ProbUI: Generalising Touch Target Representations to Enable Declarative Gesture Definition for Probabilistic GUIs

Daniel Buschek
University of Munich (LMU)
Amalienstr. 17, 80333 Munich, Germany
daniel.buschek@ifi.lmu.de

Florian Alt
University of Munich (LMU)
Amalienstr. 17, 80333 Munich, Germany
florian.alt@ifi.lmu.de

ABSTRACT

We present *ProbUI*, a mobile touch GUI framework that merges ease of use of declarative gesture definition with the benefits of probabilistic reasoning. It helps developers to handle uncertain input and implement feedback and GUI adaptations. *ProbUI* replaces today's static target models (bounding boxes) with probabilistic gestures ("bounding behaviours"). It is the first touch GUI framework to unite concepts from three areas of related work: 1) Developers *declaratively define* touch behaviours for GUI targets. As a key insight, the declarations imply simple probabilistic models (HMMs with 2D Gaussian emissions). 2) *ProbUI* derives these models automatically to *evaluate* users' touch sequences. 3) It then *infers* intended behaviour and target. Developers bind callbacks to gesture progress, completion, and other conditions. We show *ProbUI's* value by implementing existing and novel widgets, and report developer feedback from a survey and a lab study.

Author Keywords

Touch Gestures; GUI Framework; Probabilistic Modelling

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation (e.g. HCI): Input devices and strategies (e.g. mouse, touchscreen)

INTRODUCTION

GUIs today define targets as rectangles, but touch is often dynamic (e.g. slide [37, 55], cross [2, 40], rub [45], encircle [14, 27]). This box model is also challenged by ambiguity: If a finger occludes two buttons, it is unclear if the user really wants to trigger the one whose box includes the touch point (x, y), especially if the finger moved, willingly or due to mobile use. Research addressed this with probabilistic touch GUI frameworks [26, 34, 35, 48, 49, 50], which offer attractive benefits:

Reasoning under uncertainty: While a touch missing all bounding boxes (even slightly) is ignored, probabilistic GUIs may react by inferring a user's most likely intention (e.g. [9]).

Continuous feedback: This can close the control loop between user and system [54]. For example, visualising each target's activation probability (e.g. via opacity) reveals the system's current belief about the user's intention.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
CHI 2017, May 06 - 11, 2017, Denver, CO, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4655-9/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3025453.3025502

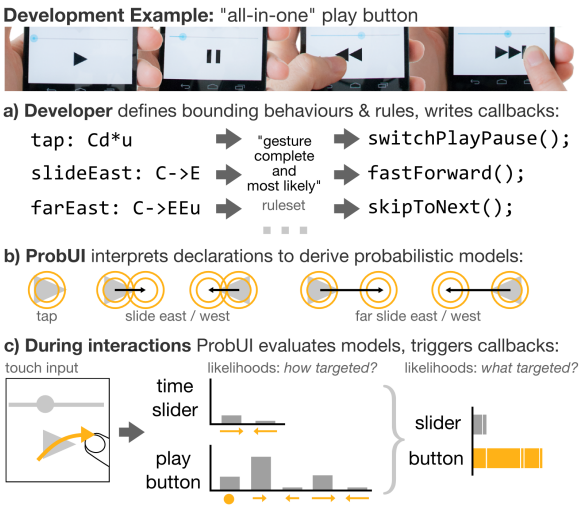

Figure 1. Basic example of using *ProbUI* to develop a novel widget that probabilistically reacts to different touch behaviours: *(a) ProbUI* offers a declarative language to represent GUI targets via gestures ("bounding behaviours"). Developers also define rules on these behaviours and attach callbacks. *(b) ProbUI* derives probabilistic gesture models from these declarations. *(c)* During use, it evaluates each behaviour's probability. The likelihood of a target's set of behaviours yields its activation probability, used to reach decisions and trigger callbacks. Developers can use the probabilities for reasoning, feedback, and GUI adaptations.

GUI adaptation: Instead of triggering actions based on touch-in-box tests, probabilistic representations enable mediation (e.g. [35, 50]), for example via presenting previews, alternatives, or opportunities for users to cancel or clarify input.

Since GUIs today use boxes (e.g. Android[1], iOS[2], web[3]), they cannot assign to inputs any probabilities. To still use probabilistic GUI frameworks, developers thus first have to provide probabilities, for example from probabilistic gesture recognisers. However, these are not directly integrated into GUIs and require training data or external editors [1, 7, 13, 31, 32, 33]. Gestures can also be easier specified via declaration, yet this does not yield probabilities [28, 29, 30, 46]. To facilitate creating probabilistic GUIs, we propose a concept that merges ease of use of declarative gesture definition with benefits of probabilistic reasoning (Figure 1). We contribute: 1) a concept to represent GUI targets via gesture models instead of static geometry tests; 2) a declarative language to define such gestures, with a method to automatically derive probabilistic models; 3) an implementation of our approach in Android.

[1] http://developer.android.com/guide/practices/ui_guidelines/widget_design.html#anatomy
[2] https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/index.html
[3] http://www.w3.org/TR/CSS21/box.html, *all last accessed 21st September 2016*

APPROACH: INTEGRATING THREE KEY AREAS

ProbUI is the first probabilistic touch GUI framework to integrate three related areas into a single system: 1) *defining* touch behaviours declaratively, 2) *evaluating* them probabilistically, and 3) *inferring* intention from such probabilities.

1) Defining input behaviour: Developers define one or more "bounding behaviours" (BBs) per GUI element. Our novel approach then automatically derives probabilistic gesture models from the developers' non-probabilistic definitions.

2) Evaluating behaviour: These models generalise previously used target representations (boxes, single Gaussians) from areas to areas over time, and from one to many behaviours: For example, a widget might react (differently) to taps and slides.

3) Inferring user intention: ProbUI infers intention, for example to allow the most likely widget to trigger an action associated with its most likely BB. It also offers callbacks on gesture progress, completion and context (e.g. speed, pressure, touch size). This helps developers to create suitable reactions, in particular via continuous feedback and GUI adaptations.

RELATED WORK

We discuss the three main related areas and how *ProbUI* addresses open challenges by adopting and integrating their key concepts into one pipeline.

Defining Touch Gestures Declaratively

Strengths: Ease of Use of Declarative Gesture Specifications

In declarative gesture frameworks [28, 29, 30, 46], developers create gestures by describing them. For example, *Proton* [29, 30] uses regular expressions composed of tokens like touch down/move/up. This offers a concise and relatively easy-to-read way of creating gestures, and does not require training data. We follow this approach motivated by this ease of use.

Challenges: Behaviour Variations and Uncertainty

Declarative frameworks generally do not provide probabilities [28, 29, 30, 46]. This makes it hard to react to varying behaviour, uncertain intention, and to give live feedback on potential outcomes. *Proton* thus asked developers to manually implement "confidence calculators" [29, 30]. This motivates our approach: We automatically derive probabilistic models from declarative statements to provide input probabilities.

ProbUI: Declarative Language that Implies Prob. Models

In contrast to *Proton*, we use gestures as part of defining a GUI, not the GUI as a part of defining gestures: For example, *Proton* shows a gesture "drag the star" [29, 30]. In *ProbUI* we would attach a "drag" to the star itself. In other words, we define gestures relative to GUI elements. This enables us to infer parameters of a probabilistic model for said element. Moreover, callbacks for feedback and adaptation logic are thus encapsulated in the widget (e.g directly in the "star" class).

Evaluating Touch Gestures Probabilistically

Strengths: Handling Behaviour Variations and Uncertainty

Given input events, probabilistic recognisers evaluate a set of (learned) gestures (e.g. [1, 7, 13, 32, 33]). We follow such approaches to enable our BBs to handle behaviour variations (e.g. varying trajectories for a sliding behaviour).

Challenges: Efforts for Model Creation and GUI Integration

Probabilistic recognisers require training data (e.g. [1, 7, 13]) and some employ interactive editors to enable capturing even highly complex gestures [31, 32, 33]. In contrast, we use gestures associated with GUI targets, which are relatively simple as related work shows (e.g. cross [2, 40], slide [37, 55], rub [45], encircle [27]). With this focus, we offer a declarative language for "in-line" use in GUI setup, without external editors, code ex/imports, or training data – yet our approach still yields (simple) probabilistic models, as described before.

ProbUI: Method to Derive Prob. Models from Declarations

In summary, probabilistic recognisers handle varying behaviour, but lack direct GUI integration, and creating gestures is often less simple than with declaration. To address this, *ProbUI* derives probabilistic models from non-probabilistic declarations. Hence, GUIs account for behaviour variations, yet developers do not need to be probability experts.

Inferring User Intention from Touch Input

Strengths: Understanding User Intention in Touch GUIs

Probabilistic touch GUI frameworks help to understand user intention, given input probabilities [48, 49, 50]. We also conduct such "mediation" to infer intended behaviours and GUI targets – by evaluating the probabilistic models of our BBs.

