An Architecture for Generating Interactive Feedback in Probabilistic User Interfaces

Julia Schwarz, Jennifer Mankoff, Scott E. Hudson
Human-Computer Interaction Institute
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213
{julia.schwarz, jmankoff, scott.hudson}@cs.cmu.edu

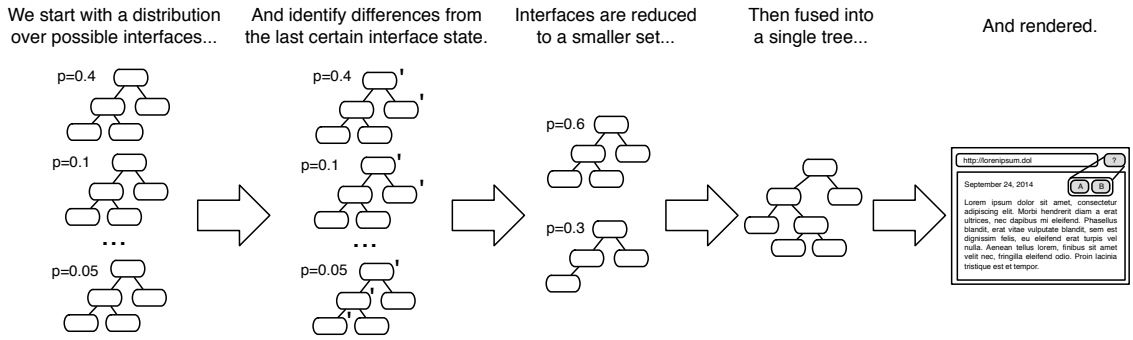

Figure 1. Architectural overview. The input is a set of interface samples, which are reduced to a smaller set of interfaces, then fused into a single interface, and rendered.

ABSTRACT

Increasingly natural, sensed, and touch-based input is being integrated into devices. Along the way, both custom and more general solutions have been developed for dealing with the uncertainty that is associated with these forms of input. However, it is difficult to provide dynamic, flexible, and continuous feedback about uncertainty using traditional interactive infrastructure. Our contribution is a general architecture with the goal of providing support for continual feedback about uncertainty.

Our architecture is based on prior work in modeling uncertainty using Monte Carlo sampling, and tracks multiple interfaces – one for each plausible and differentiable sequence of input that the user may have intended. Importantly, it considers how the presentation of uncertainty can be organized and implemented in a general way. Our primary contribution is a method for *reducing* the number of alternative interfaces and *fusing* possible interfaces into a single interface that both communicates uncertainty and allows for disambiguation. We demonstrate the value of this result through a collection of 11 new and existing feedback techniques along with two applications demonstrating the use of the feedback architecture.

CHI 2015, April 18 - 23 2015, Seoul, Republic of Korea

http://dx.doi.org/10.1145/2702123.2702228

INTRODUCTION

Recognition-based input technologies such as speech, touch gesture, and in-air motion are becoming increasingly prevalent. Unfortunately, these technologies are inherently uncertain, which violates a core assumption of most modern interface systems: that input is unambiguous. Prior work has both demonstrated the benefit of treating inputs probabilistically [3, 8, 11, 21], and proposed systems for integrating uncertainty while maintaining an architecture similar to conventional dispatch systems [18]. However, a system is needed for communicating this uncertainty back to the user.

This paper contributes a general architecture for tracking interface alternatives arising from varying interpretations of input, *reducing* the number of alternative interfaces to a few representative examples and *fusing* those possible interfaces into a single interface that dynamically communicates uncertainty and allows for disambiguation.

The overall effect of our approach is that feedback about uncertainty is present throughout the interaction. For example, as the user drags her finger in Figure 2, the system continually renders feedback about possible actions (line drawing, resizing, and dragging). Only when the finger is raised must the system select among these actions. In many cases, thanks to the continual feedback, the user may directly manipulate the degree of uncertainty during the input action. Should the correct action be unclear, the user can also disambiguate using a standard approach such as an N-Best list.

