Probabilistic GUI Representations for Adaptive User Interfaces

Daniel Buschek
daniel.buschek@uni-bayreuth.de
Research Group HCI + AI, Department of Computer Science, University of Bayreuth
Bayreuth, Germany

ABSTRACT

This workshop position paper outlines how to facilitate dynamic adaptations of graphical user interfaces by introducing a fundamentally different, probabilistic way of representing GUI elements. As a case study for such a representational framework, we reflect on our previous work on *ProbUI* [2]. In *ProbUI*, each GUI element is represented by one or more probabilistic graphical models. We motivate this by highlighting limitations of current deterministic GUIs. We then discuss examples of adaptive mobile touch GUI widgets implemented with *ProbUI* and reflect on the framewok according to the workshop's classification criteria. Finally, we conclude with ideas for future work on embedding probabilistic representations and reasoning directly into user interfaces.

CCS CONCEPTS

• **Human-centered computing → User interface toolkits**.

KEYWORDS

GUI framework, probabilistic modelling, machine learning

ACM Reference Format:
Daniel Buschek. 2020. Probabilistic GUI Representations for Adaptive User Interfaces. In *IUI '20 Workshops, March 17, 2020, Cagliari, Italy*. ACM, New York, NY, USA, 4 pages.

1 INTRODUCTION

Typically, the elements of graphical user interfaces (GUIs) today are specified and implemented in a deterministic fashion with regard to how they handle user input: This shows in particular in how GUI elements, such as buttons or sliders, are represented internally (Figure 1): Each such GUI element is modelled as a single "bounding box", that is, a rectangular area on the screen. Interpreting the user's input with this model is simply done by checking whether the user's finger touch point (x, y) falls inside such a bounding box. If this is the case, the corresponding GUI element is activated and the associated command is executed.

This deterministic box model is simple to implement and to check against input events and it has been the dominant GUI representation used in practice for many years (e.g. mobile web, Android, iOS). However, it comes with a set of limitations with regard to building adaptive user interfaces. The key limitations are:

IUI '20 Workshops, March 17, 2020, Cagliari, Italy

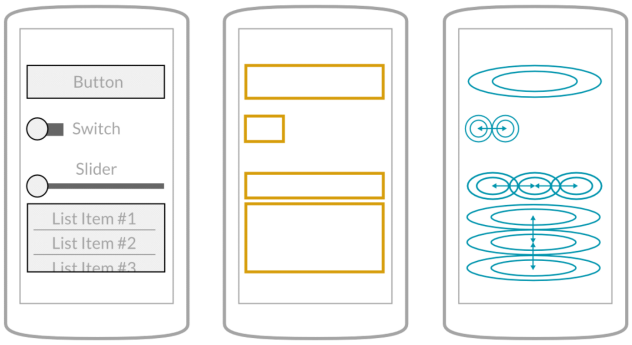

Figure 1: *Left:* A mobile GUI. *Centre:* In the widely used box model each GUI element is represented by one rectangle (bounding box). User input is then handled by checking if a touch point (x, y) falls inside a box, and the corresponding element is activated. *Right:* In *ProbUI*, each GUI element instead is represented by one or more probabilistic graphical models of touch behaviour ("bounding behaviours"). This position paper reflects on how this probabilistic GUI representation facilitates dynamic adaptations of the GUI.

- *Boxes cannot represent sequential input well:* There are many input actions that are not well represented by a single box, such as gestures (e.g. finger sliding on a touchscreen does not yield a single touch point to check with a box). Overall, crude representations of expected user behaviour limit the system's potential for adaptation, for example, to variations in said behaviour.
- *Boxes have a one-to-one mapping with GUI elements:* There is traditionally just one bounding box per GUI element. This limits implementing UI adaptations, for example, for adaptations which have the goal of accounting for different possible ways of using the UI (e.g. adaptation to hand posture, such as thumb vs index finger use).
- *Bounding boxes lack a notion of uncertainty:* In the box view, an input event such as a finger touch falls either in or out of a box. Thus, a bounding box itself does not support handling uncertainty and probabilistic reasoning. This is a limitation for implementing adaptive UIs which are often intended to react adequately to potentially uncertain user input (cf. [7]).