Challenges: Obtaining Probabilities for Mediation

Probabilistic touch GUI frameworks so far mostly assumed existing input probabilities [48, 49, 50], without directly supporting developers in implementing methods to obtain them. Hinckley and Wigdor [22] recently concluded that "a key challenge for [uncertain] input will be representation" and "how to make this information available to applications".

For example, beyond taps [47], scoring input is often *informally outlined*: For sliders, Schwarz et al. [48] state that "selection score depends on the distance [...] in addition to the direction of motion". Other work [50] described representing scrolling "using simple gestural features" without formal details. While these tools support many interactions "by varying the method for determining the selection probability" [47], developers gained no recipes to do so. We address this lack of support with our combination of declarative gestures creation and automatic derivation of underlying probabilistic models.

Work on sampling [49] enabled deterministic event handling of probabilistic input. Since in its first step it "takes a probabilistic event", it required external models, like gesture recognisers, "called repeatedly after each new input event". This motivates our method, which avoids external gesture models by deriving them directly from the developer's specified BBs.

ProbUI: Merging Modelling and Mediation

ProbUI is the first touch GUI framework to provide developers with both 1) a declarative way to describe how to *obtain* input probabilities for GUI targets; and 2) a system to *handle* these probabilities to infer user intention. With this integration we aim to streamline probabilistic GUI development, for example avoiding transfer of input events and resulting probabilities between GUI, mediators, and external recognisers.

Bounding Behaviours (BBs)

We now motivate our central concept in light of related work.

Motivation: Many Widgets Require Touch Gestures

Many widgets do not match the simple box model, but rather observe input over time: sliding [37, 55], encircling [14, 27], crossing [2, 40], rubbing [45], double taps [21], and so on. This lack of a sequential/temporal dimension in current target representations motivates our generalisation to BBs.

BBs Facilitate Implementing Gestures for GUI Elements

Linking gestures to GUI targets is inspired by *Grafiti's* [15] "local gestures" for specific areas (e.g. a widget's visuals). *Grafiti* allows developers to implement recognisers for new gestures, but does not directly help with creating these, in contrast to our work. *Grafiti* also considers gesture confidences as optional, while deriving them is at our focus.

BBs Provide Probabilistic GUI Representations

Touch targets are often modelled as 2D Gaussians [3, 8, 9, 48, 51], mostly for keyboards (e.g. [3, 17, 19, 56]), but less so for general touch GUIs, as in *ProbUI*. In contrast to the related work, we use multiple connected Gaussians per target in a touch sequence model (i.e. a Hidden Markov Model).

BBs Allow Widgets to Distinguish Multiple Behaviours

Defining multiple BBs per target is motivated by work such as *Sliding Widgets* [37] and *Escape* [55]: Their targets can receive – and decide to accept or deny – *different* types of behaviour, namely slides in different directions. Moreover, multiple behaviours support different preferred interaction styles, and may cater to contexts and constraints, like casual use [41], walking [6, 38], impaired sight/precision, or thumb use [5].

BBs Support Multiple Behavioural Components

FFitts' Law [8] has two components (Gaussians for precision and speed-accuracy tradeoff) to define one behaviour (tap) and score targets [9]. Our BBs support using such concepts in GUIs. For example, following *FFitts' Law* and its parameters, we could define two single-Gaussian BBs for a set of buttons. *ProbUI* then scores these targets based on both (like [9]).

FRAMEWORK OVERVIEW

Architecture

ProbUI is structured into four layers, motivated by the typical process of development and runtime input handling in GUIs:

Layer I is used by developers to define bounding behaviours, and callbacks that allow widgets to react to them.

Layer II updates each BB's probability (small dots in Figure 2) of being performed by the user, regardless of whether the user intends to use the associated GUI element at all.

Layer III updates each GUI element's probability (large dots in Figure 2), expressing the system's belief that the user currently indeed intends to activate this widget.

Layer IV "manages" the system by: 1) distributing events to interactors and behaviours; 2) triggering callbacks; and 3) mediation – deciding if and which interactor(s) may trigger.

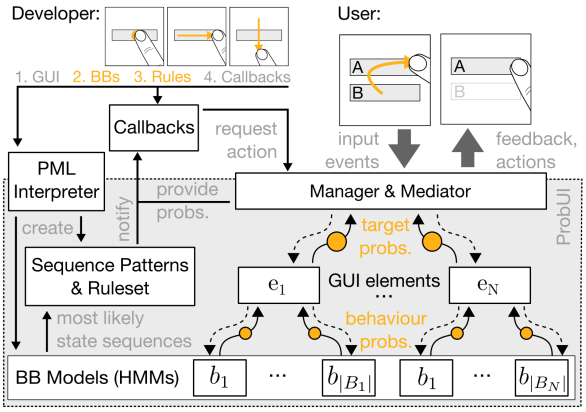

Figure 2. *ProbUI* overview with example: A developer wants to implement a button with several bounding behaviours (top left). She 1) creates a GUI (Android: Java, XML), and defines 2) behaviours and 3) rules (in PML) with 4) callbacks (Android: Java). Our PML interpreter takes her input (e.g. `slide: Ld->Ru`) to 1) create probabilistic models for her BBs, 2) sets up her rules (e.g. `slide on complete`) with her callback references, and 3) stores the implied touch sequence patterns (e.g. "start with down on the left side, end with up to the right"). During use (top right), a manager distributes input events (dashed arrows) to all widgets, to be evaluated by their BB models. Resulting behaviour scores (small dots) are reported back up to derive target scores (large dots). Both can be used in callbacks (e.g. for feedback such as highlighting), and inform decisions by the mediator. Each BB model further infers a most likely state sequence for the given input. These sequences are matched against the stored patterns and rules to decide which callbacks to notify.

Probabilistic Model Overview

We introduce the formal variables used in the next sections: A user's input gesture is a sequence of n touch locations, $t = t_1 t_2 ... t_n$. A GUI has a set of GUI elements E. Each element $e \in E$ has a set of bounding behaviours B_e. Each behaviour $b \in B$ is attached to only one element: $\forall e, f \in E, e \neq f : B_e \cap B_f = \emptyset$, and $B = \bigcup_{e \in E} B_e$ denotes the set of all behaviours of the GUI. Note that multiple behaviours b_i may of course represent the same kind of gesture (e.g. slide right), but each b_i is attached to a different element.

Our model is given by this factorisation of the joint distribution over touch sequences t, behaviours b, and elements e:

$$p(t,b,e) = p(t|b)p(b|e)p(e) \quad (1)$$

- $p(t|b)$ denotes the probability of a touch sequence t given a behaviour b. It is modelled with one HMM per behaviour.
- $p(b|e)$ denotes the probability of a behaviour b given an element e. We define that $p(b|e) > 0$ only if $b \in B_e$ (i.e. if b is attached to e). Per default, $p(b|e)$ is uniform for a given e and $b \in B_e$. Developers may change this.
- $p(e)$ denotes the prior probability of an element e before observing interactions. Per default, all elements are equally likely ($p(e)$ is uniform). Developers can change this.

Using this model and the rules of probability, *ProbUI* infers two distributions from observed touch input: $p(b|e,t)$ the probability of behaviours per element (*how targeted?*); and $p(e|t)$, the probability of the elements (*what targeted?*).

The next sections explain in detail 1) how developers specify this probabilistic model implicitly via our declarative gesture language, and 2) how *ProbUI* then conducts inference.

LAYER I: BOUNDING BEHAVIOURS

Bounding behaviours can be created in four ways: 1) using a preset, 2) defining them in *ProbUI's Modelling Language* (PML), 3) setting the underlying model by hand, or 4) learning it from data. In this paper, we focus on creation via PML for the "default" cases 1) and 2), and consider learning from data as future work. First we describe the probabilistic model that underlies each behaviour, then we explain how *ProbUI* can automatically map from PML expressions to this model.

Bounding Behaviours' Underlying Probabilistic Model

Touch can be imprecise, for example due to finger pitch and roll [23] and perceived input points [24]. Intended touch locations are thus different from the device's sensed locations [12, 20, 52, 53]. To handle this uncertainty, touches t for GUI elements e can be modelled as 2D Gaussians $p(t|e)$ [18, 51], as often used to replace bounding boxes (e.g. [17, 19, 56]):

$$p(t|e) = \mathcal{N}(\mu_e, \Sigma_e), \text{with mean } \mu_e \text{ and covariance } \Sigma_e \quad (2)$$

To generalise from touch areas to sets of touch sequences, we replace the Gaussian (Eq. 2) with one or more gesture models. Each uses one or more Gaussians, linked by transitions over time: a Hidden Markov Model (HMM) [4, 43]. We chose HMMs since 1) they are a well-established model for gestures and 2) they are simple enough to be related to bounding boxes rather directly (Figure 3). This helps to realise an understandable mapping from the declarative language (PML) to the probabilistic model (HMM).

PML: Declarative Statements Define Probabilistic Models

PML merges the best of two worlds: readable declarative gesture definition, and handling behaviour variations via probabilistic models. It offers gesture declaration like *Proton* [29, 30], plus rules like *Midas* [46] and *GDL* [28]. In contrast to prior work, PML also yields probabilistic models (without requiring developers to think about this and without training data). For HMMs, the system thus needs to infer states, transitions, and starting probabilities from the developer's PML expression. The following paragraphs explain this mapping.