We demonstrate the flexibility of our architecture with a collection of 11 new and existing interaction techniques along with two complete applications implemented with our architecture. Ranging from predictive menu systems to interactive diagram beautification, these demonstrations not

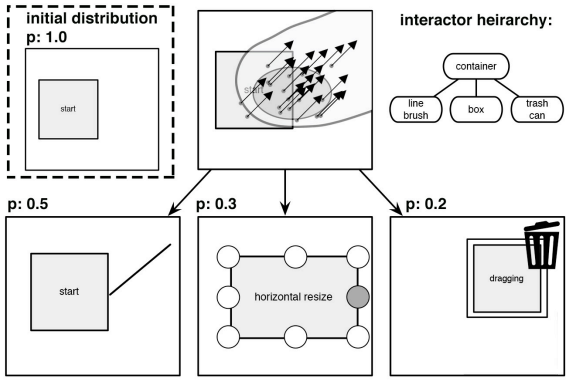

Figure 2. Example scenario. Initially the interface is certain. User presses and drags on a draggable/resizable box. Touch event samples are dispatched to possible interfaces. Three varieties of interfaces are possible: (left) the user is not dragging the box, but rather drawing a line, (center) the box is being resized, (right) the box is being dragged (a garbage can shows up, the user may drag to the trash to delete the box).

only show the generality of our architecture, but also the power of probabilistic approaches in user interfaces.

BACKGROUND

This work builds upon the contributions of prior research in three areas. First, the work builds on user interface frameworks for handling inputs with uncertainty. Additionally, many of the feedback techniques we describe are either inspired by or directly implement existing techniques for visualizing uncertain system state. A final area of related work explores interfaces that allow users to see the results of their actions in multiple parallel scenarios.

Frameworks for Handling Inputs With Uncertainty

A large body of work focuses on how to leverage probabilistic models to reduce ambiguity within specific input domains such as text entry [8, 21] touch targeting [3], speech [17], and gesture [11].

Moving beyond specific input technologies, there are at least three general approaches to building user interface systems for handling uncertain inputs. First, multimodal input systems provide disambiguation when multiple different input types (*e.g.,* speech and gesture) are combined [9]. For example, the TYCOON system [16] provided a fusion method that computed input likelihoods based on temporal proximity of fused events. Additionally, XWand [25] uses a Bayesian network to fuse multiple input sources, as well as a user's previous actions, to approximate a user's intent.

A second approach aims to manage uncertain input similarly to mainstream approaches for input handling. Schwarz *et al* [18] builds off of work by Hudson *et al* [6], and Mankoff *et al* [14,15] to provide an architecture for tracking the likelihood of interactive states given probabilistic inputs. These likelihoods are then used to decide on application action and provide feedback to the user. In the system provided by Schwarz, rendering of meaningful feedback is left to the developer: interactors need to consider the likelihood distribution over their state, as well as other interactors, to render themselves. As a result, only simple feedback techniques are demonstrated. Our work builds directly off of [18] by providing a general mechanism for identifying differences between alternate possible interfaces as well as a mechanism for fusing these alternatives so that interactor writers do not need to consider probabilistic state when rendering interactors.

A final approach presented by Williamson models the entire user interface as an uncertain process [23]. The input-action-feedback loop is used as a means to control a point p in an 'intention space', and the dynamics of the input system form a series of control loops that modify how sensed input affects p. Knowledge about the likelihood of actions enables modification of the dynamics of the input system, as demonstrated in [24]. Like our work, Williamson's system provides a way to synthesize feedback. However, the system design is quite different from the structure of conventional user interfaces. Our architecture aims to provide a developer-facing API that is as similar as possible to conventional interfaces, reducing the amount of developer effort needed for adoption.

Existing Techniques for Communicating Uncertainty

Communicating uncertainty in user interfaces is a tricky problem. When user input is ambiguous, users must understand not only that their input is ambiguous, but also how to provide disambiguation. To be most effective, the disambiguation interface must be a continuous interaction stream: a rapid-fire conversation between human and computer.