For for mobile touch devices in particular these limitations seem of special importance given that in mobile contexts 1) uncertainty is paramount and 2) changing everyday contexts render GUI adaptation especially interesting and potentially useful. Supporting this, the literature presents user- and context-specific adaptations in (mobile) touch interfaces. Keyboards in particular are a common

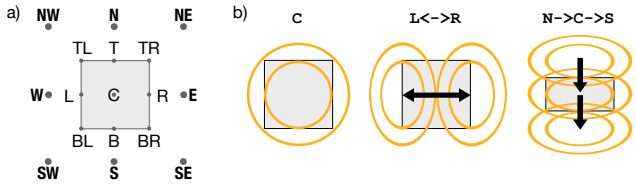

Figure 2: Area tokens used in *ProbUI*'s declarative language: Developers specify short touch behaviours by chaining these tokens. *ProbUI* then maps these to simple probabilistic graphical models (HMMs), as visualised. These HMMs then represent GUI elements instead of the traditional bounding boxes, thus supporting probabilistic inference and GUI adaptations.

target for such research efforts: For instance, related work presented keyboards which adapt to walking [4] and ways of holding the device [3, 5], as well as combinations of such factors [8].

To facilitate the realisation of adaptive (mobile) GUIs more generally, also beyond keyboards, our *ProbUI* framework [2] addresses the shortcomings of the box model listed above. To achieve this it introduces a generalised representation for GUI elements, using probabilistic graphical models instead of bounding boxes.

2 FRAMEWORK AND EXAMPLES

Overall, from a developer's perspective *ProbUI* is used as follows:

First, developers define input behaviours (e.g. for touch input: "tap", "slide right", etc.) via a declarative language (Figure 2) and attach these to specific GUI elements. In analogy to bounding boxes, and to highlight the generalisation in *ProbUI*, we refer to these input behaviours as "boundig behaviours". *ProbUI* then internally maps these to simple probabilistic graphical models (Hidden Markov Models, HMMs [1]). This mapping is a key contribution of the framework [2]. It allows developers to build probabilistic GUIs without having to become experts in HMMs or similar models.

For illustration, the following code snippet specifies a button that counts the number of times a user swipes on it left to right (e.g. as in a "slide to unlock" lockscreen widget). Crucially, this button is then represented with a probabilistic model of said swipe although in the code we did not have to set up an HMM manually.

```
public class MyButton extends ProbUIButton {
    private int counter;
    // Called when setting up the GUI:
    public void onProbSetup() {
        // Add a bounding behaviour to this button:
        this.core.addBehaviour("swipe_right: L->C->R");
        // Add a rule with callback:
        this.core.addRule("across on complete
        and across is most_likely",
            new RuleListener() {
                public void onSatisfied() {
                    counter++;
        }});
    }
    // ...
```

Second, developers can refer to the probabilistic information provided by *ProbUI* anywhere in their UI code, for example, to implement UI adaptations and feedback/feedforward. As a minimal example, the code snippet below maps the probability of the swipe to the GUI element's opacity.

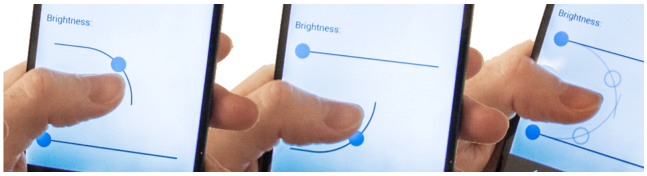

Figure 3: Adaptive slider impltemented with *ProbUI*. *Left:* While the user is sliding their thumb on the screen, the slider continuously adapts its shape to match the thumb's movement arc and thus its reachable area. This uses probabilistic information provided by *ProbUI*, in particular, to decide which way to bend the slider. *Centre:* The framework automatically handles determining which slider to activate, based on the probabilistic information (e.g. here the bottom slider is more likely to be used in an updwards motion, compared to the top slider being used in a downwards motion). *Right:* Uncertainty in target selection can be displayed as feedback/feedforward information, such as via transparent slider previews here.

```
public void drawSpecific(Canvas canvas) {

    double prob = this.core.getBehaviourProb("swipe_right");
    this.paint.setAlpha((int) (prob * 255));
    // ... draw GUI element
```

For a more realistic and useful example, see the slider widget in Figure 3. Further examples can be found in the paper [2].