As a simple running example[4], we create a button that counts the number of times a user crosses it vertically [2]:

```
public class MyButton extends ProbUIButton {
   // Called by the manager when setting up the GUI:
   public void onProbSetup() {
      // Add a bounding behaviour to this button:
      this.addBehaviour("across: N->C->S");
      // Add a rule with callback:
      this.addRule("across on complete",
         new RuleListener() { public void onSatisfied() {
            counter++;
      }});
   }
   ... // rest of the class
```

Declaring Touch Areas ↦ Probabilistic Model States

Table 1 shows PML's core components. In contrast to *Proton*, a PML expression is linked to a GUI element. Hence, touch locations are interpreted *relative* to the element. This strongly supports the analogy to bounding boxes when using this notation. For example, the default meaning of **N** in PML is "north of the widget's visual bounds" (Figure 4).

[4]We show Android/Java code and **highlight** relevant parts.

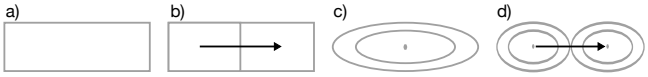

Figure 3. Intuition for replacing bounding boxes with distributions: *(a)* A box describes an area to touch to activate an element. *(b)* Similarly, a slide may be detected with two boxes, triggering if a touch down occurs in the left box, followed by an up event in the right box. *(c)* Intuitively, probabilistic models replace the box with a Gaussian. *(d)* HMMs generalise this to multiple distributions (states), linked by transitions.

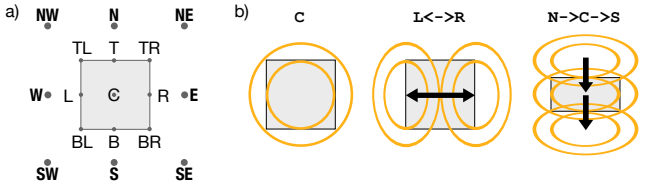

Figure 4. Buttons with *(a)* PML's area tokens and *(b)* examples with resulting HMMs. Circles represent the HMMs' Gaussian states with two/three standard deviations, arrows indicate state transitions. Area tokens can also be stacked. For example, `NN` is twice as far north as `N`.

Expression	Explanation
Touch Area Selectors	
`C`	area on the element (i.e. centred on its visual bounding box)
`T, R, B, L`	sub-area centred at the top/left/right/bottom of the element or current area
`N, E, S, W`	area to the north/east/south/west next to the element or current area
`A[x=?,y=?,w=?,h=?]`	specify area directly with the given centre location and width and height
Area Size Modifiers	
`x, y, z, X, Y, Z`	shrink/enlarge area by half its width/height/both, e.g. "inner centre": `Cz`
`sx, sy, s`	scale area width/height/both, e.g. `C[sx=2]` (twice as wide), or `C[sx=100dp]`
Area Transitions	
`->`	finger moves from one touch area to the next, e.g. from left to right: `L->R`
`<->`	finger moves between two touch areas, e.g. horizontal rubbing: `L<->R`
Touch Event Type Filters	
`d, m, u`	a down/move/up event occurs at the current area, e.g. tap on button: `Cdu`
`*, +`	zero-or-more/one-or-more such events occur, e.g. lift on button: `Cd*u`
Gesture Progress Markers	
`$`	gesture progress notifier, e.g. `Cd$u` fires callback after down but before up
`[Area Selectors].`	gesture end marker, e.g. rubbing has to end on the right: `L<->R.`
`.[Area Selectors]`	gesture start marker, e.g. rubbing should start in centre: `L<->.C<->R`
Other	
`[name]:`	name tag (e.g. `tap: Cdu`), can be referenced in rules (Table 2)
`O`	set gesture origin at first touch down, e.g. horizontal rub anywhere: `O<->E`

Table 1. Core parts of PML for defining bounding behaviours.

Thus, declaring a touch area in PML also defines a location and size for a *state* in the HMM, namely a 2D Gaussian. By default, the "size" is chosen as shown in Figure 4, motivated as a "pessimistic" version of the standard error rate aimed at in Fitts' Law tasks [57]. Developers can fully change this. In our example (**`N->C->S`**), the PML interpreter will thus create three states: one above the button (**`N`**), one centred on it (**`C`**), and one below it (**`S`**), as shown in Figure 4.

As a side-note, PML also provides an "origin" token (**`O`**, Table 1) to interpret areas relative to the *gesture's start* instead of the widget's centre. This is used to define behaviours such as "rubbing *anywhere* on the widget" (see example section).

Declaring Finger Movements ↦ Prob. Model Transitions

We define gestures by chaining touch areas via transitions (Table 1). Hence, declaring finger movements in PML also defines state *transitions* in the HMM: Our interpreter sets non-zero weights for the transitions in the PML statement. It also stores these developer-specified transitions for later rule-checking. It then applies a Laplace correction to the HMM's

transitions so that all states are connected. Otherwise, the model could only ever output one most likely state path (e.g. downwards crossing in our example), regardless of how unlikely that is (e.g. user actually moves the finger upwards).

Declarative Order ↦ Probabilistic Model Starting States
States in an HMM have starting probabilities [4, 43]. By default, our PML interpreter sets a non-zero starting probability for the first (leftmost) state in the expression, and for the rule-system considers the last (rightmost) state as an end-state.

If there are only two-way transitions (`<->`), the interpreter considers both first and last state as start and end states (e.g. a rubbing gesture **`L<->R`** may likely start at either end). However, this can be fully altered with the "**`.`**" token (Table 1).

The interpreter stores the developer-specified start/end states for rule-checking. As for the transitions, we internally apply a Laplace correction to ensure that the HMM can output more than one hypothesis about the most likely starting state.

Evaluating PML Expressions

At runtime, *ProbUI* performs both probabilistic inference and deterministic rule-checking to provide developers with a maximum of information on the user's ongoing interactions.

Probabilistic Inference: Continuous Feedback & Adaptations
Given a touch event stream, *ProbUI* uses the derived HMMs to infer the *input's probability* and its *most likely state sequence*, using the Viterbi algorithm [4, 43]. Input probability describes how well current touch events match the model's "expectations" (i.e. its states and transitions), regardless of gesture progress/completion. For example, our crossing button may already return high probability once the finger is placed above the button (**`N`**), since crossing (**`N->C->S`**) initially expects touches there. Developers can use this information, for example, to provide live feedback, such as previews about likely consequences of completing the ongoing action.

Deterministic Rule-Checking: Triggering Reactions
Complementary, gesture progression and completion is analysed by checking the HMMs inferred most likely state sequence against the developer's specified PML expression (Table 1) and rules (Table 2). Developers bind callbacks to points in a gesture, and to rule states (all updated continuously).

Checking rules: The rule **`across on complete`** in our example triggers our callback when the probabilistically inferred most likely sequence matches our defined sequence (**`N`**, **`C`**, **`S`**) and has just reached a valid end state (**`S`**).

Checking progress: To already react to reaching the centre, for example to play an animation, we can insert a notification marker **`$`** at that point (**`N->C$->S`**) and bind a callback to it:

```
// Add a behaviour with a progress notification marker:
addBehaviour("across: N->C$->S", new BehaviourListener() {
   public void onUpdate(Notifier notifier) {
      if(notifier.isJustReached(0)) //0: first marker
         playAnimation(); }}); // do something
```

Checking touch event types: Event type tokens indicate that a gesture is only complete if such events occur at the specified location in the right order. We could change our example to **`Nd->C->Su`** to only count crossings which started by touching down (**`d`**) above the button, and ended with the finger leaving (**`u`**) below it. In comparison, the previous version also counted crossings as part of slides starting and ending anywhere. Touch move events (**`m`**) are by default always allowed, since it is likely that users move the finger even if they just tap. Hence, a pattern like **`Cdu`** is interpreted as **`Cdm*u`**.

Expression	Explanation
`and, or, not`	boolean and/or/not for sub-rules
`[on|is] complete`	gesture has just been completed / is (already) completed
`[on|is] most_likely`	gesture has just become / is this widget's most likely one
`in [>n|<n|m-n] [ms|s]`	gesture performed in the given amount of time
`with [>n|<n|low|high] p`	gesture performed with the given mean touch pressure
`with [>n|<n|small|large] a`	gesture performed with the given mean touch size
`using [>n|<n|n|m-n] fingers`	gesture performed by the given number of fingers
`[name]:`	name tag, can be referenced in rules

Table 2. PML expressions for defining rules on bounding behaviours.

LAYER II: PROVIDING BEHAVIOUR PROBABILITIES

We have explained how BBs are created and evaluated. Now we describe how *ProbUI* conducts this evaluation: While interacting, layer two continuously evaluates all BBs of all GUI targets. For each target, each of its BBs is assigned the probability of indeed being currently performed by the user.

For example, for a button that triggers by left/right slides (e.g. [37]), layer two continuously updates the probabilities of the user performing a left/right slide. These probabilities can inform feedback while sliding (e.g. transparent preview [48]).