Examples of such fluid feedback mechanisms exist. In "Predictive Uncertain Displays" [23], Williamson uses inference to continuously update a visual display of a user's interpreted input. A similar approach is provided in Octopocus [1] where the interface continually updates the likelihood of possible gestures and renders possible completions to the user. In text entry and speech recognition systems, a list of alternate interpretations is often presented to the user; an early example is the ViaVoice system [2]. In many cases, how an input is being interpreted can be ambiguous. Wigdor [22] and Li [10] developed feedback mechanisms that show users how their touch inputs are being interpreted.

When the data primitives themselves (*e.g.,* the location of a mouse cursor, the width of an icon) are uncertain, prior work on visualization of uncertain information from the data visualization community may be used to communicate uncertainty (see for example [13]). This early work shows how drawing primitives such as contour crispness, fill clarity, and blur can be used to communicate uncertainty.

Single Document Model and Subjunctive interfaces.

Modern user interfaces often maintain a single interface state accomplishing a single possible task. Therefore, to consider, *e.g.,* alternate floor plans or different design

choices, users must undo and redo their actions. This is called the single state document model [19], and makes working with alternate scenarios laborious.

An alternative approach is to provide mechanisms for exploring multiple scenarios. In 2008, Lunzer coined the term *Subjunctive Interfaces* to describe interfaces that support the exploration of multiple scenarios [12]. For example, Side Views [20] is an interesting example of an early feedback technique that gave previews of the results of different choices a user may make (*e.g.,* making text bold *vs.* italic). Feedback about possible actions gets displayed in an interface as possible outcomes. In a similar vein Parallel Paths [19] presents a model of interaction that facilitates the generation, manipulation, and comparison of alternate solutions. These lenses into other 'possible worlds' are reminiscent of Magic Lenses [4], though that work predates Lunzer. Additionally, Igarashi's work on interactive beautification [7] shows possible drawings (*e.g.,* different snap targets, alternate interpretations) in a ghosted or dotted form, which can then be selected.

The body of work in subjunctive interfaces is especially interesting because it provides a rich set of interaction techniques for interacting with alternatives beyond even what is explored in this paper. This paper focuses on fusing and presenting multiple alternatives, one of which is eventually selected to disambiguate input. In contrast, both Parallel Paths and the RecipeSheet [12] provide facilities for manipulating alternatives interpretations directly. The ideas presented in subjunctive interfaces contain a wide range of possibilities for future work.

While the end results achieved by the methods described above look similar to many of the demos presented in this paper, the contribution of our work is different. These interfaces and interaction techniques are contributing just that: interface designs and interaction techniques. The contribution of this work is in an architecture that can enable simple construction of all of the interaction techniques presented, and more. Our architecture enables developers to easily experiment with different feedback techniques by allowing developers to easily switch between feedback methods, as well as develop new techniques. The task of analyzing interface differences, selecting alternatives, and fusing alternatives is abstracted away, allowing developers to focus on the interaction logic and presentation layer of the interface.

ARCHITECTURE DESCRIPTION

In conventional interfaces, the state of a single interface is the basis of everything that is displayed to the user. More specifically, if we know the structure of the interactor tree and the values of its associated variables, we know the state of an interface and have all of the information needed to render it on the screen. A necessary precondition for our feedback system is to replace that single interface with a distribution (a probability mass function or *PMF*) over possible interfaces. There are several ways to obtain a distribution over possible interfaces, and we structured our architecture similarly to the method described by Schwarz *et al* in [18].

In what follows, we describe both derivative work (which we compare to [18] in the text), and our feedback system, which is novel and the main contribution of this paper. We first describe the architecture we derived from [18] briefly. During event dispatch, this architecture tracks the probabilistic state of the interface and produces a probabilistic distribution of *action requests* representing the user's intent.