During use, the framework continuously updates the probability of each bounding behaviour being currently performed by the user. It also continuously updates each GUI element's probability of the user currently intending to activate this element. Together, these probabilities support the implementation of adaptive GUIs, as in the slider example.

3 REFLECTION

The workshop call[1] motivated reflection on several aspects, which we address in the following paragraphs.

3.1 Input

Since *ProbUI* as implemented for Android targets mobile touch GUIs, the main input is finger touch – both single touch events ("taps") as well as touch sequences (e.g. "swipe", "slide") or generally touch "gestures". However, the conceptual framework itself is flexible and could account for other input modalities in the future, for example IMU sensors (e.g. tilting or shaking the device as input).

3.2 Technique

3.2.1 Model. *ProbUI* uses a Hidden Markov Model to represent a touch behaviour. It can handle multiple such behaviours and thus multiple HMMs for a single GUI element, and of course the whole

[1] https://ai4aui.wordpress.com/

GUI likely consists of multiple such GUI elements. Thus, the whole GUI is represented by a set of sets of HMMs.

3.2.2 Inference. Inference is conducted during interaction in a Bayesian fashion, by 1) using the HMMs to evalute the likelihood of the current touch input per expected behaviour, 2) inverting this with Bayes rule to find the likelihood of each behaviour for the given input (i.e. leading to a posterior on the question of *Which touch gesture is the user performing?*), and 3) integrating over these likelihhods for all behaviours per GUI element to get the overall likelihood per element (i.e. leading to a posterior on the question of *Which GUI element is the user targeting?*). See the *ProbUI* paper for formal details [2].

3.2.3 Learning. One unusual aspect of *ProbUI* in its current form is that while it applies a machine learning technique it does not learn from data. The framework as implemented instead supports developers in manually specifying the HMM models (without requiring knowledge of HMMs or probabilistic modelling) by introducing a simple declarative modelling language. This language enables developers to write down, for example, that a certain GUI element should react to a "tap" and a "slide right". *ProbUI* then maps these declarative statements to HMMs internally, additionally informed by defaults from the literature (e.g. about the spread of touch points around the target).

This decision supports direct integration in GUI frameworks as no external data collection and training is required. Nevertheless, conceptually, *ProbUI* can directly be extended to 1) instead use HMMs fitted to recorded data, and 2) update HMMs as new touch data is observed. In the latter case, the developer's declarative statements can be seen as specifying a prior on the probabilistic GUI representation before observing user input.

3.3 Output

ProbUI outputs two sets of probabilities, updated live during interaction. In particular, at each new touch event *ProbUI* provides 1) one probability distribution for each GUI element describing how likely each of that element's expected behaviours currently is; and 2) one probability distribtion across all GUI elements describing how likely each element is the currently intended target of the user.

Developers can then use these probabilities to implement GUI adaptations, feedback / feedforwad, decision-making, and so on.

3.4 UI Adaptation

ProbUI does not directly specify or implement a certain kind of UI adaptation and level of automation since it is a general framework for probabilistic GUIs. Hence, in many cases it depends on what the developers do with the probabilistic information provided by *ProbUI*. However, as such *ProbUI* particularly facilitates adaptations that respond to variations in input behaviour (as opposed to e.g. adapting to strategic usage patterns such as usage frequencies etc.). Since it targets GUIs, adaptations typically will be visual.

Regarding the time of adaptations, *ProbUI* supports continuous UI changes, i.e. dynamic adaptations, since it updates its provided probabilities at each new touch event.

3.5 Adressed Properties of Software Quality

The key motivation for the declarative specification of input behaviour models in *ProbUI* is ease-of-use for the developer who might not be an expert in probabilistic modelling. In this sense, *ProbUI* addresses the implementation process.