We derive these behaviour probabilities with Bayes' Rule [4], as in similar procedures in probabilistic keyboards [17]. We next explain its three components: prior, likelihood, posterior.

The prior $p(b|e)$ defines relative importances of the BBs per target e, meaning their probability *before* observing touches. A uniform distribution is the default (i.e. all BBs of e equally likely), but we may consider, for example, frequencies (e.g. most used behaviour) or context (e.g. hand posture, walking).

The likelihood $p(t|b)$ of a sequence of (the last) n touch events $t = t_1 t_2 ... t_n$ given the behaviour b is defined by:

$$p(t|b) = \mathrm{HMM}_b(t), \tag{3}$$

where HMM_b denotes the sequence evaluation function (see e.g. [4, 43]) of the HMM that models the behaviour b. Using Bayes' Rule, we compute the posterior over behaviours $b \in B_e$ for target e, given the observed sequence $t = t_1 t_2 ... t_n$ as:

$$p(b|t,e) = \frac{p(t|b)p(b|e)}{\sum_{b_i \in B_e} p(t|b_i)p(b_i|e)} \tag{4}$$

LAYER III: PROVIDING TARGET PROBABILITIES

Layer two evaluated *how* each target is likely used. Layer three complementary derives *which* target(s) the user likely intends to use. We again use Bayes' Rule. The prior $p(e)$ defines the targets' relative importances. It is a uniform distribution by default (i.e. all equally important), yet could also reflect usage (e.g. keyboards often use letter frequencies).

The likelihood $p(t|e)$ of a touch sequence t given element e is obtained by marginalisation over e's behaviours $b_i \in B_e$:

$$p(t|e) = \sum_{b_i \in B_e} p(t|b_i)p(b_i|e) \tag{5}$$

Intuitively, a target e is thus more likely than another one if its bounding behaviours B_e altogether better explain the user's input t. With Bayes' Rule, the probability of e for input t is:

$$p(e|t) = \frac{p(t|e)p(e)}{\sum_{e_i \in E} p(t|e_i)p(e_i)} \quad (6)$$

Note the *generalisation* of target representations: Instead of a binary point-in-box test, for each target we get a *probability* (Eq. 6) based on a *set* of *sequence* models (B_e in Eq. 5).

LAYER IV: MANAGEMENT AND MEDIATION

This "management" layer passes on the fingers' touch events to all elements, to be processed on layers three and two. It also updates rules, triggers callbacks, and decides when to activate which element ("mediation"). As GUI management is mostly "bookkeeping" work, we here focus on explaining mediation.

We overall adopt mediation concepts from related work [34, 35, 48, 49]. This is also a plug-point for using *ProbUI's* probabilities in other frameworks, namely by extending/replacing our basic mediator class. We describe this mediator next.

Tracking candidates: As touch events arrive, *ProbUI* updates 1) probabilities on layers two and three, and 2) our mediator's set of candidates. As long as a GUI target is more likely than a minimum threshold, it holds candidate status. Candidates can use their behaviour probabilities (layer two) and activation probability (layer three) to update feedback, like changing transparency, live result previews, and so on.

Activating candidates: Widgets can request "determination". A simple button would do so on touch up. Our mediator determines the candidate with the highest probability (Eq. 6), or the only one with a request. Alternatively, we can choose all above a certain threshold to enable multi-selection (e.g. [37]). All non-determined candidates are then "excluded". Developers can use 1) their widget's `onDetermined()` method to perform lasting changes (e.g. a simple button would trigger its associated action), and 2) their widget's `onExclude()` method for clean-up (e.g. removing visual feedback).

MODEL DISCUSSION AND LIMITATIONS

Search and simplicity: We apply the Viterbi algorithm to search for most likely state sequences (see Figure 2). It has complexity $\mathcal{O}(S^2T)$ [43] for S states and T touch events. Extending our approach to highly complex gestures with many states and long touch sequences may require other search methods, yet such complex gestures might not be a suitable choice for mobile touch GUIs regarding usability. In this work, we are interested in gestures tied to GUI targets, which are relatively simple and short, as our examples show, including those from the literature. Thus, PML (without rules) currently only supports direct specification of linear state sequences. This could be extended in future work, for example by adding an alternation operator ("|").

Utility: The framework currently offers no integration of utility of actions. Future work could provide a way to assign utility to outcomes, possibly via a utility parameter for callbacks.

Discrete time focus: By fixing the touch event rate, *Proton++* allows developers to repeat symbols to extend gesture duration in fixed steps [29]. In PML, we can specify (min/-max) gesture durations. Both methods are clearly limited, indicating that timing is more difficult to describe with fixed symbols than space. Deriving probabilistic models from such mainly spatial declarations thus leads to a lack of temporal continuum (our HMMs place distributions in *space*). However, most gestures have discrete points of interest in space (e.g. start/end, turns, repetitions) at which progress can be assessed (**$** marker, Table 1). Moreover, live feedback for GUI elements – like highlighting (potential) targets – is usually related to finger location or distance from targets, not speed.

Distributions: We limited the model to 2D Gaussians. They are a suitable choice for modelling touch/target locations [17, 18, 19, 51, 56], and enable the direct boxes-behaviours analogy with our mapping from declarations to probabilistic models. However, other distributions – or further dimensions – might be useful, for example, to consider features which we currently excluded from the probabilistic model, although they can still already be used in declarations (e.g. pressure).

Overall, limitations arise since we use simple models to 1) be able to automatically derive them from declarations, and to 2) keep the analogy to bounding boxes, with which developers are already familiar. Ease of use of declarations may not always be needed, for example when developers are deeply involved with probabilistic modelling themselves. In such cases, we recommend to consider general probabilistic programming languages like *Infer.NET* [36]. While more powerful, they likely require extra initial development effort compared to a more specific framework like *ProbUI*, which comes directly integrated into an existing GUI toolkit (Android).

APPLICATION EXAMPLES

Considering the fundamental role of proof-of-concept implementations [25] and examples [16, 30, 34, 35, 48, 49, 50], we show *ProbUI*'s value with 1) implementing behaviours from the literature, and 2) new widgets (also see video[5]).

Examples I: Realising Behaviours from Related Work

Table 3 shows BBs for well-known touch interactions. We next implement selected behaviours from related work.

Expression – Behaviour, Rule(s)	Explanation and Example
`Cd*u`	lift finger on widget [42]
`Cdu`	tap on widget
with rule: `on complete with large a`	tap with large contact area, e.g. *FatThumb* [11]
with rule: `on complete in >600ms`	tap with long press
`Cdudu`	double tap on widget
`Ld->Ru`	slide left to right on widget; e.g. slide-to-unlock
`L->R`	slide left to right over widget
`L->C, R->C` with **and** rule	pinch gesture on widget (requires two fingers)
`C->L, C->R` with **and** rule	zoom gesture on widget (requires two fingers)
`Cd->Eu`	slide to right out of widget; e.g. drag to right [37]
`C->E`	slide to right over widget; e.g. bezel swipe [44]
`N->C->S`	vertically crossing the widget [2, 40]
`N<->E<->S<->W<->N`	encircling the widget [14, 27]
`N->E->S->W->N`	encircling the widget clockwise

Table 3. Examples of bounding behaviours defined in PML.

[5] http://www.medien.ifi.lmu.de/probui/video

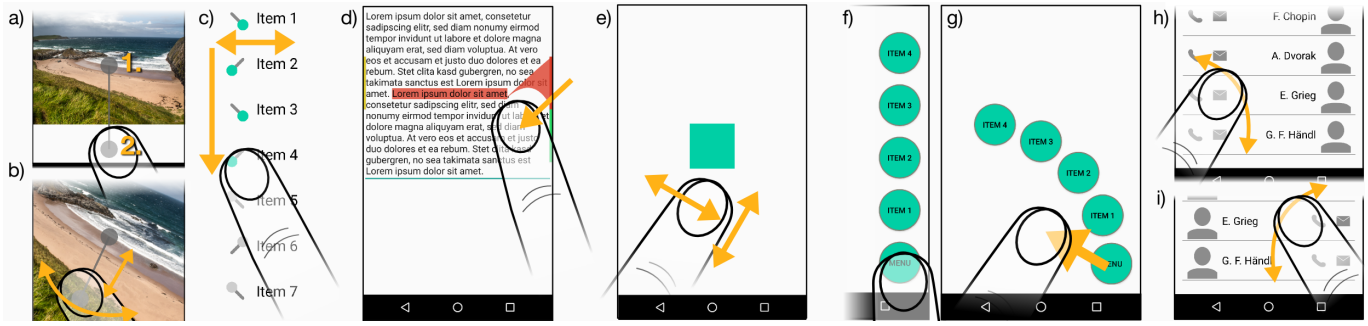

Figure 5. Screenshots of implemented example widgets (fingers/arrows added): *(a, b)* This image viewer brings up one-finger zoom/rotate controls (grey) on a vertical *Ta-tap* [21]. *(c)* Multiple toggles [37] are activated in one downwards stroke. Their alternating orientation also enables unambiguous single selection via left/right slides. *(d)* Bezel swipe [44] uses thin edge buttons for selection/cut/paste. *(e)* Two rubbing directions enable one finger zoom in/out [39]. *(f, g)* This floating menu can be opened straight on tap or curved with a short drag to its left, fitting a left thumb's reachable area. *(h, i)* This contact list adapts its alignment based on left/right thumb scrolling, so that the call/mail buttons are close to the thumb and text is not occluded.

