-prediction) without waiting for the end user, but without
any ability to apply what has been predicted, hence self-
optimisation is non-existent.

In conclusion, several quality properties are affected by the
constraint imposed by the system: only the user can make a de-
cision of what prediction could be applied. The system cannot
take any decision, therefore preventing mixed-initiative. This
generates a new suggestion for both systems: how and when to
delegate adaptivity to the system and/or embark into a conver-
sation between the user and the system to decide which candi-
date is the most appropriate based on respective knowledges.

VI. Conclusion

In this paper, we presented a design space for adaptive user in-
terfaces that depart from existing design spaces (e.g.,
[9,17,19,35]) in terms of quality properties from a software
viewpoint instead of a set of independent, not interconnected
properties (e.g., stability, visibility). This design space is ex-
plcitly based on the Perception-Decision-Action (PDA) cycle
coming from cognitive psychology [33], which is itself aug-
mented by a second cycle Learning-Prediction-Action (LPA).
This design space supports the three expected virtues: descript-
ive, comparative, and generative.

Future avenues of this work include: (i) the introduction of
time to qualify the time constraints between the PDA-LPA
steps and to determine, which is the most appropriate moment
to ensure them, not always continuously; (ii) the conducting of
a Systematic Literature Review (SLR) on a series of papers like
in [13], but on a recent base of references; and (iii) the refining
of the Prediction step into three sub-steps according to End-
elsey’s model of situational awareness where situation awareness
is decomposed into perception, comprehension, and projection.

<table>
<thead>
<tr>
<th>Quality property</th>
<th>FFUI</th>
<th>TABADA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observability</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Intelligibility</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Controllability</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Predictability</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Awareness</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Decidability</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Triggerability</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Capacity</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Autonomy</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Perceptibility</td>
<td>🟢</td>
<td>☢</td>
</tr>
<tr>
<td>Accuracy</td>
<td>🟢</td>
<td>☢</td>
</tr>
<tr>
<td>Adequacy</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Stability</td>
<td>🟢</td>
<td>☢</td>
</tr>
<tr>
<td>Self-reconfiguration</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Self-decision</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Self-perception</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Self-adaptation</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Self-prediction</td>
<td>☢</td>
<td>☢</td>
</tr>
<tr>
<td>Self-optimisation</td>
<td>☢</td>
<td>☢</td>
</tr>
</tbody>
</table>

Table 1. Comparison of FFUI and TABADA in terms of quality properties.