In a traditional non-probabilistic input system, the event dispatch process would have produced modifications to the interface as well as possibly the application. Typically, this would cause an interface to be drawn or redrawn based on the state and structure of the interactor hierarchy representing the interface. Essentially what a traditional system does is to take a tree representing the interface and render it on screen as an image. However, in our case there are multiple possibilities that need to be displayed, corresponding to possible intents of the user. We will describe our process for selecting representative examples from these multiple possibilities, combining them into a single tree, and rendering them on screen as feedback about uncertainty. A major contribution of this work is a flexible and interactive approach to doing this.

The toolkit described here and in the next sections was implemented in JavaScript and tested on the Google Chrome browser v37.0.2062.94. The toolkit uses a custom input dispatch pipeline, implemented by hooking into the normal input events available in the browser. The rendering system used in the toolkit uses SVGs, meaning that an interface draws itself onto an SVG element on an HTML page, which is then rendered by the browser. The system was developed and tested on a quad-core machine (2.3GHz each, Intel Core i7) with 16 GB of RAM. This consumer-grade machine was able to run all the demos described in this paper with reasonable responsiveness, with a maximum input dispatch time of 300 ms.

An Architecture for Tracking the User's Intent

To understand how these parts work together in the architecture consider the following setup (Figure 2): a user presses and drags with her finger on the edge of a draggable and resizable box (Figure 2, top middle). The interface also contains a 'line brush' which allows a user to draw straight lines. Initially the interface contains no ambiguity (Figure 2, top left), with the box in its neutral start state.

The implementation used in this paper takes a *Monte Carlo* approach, modeling user input as a collection of *event samples* each representing different inputs the user may have intended [18]. Note that we make use of weighted samples, where a sample's weight is an indication of its *likelihood* (an approximation of the probability that it is correct). For example, when the user presses her finger down, a collec-

tion of event samples representing an approximation of the distribution over possible locations is generated.

Each event sample is dispatched to the (probabilistic) interface to accumulate a new set of interface samples (probabilistic state for the overall interface) as well as a probability distribution over the actions that this would imply. To be more specific, we have one or more sample *interfaces* to which each sample *event* is dispatched (delivered to the appropriate interactor or interactors within the interface). We assume that the initial state of the interface is known.

For each pair of interface sample and event sample, we create a new interface sample that will process the event sample, and make requests for changes to the interface or application. Note that clones preserve key properties of interfaces such as a unique ID for interactors and interactor property values. Figure 2, bottom shows three possible samples that may be generated.

Tracking sample interface alternatives greatly simplifies dispatch, since when a sample event is dispatched to a sample interface, that sample interface is able to operate in an apparently deterministic world (the one indicated by the sample(s) involved). Note that this is different from [18], which tracks samples over *interactor state* (separately for each interactor) within a *single* interface.

When the dispatch of an input event sample to an interface sample causes an action (such as a visual change in the location or size of the rectangle in Figure 2, or a change to the application state such as the deletion of the box), an *action request* is created which encapsulates the action [18]. Action requests may be *local* (to the interface itself) or may encapsulate potentially irreversible *final* actions (*e.g.*, changing the state of an application outside the interface itself) [18]. As with other Monte Carlo representations of distributions, each action request becomes a *sample* within that distribution and has an associated likelihood. Thus, the collection of all actions requests is a representation of the probability distribution over possible actions implied by the possible input events to the possible states of the interface.

In the example in Figure 2, as the user drags her finger, a set of interface samples is tracked that produce action requests for all of the possible actions the user intends. These potentially include multiple action requests representing different start and end points for drawing a line, resizing the box, and moving (dragging) the box across the screen.

Going from Interface Samples to Feedback

At this point, in a non-probabilistic input system, any side effects of events have been executed and the tree representing the interface would be rendered on screen.

Instead, our feedback system goes through a four-stage process (Figure 1). First, it is necessary to *select* what actions to execute (and execute them). This step, for example, may eliminate highly unlikely options. Second, it is important to *reduce* the probability distribution to a manageable number of exemplars for display to the user. Presenting feedback about all possible interfaces to the user could in many cases be confusing. Third, we *fuse* the exemplars into a single tree according to the feedback technique being used. Finally, this tree must be rendered on screen to reflect any uncertainty to the user. Rendering is done in a conventional way and does not require any special infrastructure.