ThumbRock [10]

These short back-and-forth thumb rolls are detected via the touch point's vertical shift, for example to implement switches [10]. In *ProbUI*, we implement this model for a switch in one line (**`Td->B->Tu`**) – a touch appears near the widget's top, shifts to its bottom, then back up. If we need to distinguish this from sliding in the same way, we add a rule for a large touch area (e.g. **`with large a`**, Table 2), since *ThumbRock* leads to a larger area than slides [10]. In code:

```
addBehaviour("thumbRock: Td->B->Tu");
addRule("thumbRock on complete with large a",
   new RuleListener() { public void onSatisfied() {
      respondToThumbRock(); }});
```

Consecutive Distant Taps (Ta-tap) [21]

Ta-tap is a double tap with a "jump". It can enable one-finger zoom/rotation [21]. We implemented it in an image widget (Figure 5a, b) alongside the usual two-finger zoom/rotation:

```
addBehaviour("taTap: Cdu->Bd");
addBehaviour("move: Cdm");
addBehaviour("doubleTap: Cdudu");
addRule("taTap on complete in <500 ms",
   new RuleListener() { public void onSatisfied() {
         switchModeToOneFingerZoomRotate(); }});
addRule("move on complete using 2 fingers",
   new RuleListener() { public void onSatisfied() {
         switchModeToTwoFingerZoomRotate(); }});
addRule("move on complete using 1 finger",
   new RuleListener() { public void onSatisfied() {
         switchModeToPanning(); }});
addRule("doubleTap on complete in <500 ms",
   new RuleListener() { public void onSatisfied() {
         resetViewTransformation(); }});
```

In words, tapping near the centre, then quickly pressing down at the bottom (**`Cdu->Bd`**) brings up one-finger zoom/rotate controls [21]. Starting a drag (**`Cdm`**) leads to panning (one finger), or pinch-zoom/rotate (two fingers). To help implement such functionality, *ProbUI* provides methods to compute distance/angle between fingers, which we then use in Android's canvas transforms. A double tap (**`Cdudu`**) resets the view.

Sliding Toggle Switches [37]

Flipped via matching slides, alternating these switches' orientations in a list 1) enables flipping multiple ones in one stroke, and 2) avoids ambiguity when selecting just one (Figure 5c) [37]. We implement this with BBs for sliding down (**`N->S`**), up (**`S->N`**), left (**`E->W`**), and right (**`W->E`**), and these rules: 1) down-right oriented switches turn on if "down or right is complete and most likely" (see tokens in Table 2: the rule can be written like this sentence); 2) down-left switches turn on if "down or left is complete and most likely"; 3) switches turn off via up, or left and right slides, respectively.

Bezel Swipe [44]

Bezel swipe avoids conflicts of scrolling and selection (e.g. in text views) with very thin "buttons" along the screen edge [44]. Swiping from the device edge onto the screen through such a button activates its mode (e.g. select, cut, or paste) until touch up. We implement this in *ProbUI* by adding left/right swipes (**`C->E`**, **`C->W`**) to thin edge-aligned buttons (Figure 5d). Since their starting state (**`C`**) is hard to hit by direct touch, users instead have to press down on the device edge, then swipe onto the screen, thus realising bezel swipes.

Rub to Zoom [39]

Olwal et al. [39] improved target selection by allowing users to first zoom in via rubbing, then moving the finger onto their (now enlarged) target. They distinguished two rubbing directions, which we implement as **`O<->NE`** and **`O<->NW`**. Since users may rub anywhere on zoomable content (Figure 5e), we use the origin token (**`O`**). It indicates that the areas should be interpreted relative to the gesture's start. Instead of reflecting the size of the widget, the states then relate to a default square area, roughly the width of a finger. Developers may change this (e.g. **`O[w=100dp,h=60dp]`**).

Examples II: Novel Widgets for One-handed Use

Bending Sliders

Our novel slider uses five BBs to bend itself to stay within the thumb's reach [5], see Figure 6. We define BBs as shown:

```
addBehaviour("onSlider: C");
addBehaviour("leftDown: SL<->SSY");
addBehaviour("rightDown: SR<->SSY");
addBehaviour("leftUp: NL<->NNY");
addBehaviour("rightUp: NR<->NNY");
addRule("onSlider is most_likely", new RuleListener() {
   public void onSatisfied() { setNoBend(); }});
addRule("leftDown is most_likely", new RuleListener() {
   public void onSatisfied() { setBendLeftDown(); }});
addRule("rightDown is most_likely", new RuleListener() {
   public void onSatisfied() { setBendRightDown(); }});
addRule("leftUp is most_likely", new RuleListener() {
   public void onSatisfied() { setBendLeftUp(); }});
addRule("rightUp is most_likely", new RuleListener() {
   public void onSatisfied() { setBendRightUp(); }});
```

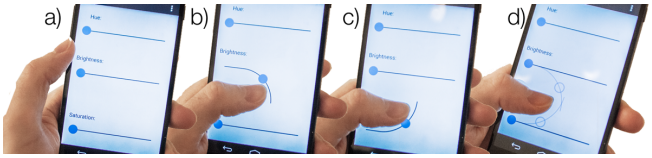

Figure 6. Multiple bounding behaviours let these sliders *(a)* adapt their visuals and transfer functions to the thumb *(b, c)*. The sliders' activation probabilities, visualised via transparency *(d)*, allow users to resolve ambiguity by moving the finger in line with the desired preview. For clarity, we only show sliders in this example. However, they also work next to other GUI widgets, overlapping them if needed when bending.

We further use this example to illustrate how *ProbUI's* probabilistic information helps to provide feedback. While a slider is a candidate for the user's intended target, we draw a preview with opacity indicating its activation probability (Eq. 6):

```
public void onDraw(Canvas canvas) { //Android draw method
   if (this.isCandidate()) {
      float previewOpacity = this.getCandidateProb();
      ... // draw preview
   }
}
```

Adaptive Menu

Google's Material Design[6] has a floating button, that may also open a vertical menu. The topmost items can be difficult to reach with the left thumb, when the menu is placed at the right (Figure 5f, g). Our example widget opens such a menu on tap (**`Cdu`**). However, a short "flick" from its left (**`Ld->NWu`**), fitting a left thumb's movement arc, opens the menu curved to the left, so that all items can be reached more easily.

Adaptive List

Lists (e.g. Android settings) and their entry labels (e.g. "Wi-fi") are often left-aligned; related buttons are placed to the right (e.g. Wi-fi on/off switch). There, they can be hard to reach with the left thumb, which may also occlude the labels while scrolling. A reverse alignment for left-hand use seems more suitable. Our widget (Figure 5h, i) uses three BBs for automatic adaptation: straight scrolling (**`T<->B`**), scrolling in a left arc (**`L<->B`**), and a right one (**`R<->B`**). As in the previous examples, callbacks update the alignment accordingly.

DEVELOPER EVALUATION

We evaluated *ProbUI* with developers in a survey and a study.

Online Survey: Understanding PML

We conducted a survey, similar to Proton's evaluation [29], to assess how easy it is for developers to understand PML.

Survey Design

The survey first explained the basics of PML, then presented three tasks with four questions each: 1) *Gesture to PML:* Given a gesture (as video and still image with annotated trajectory), choose the PML expression (out of four) that best matches the gesture. 2) *PML to gesture:* Given a PML expression and four trajectories, select the correct trajectory (Figure 7). 3) *Writing PML:* Express the shown trajectory in PML. Level of difficulty was informed by a short pre-study.

[6] `https://www.google.com/design/spec/material-design/`, *last accessed 21st December 2016*

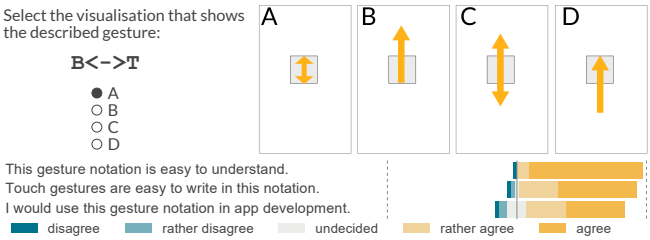

Figure 7. *Top:* An example question from the survey (with correct answer). *Bottom:* Results of the survey's Likert questions.

Participants

We recruited participants via social media, an Android forum, and a university IT mailing list. 33 developers with a mean age of 25 years (range: 19-35) completed the survey (11 female). They could win one of three € 20 gift cards; chance of winning increased with the number of correct answers.

Results and Discussion

On average, developers achieved a score of 95.45% correct answers in 8:06 minutes, including reading the explanation. This shows that developers can successfully and quickly learn PML's basics. In Likert questions (Figure 7), 97% positively rated PML as easy to understand and 91% as easy to write. The majority (76%) indicated interest in using it. Very few people rated it negatively; one commented that he would also like a relative notation. This is in fact possible with the "origin" token (**`0`**, Table 1), which did not appear in the survey.