In terms of software qualities such as robustness and performance, the current implementation of *ProbUI* is on an advanced prototype level: While it is mature enough to support building prototypes for research and user studies it is not engineered for maximum performance. In particular, the inference framework lends itself to future optimisation via parallelisation: For example, for incoming touch data, each HMM could be evaluated in a separate thread.

3.6 Benefits and Drawbacks

As a general key benefit, *ProbUI* faciliates building probabilistic GUIs, that is, GUIs that directly embed probabilistic handling of user input and corresponding reasoning with regard to user intention (which behaviour and which target).

The benefits and drawbacks of specific adaptions (e.g. layout changes depending on hand posture) likely depend more on those adaptations than the underlying framework that is *ProbUI*.

However, general limitations of *ProbUI* in this context include 1) not directly learning from data (in the current implementation), 2) limited expressivity of the declarative language (i.e. limited gesture complexity that can be expressed), and 3) higher computational costs compared to traditional GUI representations with bounding boxes. Moreover, using the framework requires some introduction (also see the analysis with developers in the *ProbUI* paper [2]).

4 OUTLOOK

As an outlook for the concepts introduced with *ProbUI*, we highlight in particular the following aspects: First, the concept could be extended to learn from user behaviour data, for example, to update the models over time, as user(-specific) input behaviour data is observed. Second, the concept could integrate further modalities, both on mobile devices, as well as part of a transfer to different devices and contexts. Conceptually, the idea of a "bounding behaviour" specifies a meaningful region in any behaviour space and thus in principle may be used for any input modality. Third, other models than HMMs could be explored, in particular also those that may account for more cognitive aspects (e.g. decision-making), to go beyond modelling the physical aspects of input behaviour.

Finally, fundamentally, *ProbUI* embeds generative models of input behaviour directly in the GUI representation. This essentially means that such UIs come equipped with their own usage simulations. Future work could build on this idea: UIs that can "use themselves" seem highly relevant from a perspective of computational interaction [6], for example, with regard to computational optimisation, automated UI testing, or automated generation of usage explanations.

REFERENCES

[1] David Barber. 2012. *Bayesian Rreasoning and Machine Learning.* Cambridge University Press. http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/090310.pdf

[2] Daniel Buschek and Florian Alt. 2017. ProbUI: Generalising Touch Target Representations to Enable Declarative Gesture Definition for Probabilistic GUIs. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4640–4653. https://doi.org/10.1145/3025453.3025502

[3] Daniel Buschek, Oliver Schoenleben, and Antti Oulasvirta. 2014. Improving Accuracy in Back-of-device Multitouch Typing: A Clustering-based Approach to Keyboard Updating. In *Proceedings of the 19th International Conference on Intelligent User Interfaces (IUI '14)*. ACM, New York, NY, USA, 57–66. https://doi.org/10.1145/2557500.2557501

[4] Mayank Goel, Leah Findlater, and Jacob Wobbrock. 2012. WalkType: Using Accelerometer Data to Accomodate Situational Impairments in Mobile Touch Screen Text Entry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2687–2696. https://doi.org/10.1145/2207676.2208662

[5] Mayank Goel, Alex Jansen, Travis Mandel, Shwetak N. Patel, and Jacob O. Wobbrock. 2013. ContextType: Using Hand Posture Information to Improve Mobile Touch Screen Text Entry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2795–2798. https://doi.org/10.1145/2470654.2481386

[6] Antti Oulasvirta, Per Ola Kristensson, Xiaojun Bi, and Andrew Howes (Eds.). 2018. *Computational Interaction*. Oxford University Press.

[7] John Williamson. 2006. *Continuous Uncertain Interaction*. Ph.D. Dissertation. University of Glasgow.

[8] Ying Yin, Tom Yu Ouyang, Kurt Partridge, and Shumin Zhai. 2013. Making Touchscreen Keyboards Adaptive to Keys, Hand Postures, and Individuals: A Hierarchical Spatial Backoff Model Approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2775–2784. https://doi.org/10.1145/2470654.2481384