Developer Workshop: Feedback on Using ProbUI

We conducted a workshop to gather feedback from developers based on hands-on experience with using *ProbUI*.

Design and Procedure

We invited developers to our lab, one at a time, for 90 minute sessions. We provided a printed reference, and scaffolding code for parts unrelated to the focus of the tasks to save participants' time. We introduced *ProbUI* in a presentation of about 20 minutes, including answering initial questions. Participants then coded six short "projects" (Table 4). We encouraged them to ask questions and "think aloud". We recorded audio for later analysis. At the end, they filled in a questionnaire on what they (dis)liked about the framework and study, and answered a set of Likert questions (Figure 9).

1. Hello World – A button that shows "Hello World!" on tap. *Focus:* Defining a touch behaviour and reacting to it with a rule and callback.

2. Quiz – Users can reconsider their answer after touch down by moving the finger. An answer's transparency is based on its selection probability. *Focus:* Using probabilities for continuous feedback.

3. Colour Mixer – A canvas with red/green/blue swatches in the corners. Colour is mixed based on the finger's proximity to each swatch. *Focus:* Accessing probabilities from "anywhere" for custom effects. Modifying touch areas ("touching in a wide area around a swatch"; e.g. **`C[s=8]`**).

4. Sliding Widgets – Sliding toggles [37] (Figure 5c). *Focus:* Using multiple gestures per widget and more complex rules with **`and`**, **`or`**.

5. Photo Gallery – A gallery with left/right swipes to switch through the photos. *Focus:* Adapting the GUI based on the most likely behaviour. Using the "origin" token (**`0`**) to enable starting swipes anywhere.

6. Photo Gallery Transitions – The gallery's previous/next photo fades in as the user is swiping. *Focus:* Continuous feedback based on the probabilities of multiple behaviours for one GUI element.

Table 4. The programming projects from the study.

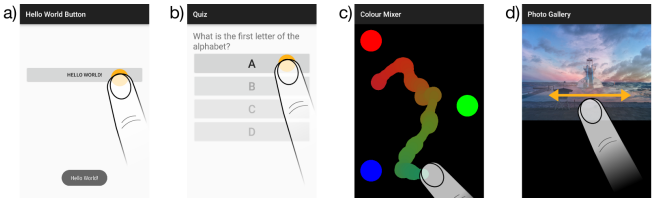

Figure 8. Screenshots of the study projects (fingers/arrows added): *(a)* "Hello World" button; *(b)* GUI for a quiz game; *(c)* continuous colour mixer / drawing app; *(d)* image gallery app with transition effects. Participants also implemented the sliding toggle widgets shown in Figure 5c.

Participants

We recruited 8 people (3 female, mean age: 25 years, range: 22-27) with Android development experience and related jobs via social media and mailing lists. They received € 10/h.

Results and Discussion

All participants could solve all tasks. However, one was short on time and thus had to skip the last project.

Concepts are positively accepted: Our framework received positive overall feedback, both in the Likert ratings (Figure 9) and people's comments. For example, developers liked that "it opens doors to many new possibilities that are otherwise not feasible" (P_1), that it "motivates to think of completely new ways of interaction" (P_3), and that it "saves lines of code" and is "more efficient than conventional GUI coding" (P_5). P_2 liked that she could "make design changes on the basis of probabilities". P_4 noted that he could still think in terms of bounding boxes, and stated that this helped him to learn and transition to the new framework. P_6 expressed a similar view, and P_7 found "that it allows for very nice applications".

PML is easy to use after introduction: Developers liked specifying gestures and callback-rules via PML, yet there was a learning curve: For example, the difference between the two sets of area tokens (on widget vs around it) was not always clear initially. In general, everyone said that they had to get used to the concepts at first, but later commented on ease of use. For instance, P_7 stated: "The framework was easy to use after getting familiar with the concept". Similarly, P_8 said: "It was pretty easy to use the framework, once the possibilities are known. The syntax was easy to understand and short in lines of code." P_1 said that "PML is really really easy and straight forward". Others liked "how easy it was to make and work with custom gestures" (P_2), and "that the method calls are very verbatim" (P_3). This feedback fits to the high scores in the survey on understanding PML.

Gesture expressions vary: For example, developers expressed a "slide right" for the sliding widgets as **`W->E`** or **`L->R`**. Both solutions work and people used their choice consistently for the other slides. Variations partly occurred since it was not clear whether the given widget was the whole slider or the knob. As P_6 said, "an introductory [...] visualisation could facilitate that process". We thus amended our documentation with visualisations for basic GUI elements. Moreover, we added a "debug mode" that shows both boxes as well as gesture models (see video for similar visualisations). This facilitates learning PML by allowing developers to visually verify their bounding behaviours directly in the running app.

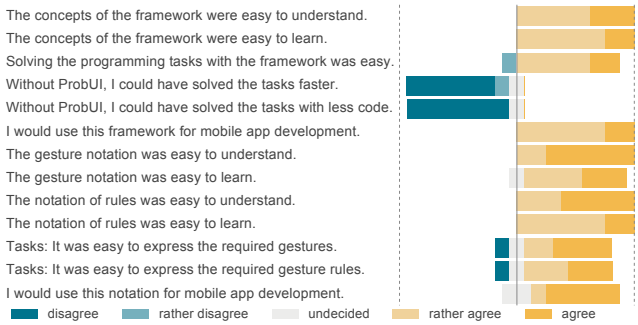

Figure 9. Likert questions and results from the study. Overall, developers rated our framework and concepts favourably. Comments revealed two main points of critique: 1) lack of IDE support for PML, and 2) initial learning curve for the API and the probabilistic concepts.

PML lacks IDE support: P_3, P_4 and P_8 noticed that a downside of PML is the lack of syntax checking and auto-completion. This is the reason for the few non-positive ratings on ease of use in Figure 9. Such features could be realised as an IDE plugin. As a quick fix, P_4 suggested that he could define string constants for PML parts to enable auto-completion.

Working with probabilities is unfamiliar but welcomed: Probabilistic GUIs were new to everyone. Several people asked about the update of probabilities and their scope. For example, P_4 asked whether the **`most_likely`** rule refers to behaviours of this element or the whole GUI. The most common initial hurdle (five people) was the difference between probabilities for gestures and for GUI elements. Promisingly, comments and observations also showed that throughout the study, developers grasped the practical value of these two types of probabilities. They accessed probabilities for behaviours (gallery, sliding widgets) and GUI elements (quiz, colour mixer). Four people even found a shorter way of using the probabilities for the colour mixer, which we then adopted as the reference solution. The probabilistic concept received positive final feedback, for example regarding GUI adaptation and feedback (P_2, P_5, P_6), and enabling novel applications and interactions (everyone). P_4 critically stated that "it would be great if any element could be enhanced with probability", suggesting games and touchscreen musical instruments in addition to our tasks. P_3 and P_7 noted that the probabilities inspired them to think about interaction from a new perspective.

Refining API and documentation: Based on comments by the first two participants, we revised the examples and introduction, which noticeably improved the learning curve. Following feedback by three developers (P_4, P_6, P_7), we clarified several method names. We also followed P_1's suggestion to remove a "setReady" method that used to be required after adding behaviours/rules. Based on feedback by P_3 and P_8, we modified the API to also return the behaviour-objects when adding behaviours. Developers can now assess gestures both by their labels or via those objects (e.g. to get probabilities). We also improved the "origin" token (**`0`**). Originally, it was merely a flag to indicate a relative gesture, but almost everyone used it as an area itself ("finger centre"), since that is more consistent to the rest of PML. Hence, we changed it to work that way (Table 1 already shows the *new* version).

SUMMARY AND DISCUSSION

As the examples and evaluation show, *ProbUI* helps developers to define, detect, and react to their widgets' required touch behaviours. Crucially, it automatically creates probabilistic models for these behaviours. Regarding widgets from the literature, the original work did not provide such models. Each project rather invented its own method for detecting the target behaviour and distinguishing it from other behaviours.

In contrast, *ProbUI* realises these widgets with one consistent model. This also greatly facilitates combining multiple such widgets in one probabilistic GUI. For example, we can easily add our adaptive menu on top of our adaptive list (see video).

In the following paragraphs, we address questions from discussions with fellow researchers:

Do touch GUIs get better if they use many gestures?
Not *every* button has to react to multiple gestures; developers still decide if/when to use gestures. Our one-hand widgets adapt to behaviour variations and constraints, rather than introducing completely new behaviours. *ProbUI* does not preclude the use of normal (Android) widgets: they either receive normal events or use a wrapper (which e.g. simply defines `C`).

Which gestures cannot be expressed in ProbUI?
As a declarative language, PML trades some expressiveness for ease of use when defining gestures for GUI elements. It is not as powerful as general regular expressions, but covers many GUI-related gestures from the literature. However, complex (multitouch) trajectories (e.g. as shortcuts unrelated to GUI targets) are better created via demonstration [33]. It is also currently not possible to directly express aspects like "not entering" a region and invariance to distance (e.g. encircling with any radius). Here, future work could explore other features, like angles instead of x, y. Conceptually, instead of writing PML, HMMs could also be learned from data.

How do I choose the best PML expression for my gesture?
Gestures for GUI elements can be expressed in PML directly as a chain of areas which the finger should pass, in analogy to bounding boxes. It is generally not important to find a "best" expression. Due to the probabilistic approach, PML's areas can be used rather robustly. For example, a "slide right" may work practically well for a widget regardless of whether it was defined as `L->R` or `W->E`.

How accurately do BB models fit users' actual behaviour?
HMMs created via PML cannot match behaviour as well as if they were learned from user behaviour data. However, PML enables developers to define probabilistic GUIs without collecting training data. Note that useful *relative* comparisons are indeed possible with our more "approximative" use of HMMs, as our examples demonstrate. These examples and the literature show that most GUIs and widgets are interested in such relative comparisons.

How does ProbUI's inference respect interface states?
While HMMs themselves do not use "GUI states", *ProbUI* indeed considers widgets' locations and sizes, since each HMM is based on its widget's location and size. Moreover, widgets' *relative* locations and sizes reflect in the target posterior (Eq. 6). Our mediator can also access GUI properties. For example, it ignores invisible or disabled widgets, but developers may write other mediators to consider such states differently. *ProbUI's* API does not directly include (nor require) defining widget state machines (e.g. [49]). Nevertheless, developers can use *ProbUI's* probabilities to inform probabilistic "state" changes for widgets (e.g. changing our slider's bending state, i.e. direction). Finally, bounding behaviours follow their widget, if it moves on the screen (e.g. respecting scrolling state).

Are bounding behaviours computationally expensive?
BBs require more computations per touch than checking boxes, but less than adaptive keyboards (e.g. [56]) with language models. We experienced no delays (Nexus 5). While costs increase with more targets and behaviours, their number in any GUI is inherently limited by usability. *ProbUI* can also be highly parallelised (e.g. one job per target and behaviour).

CONCLUSION

GUI toolkits today describe targets with static rectangles. This limits possibilities for feedback, GUI adaptations, dynamic interaction, and reasoning under uncertainty, as envisioned by probabilistic frameworks [26, 34, 35, 48, 49, 50]. By generalising target representations from static geometry tests to sets of gestures, *ProbUI* enables developers to create probabilistic GUIs that not only consider *where* targets expect touches, but also *how* input is performed over time.

ProbUI for the first time integrates 1) easy to use declarative gesture definition for GUIs with 2) probabilistic models and 3) reasoning in one framework. It achieves this with the first declarative gesture modelling language (PML) and interpreter which automatically yield probabilistic models for touch GUIs, given non-probabilistic definitions.

We demonstrated *ProbUI's* value by implementing examples from the literature and novel widgets. A survey showed that developers can learn the basics of PML in under ten minutes. We also gathered feedback in a lab study. Developers liked the ease of use of PML. Although working with probabilities was unfamiliar to them at first, they successfully completed the tasks and found *ProbUI's* probabilities useful. They identified areas for further improvement, which we partly implemented directly via refinements to both API and documentation.

ProbUI can be used by developers and researchers to implement and prototype novel widgets, adaptations, and feedback. Independent of *ProbUI*, our generalisation from boxes to "behaviours" offers a new perspective on GUI target representation. In general, we hope to spark more widespread use of probabilistic reasoning, feedback and adaptation in the design and development of future mobile touch GUIs.

Bounding behaviours are not limited to touch: Future work could support further modalities, for example to represent GUI elements that react to shaking the device.

ProbUI for Android, a detailed documentation, and a tutorial course based on the improved workshop material are available on the project's website:

`http://www.medien.ifi.lmu.de/probui/`

REFERENCES

1. Derek Anderson, Craig Bailey, and Marjorie Skubic. 2004. Hidden Markov Model Symbol Recognition for Sketch-Based Interfaces. In *AAAI Fall Symposium*. 15–21. https://www.aaai.org/Papers/Symposia/Fall/2004/FS-04-06/FS04-06-003.pdf

2. Georg Apitz, François Guimbretière, and Shumin Zhai. 2008. Foundations for Designing and Evaluating User Interfaces Based on the Crossing Paradigm. *ACM Trans. Comput.-Hum. Interact.* 17, 2, Article 9 (May 2008), 42 pages. DOI:http://dx.doi.org/10.1145/1746259.1746263

3. Tyler Baldwin and Joyce Chai. 2012. Towards Online Adaptation and Personalization of Key-target Resizing for Mobile Devices. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces (IUI '12)*. ACM, New York, NY, USA, 11–20. DOI: http://dx.doi.org/10.1145/2166966.2166969

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

5. Joanna Bergstrom-Lehtovirta and Antti Oulasvirta. 2014. Modeling the Functional Area of the Thumb on Mobile Touchscreen Surfaces. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1991–2000. DOI: http://dx.doi.org/10.1145/2556288.2557354

6. Joanna Bergstrom-Lehtovirta, Antti Oulasvirta, and Stephen Brewster. 2011. The Effects of Walking Speed on Target Acquisition on a Touchscreen Interface. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11)*. ACM, New York, NY, USA, 143–146. DOI: http://dx.doi.org/10.1145/2037373.2037396

7. Frédéric Bevilacqua, Bruno Zamborlin, Anthony Sypniewski, Norbert Schnell, Fabrice Guédy, and Nicolas Rasamimanana. 2010. Continuous Realtime Gesture Following and Recognition. In *Proceedings of the 8th International Conference on Gesture in Embodied Communication and Human-Computer Interaction (GW '09)*. Springer-Verlag, Berlin, Heidelberg, 73–84. DOI: http://dx.doi.org/10.1007/978-3-642-12553-9_7

8. Xiaojun Bi, Yang Li, and Shumin Zhai. 2013. FFitts Law: Modeling Finger Touch with Fitts' Law. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 1363–1372. DOI: http://dx.doi.org/10.1145/2470654.2466180

9. Xiaojun Bi and Shumin Zhai. 2013. Bayesian Touch: A Statistical Criterion of Target Selection with Finger Touch. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 51–60. DOI: http://dx.doi.org/10.1145/2501988.2502058

10. David Bonnet, Caroline Appert, and Michel Beaudouin-Lafon. 2013. Extending the Vocabulary of Touch Events with ThumbRock. In *Proceedings of Graphics Interface 2013 (GI '13)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 221–228. http://dl.acm.org/citation.cfm?id=2532129.2532166

11. Sebastian Boring, David Ledo, Xiang 'Anthony' Chen, Nicolai Marquardt, Anthony Tang, and Saul Greenberg. 2012. The Fat Thumb: Using the Thumb's Contact Size for Single-handed Mobile Interaction. In *Proceedings of the 14th International Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI '12)*. ACM, New York, NY, USA, 39–48. DOI:http://dx.doi.org/10.1145/2371574.2371582

12. Daniel Buschek, Simon Rogers, and Roderick Murray-Smith. 2013. User-specific Touch Models in a Cross-device Context. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI '13)*. ACM, New York, NY, USA, 382–391. DOI:http://dx.doi.org/10.1145/2493190.2493206

13. Baptiste Caramiaux, Nicola Montecchio, Atau Tanaka, and Frédéric Bevilacqua. 2014. Adaptive Gesture Recognition with Variation Estimation for Interactive Systems. *ACM Trans. Interact. Intell. Syst.* 4, 4, Article 18 (Dec. 2014), 34 pages. DOI: http://dx.doi.org/10.1145/2643204

14. Eun Kyoung Choe, Kristen Shinohara, Parmit K. Chilana, Morgan Dixon, and Jacob O. Wobbrock. 2009. Exploring the Design of Accessible Goal Crossing Desktop Widgets. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems (CHI EA '09)*. ACM, New York, NY, USA, 3733–3738. DOI: http://dx.doi.org/10.1145/1520340.1520563

15. Alessandro De Nardi. 2008. *Grafiti: Gesture recognition management framework for interactive tabletop interfaces*. Master's thesis. University of Pisa.

16. Anind.K. Dey, Jennifer Mankoff, and Gregory D. Abowd. 2000. *Distributed mediation of imperfectly sensed context in aware environments*. Technical Report. Georgia Institute of Technology. http://hdl.handle.net/1853/3424

17. Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. 2002. Language Modeling for Soft Keyboards. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI '02)*. ACM, New York, NY, USA, 194–195. DOI: http://dx.doi.org/10.1145/502716.502753

18. Tovi Grossman and Ravin Balakrishnan. 2005. A Probabilistic Approach to Modeling Two-dimensional Pointing. *ACM Trans. Comput.-Hum. Interact.* 12, 3 (Sept. 2005), 435–459. DOI: http://dx.doi.org/10.1145/1096737.1096741

19. Asela Gunawardana, Tim Paek, and Christopher Meek. 2010. Usability Guided Key-target Resizing for Soft Keyboards. In *Proceedings of the 15th International Conference on Intelligent User Interfaces (IUI '10)*. ACM, New York, NY, USA, 111–118. DOI: http://dx.doi.org/10.1145/1719970.1719986

20. Niels Henze, Enrico Rukzio, and Susanne Boll. 2011. 100,000,000 Taps: Analysis and Improvement of Touch Performance in the Large. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11)*. ACM, New York, NY, USA, 133–142. DOI:http://dx.doi.org/10.1145/2037373.2037395

21. Seongkook Heo, Jiseong Gu, and Geehyuk Lee. 2014. Expanding Touch Input Vocabulary by Using Consecutive Distant Taps. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 2597–2606. DOI:http://dx.doi.org/10.1145/2556288.2557234

22. Ken Hinckley and Daniel Wigdor. 2012. Input Technologies and Techniques. In *The Human-Computer Interaction Handbook – Fundamentals, Evolving Technologies and Emerging Applications* (third edit ed.), Julie A. Jacko (Ed.). CRC Press, Chapter 9, 151–168.

23. Christian Holz and Patrick Baudisch. 2010. The Generalized Perceived Input Point Model and How to Double Touch Accuracy by Extracting Fingerprints. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 581–590. DOI: http://dx.doi.org/10.1145/1753326.1753413

24. Christian Holz and Patrick Baudisch. 2011. Understanding Touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2501–2510. DOI:http://dx.doi.org/10.1145/1978942.1979308

25. Scott E. Hudson and Jennifer Mankoff. 2014. Concepts, Values, and Methods for Technical Human-Computer Interaction Research. In *Ways of Knowing in HCI*, Judith S. Olson and Wendy A. Kellogg (Eds.). Springer New York, 69–93.

26. Scott E. Hudson and Gary L. Newell. 1992. Probabilistic State Machines: Dialog Management for Inputs with Uncertainty. In *Proceedings of the 5th Annual ACM Symposium on User Interface Software and Technology (UIST '92)*. ACM, New York, NY, USA, 199–208. DOI: http://dx.doi.org/10.1145/142621.142650

27. Hyun W. Ka. 2013. *Circling Interface: An Alternative Interaction Method for On-Screen Object Manipulation*. Ph.D. Dissertation. University of Pittburgh. http://d-scholarship.pitt.edu/19305/

28. Shahedul Huq Khandkar and Frank Maurer. 2010. A Domain Specific Language to Define Gestures for Multi-touch Applications. In *Proceedings of the 10th Workshop on Domain-Specific Modeling (DSM '10)*. ACM, New York, NY, USA, Article 2, 6 pages. DOI: http://dx.doi.org/10.1145/2060329.2060339

29. Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. 2012a. Proton++: A Customizable Declarative Multitouch Framework. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 477–486. DOI: http://dx.doi.org/10.1145/2380116.2380176

30. Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. 2012b. Proton: Multitouch Gestures As Regular Expressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2885–2894. DOI:http://dx.doi.org/10.1145/2207676.2208694

31. Yang Li, Hao Lu, and Haimo Zhang. 2014. Optimistic Programming of Touch Interaction. *ACM Trans. Comput.-Hum. Interact.* 21, 4, Article 24 (Aug. 2014), 24 pages. DOI:http://dx.doi.org/10.1145/2631914

32. Hao Lü and Yang Li. 2012. Gesture Coder: A Tool for Programming Multi-touch Gestures by Demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2875–2884. DOI: http://dx.doi.org/10.1145/2207676.2208693

33. Hao Lü and Yang Li. 2013. Gesture Studio: Authoring Multi-touch Interactions Through Demonstration and Declaration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 257–266. DOI: http://dx.doi.org/10.1145/2470654.2470690

34. Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. 2000a. Interaction Techniques for Ambiguity Resolution in Recognition-based Interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. ACM, New York, NY, USA, 11–20. DOI: http://dx.doi.org/10.1145/354401.354407

35. Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. 2000b. Providing Integrated Toolkit-level Support for Ambiguity in Recognition-based Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00)*. ACM, New York, NY, USA, 368–375. DOI: http://dx.doi.org/10.1145/332040.332459

36. T. Minka, J.M. Winn, J.P. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. 2014. Infer.NET 2.6. (2014). http://research.microsoft.com/infernet Microsoft Research Cambridge.

37. Tomer Moscovich. 2009. Contact Area Interaction with Sliding Widgets. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, New York, NY, USA, 13–22. DOI:http://dx.doi.org/10.1145/1622176.1622181

38. Alexander Ng, Stephen A. Brewster, and John H. Williamson. 2014. Investigating the Effects of Encumbrance on One- and Two- Handed Interactions with Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1981–1990. DOI:http://dx.doi.org/10.1145/2556288.2557312

39. Alex Olwal, Steven Feiner, and Susanna Heyman. 2008. Rubbing and Tapping for Precise and Rapid Selection on Touch-screen Displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 295–304. DOI: http://dx.doi.org/10.1145/1357054.1357105

40. Charles Perin, Pierre Dragicevic, and Jean-Daniel Fekete. 2015. Crossets: Manipulating Multiple Sliders by Crossing. In *Proceedings of the 41st Graphics Interface Conference (GI '15)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 233–240. http://dl.acm.org/citation.cfm?id=2788890.2788931

41. Henning Pohl and Roderick Murray-Smith. 2013. Focused and Casual Interactions: Allowing Users to Vary Their Level of Engagement. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2223–2232. DOI: http://dx.doi.org/10.1145/2470654.2481307

42. R. L. Potter, L. J. Weldon, and B. Shneiderman. 1988. Improving the Accuracy of Touch Screens: An Experimental Evaluation of Three Strategies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '88)*. ACM, New York, NY, USA, 27–32. DOI: http://dx.doi.org/10.1145/57167.57171

43. L. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (Feb 1989), 257–286.

44. Volker Roth and Thea Turner. 2009. Bezel Swipe: Conflict-free Scrolling and Multiple Selection on Mobile Touch Screen Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1523–1526. DOI:http://dx.doi.org/10.1145/1518701.1518933

45. Anne Roudaut, Eric Lecolinet, and Yves Guiard. 2009. MicroRolls: Expanding Touch-screen Input Vocabulary by Distinguishing Rolls vs. Slides of the Thumb. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 927–936. DOI: http://dx.doi.org/10.1145/1518701.1518843

46. Christophe Scholliers, Lode Hoste, Beat Signer, and Wolfgang De Meuter. 2011. Midas: A Declarative Multi-touch Interaction Framework. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '11)*. ACM, New York, NY, USA, 49–56. DOI: http://dx.doi.org/10.1145/1935701.1935712

47. Julia Schwarz. 2010. Towards a Unified Framework for Modeling, Dispatching, and Interpreting Uncertain Input. In *Adjunct Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 367–370. DOI: http://dx.doi.org/10.1145/1866218.1866225

48. Julia Schwarz, Scott Hudson, Jennifer Mankoff, and Andrew D. Wilson. 2010. A Framework for Robust and Flexible Handling of Inputs with Uncertainty. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 47–56. DOI: http://dx.doi.org/10.1145/1866029.1866039

49. Julia Schwarz, Jennifer Mankoff, and Scott Hudson. 2011. Monte Carlo Methods for Managing Interactive State, Action and Feedback Under Uncertainty. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 235–244. DOI: http://dx.doi.org/10.1145/2047196.2047227

50. Julia Schwarz, Jennifer Mankoff, and Scott E. Hudson. 2015. An Architecture for Generating Interactive Feedback in Probabilistic User Interfaces. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 2545–2554. DOI: http://dx.doi.org/10.1145/2702123.2702228

51. Feng Wang and Xiangshi Ren. 2009. Empirical Evaluation for Finger Input Properties in Multi-touch Interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1063–1072. DOI: http://dx.doi.org/10.1145/1518701.1518864

52. Daryl Weir, Daniel Buschek, and Simon Rogers. 2013. Sparse Selection of Training Data for Touch Correction Systems. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI '13)*. ACM, New York, NY, USA, 404–407. DOI: http://dx.doi.org/10.1145/2493190.2493241

53. Daryl Weir, Simon Rogers, Roderick Murray-Smith, and Markus Löchtefeld. 2012. A User-specific Machine Learning Approach for Improving Touch Accuracy on Mobile Devices. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 465–476. DOI: http://dx.doi.org/10.1145/2380116.2380175

54. John Williamson. 2006. *Continuous Uncertain Interaction*. Ph.D. Dissertation. University of Glasgow.

55. Koji Yatani, Kurt Partridge, Marshall Bern, and Mark W. Newman. 2008. Escape: A Target Selection Technique Using Visually-cued Gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing*

Systems (CHI '08). ACM, New York, NY, USA, 285–294. DOI: http://dx.doi.org/10.1145/1357054.1357104

56. 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. DOI: http://dx.doi.org/10.1145/2470654.2481384

57. Shumin Zhai, Jing Kong, and Xiangshi Ren. 2004. Speed-accuracy tradeoff in Fitts' law tasks - On the equivalency of actual and nominal pointing precision. *International Journal of Human-Computer Studies* 61, 6 (2004), 823–856. DOI: http://dx.doi.org/10.1016/j.ijhcs.2004.09.007