Before describing this process in detail, it is important to understand how it fits into the overall event dispatch and redraw cycle of the system. Each time a new set of sample events produces a PMF over alternative interfaces, a new fused interface is displayed on screen. In other words, the fused interface exists only for the instant in time between one user action and the next. For example, while the finger is dragging across the screen in our running example, the user essentially is using a direct manipulation interface in which they can see and influence the impact of their actions as they act.

Note that when an input event arrives, a feedback wrapper (present only in the fused interface) may elect to handle the input event and react to it rather than sending it to the probabilistic input system. For example, if a user selects an item in an N-Best list by pressing with her finger, the interactor generated by the N-Best fusion technique (an *N-Best Item Container*) will set that alternative interface as certain.

A major contribution of our work is our flexible approach to *selection*, *reduction* and *fusion*. Figure 1 shows the process for reduction and fusion. Each stage of the process is pluggable in our system.

Selecting Among Alternative Actions

Once dispatch is complete for a given input event, a collection of *action requests* has been generated which can be programmatically examined. The selection phase can deal with situations when an action is obviously wrong or right (*i.e.* has a very high or low likelihood). We provide support for automatic mediation (following the methods described in [18] and [14]).

Recall that actions are produced by interface samples, and represent a certain world view (with respect to that sample). Thus, to *reject* an action, we discard the action request and its associated interface state. This is the approach used when removing things that have very low likelihood.

Handling of accepted actions depends on the nature of the action: When an action request is *local*, executing it will make necessary updates to its associated interface without affecting the application. Since this type of request does not modify the application, its effects can be undone simply by discarding the interface sample it acted upon. As a result, we allow multiple local actions to be accepted, meaning they are executed on their associated interface alternative. All changed interactors in the interactor tree are marked as *changed* during this process (including changes to the properties of any interactor or number of children). Figure 3

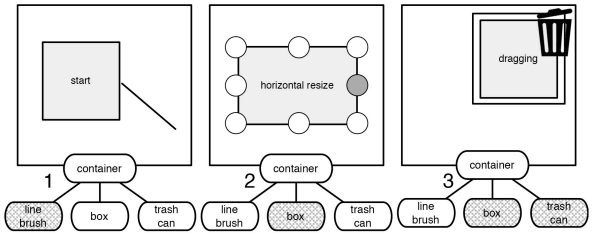

Figure 3. For each alternative in Figure 2, the system tracks which interactors in the hierarchy are different from the original (shaded in).

shows the interactors that would be marked as *changed* for three alternative interfaces associated with the running example in Figure 2.

When an action request is *final*, that implies that the action will have an impact on the application state (*e.g.,* saving a file, killing a process, *etc*.). If a final action is selected for execution, since the action is irreversible, the distribution across possible interface states must be reduced to a single interface resulting from executing that action alone. The decision about whether to accept any final actions is handled by a pluggable automatic mediator [14], which checks to see whether multiple final actions with similar high likelihood are all present or not.

When a final action is accepted, its associated interface alternative can be displayed without further modification and the system can wait for the next set of event samples. More typically there will be a large number of local actions whose associated interface alternatives need to be displayed, in which case we go on to the reduction step. A third possibility is that multiple final and local actions remain along with their interface alternatives, in which case we may show feedback about the alternatives but *defer* execution of the final actions until further user input arrives.

Reducing the Set of Alternative Interfaces

The result of executing multiple local actions in the *Selection* step is multiple interface samples. These represent a probability mass function (PMF) approximating the probability distribution over possible actions the user intends. In many cases, the number of interface alternatives exceeds what could reasonably be communicated to the user. Therefore, our feedback system must reduce the number of interface alternatives into a representative subset. The result is a new collection of alternate interfaces. In our example from Figure 2, the system feedback mechanisms choose to reduce all of the alternative interfaces to three different interfaces (Figure 2, bottom). These interface alternatives represent the major classes of actions possible in the system: drawing a line (Figure 2, bottom left); resizing the box *via* the handle on its edge (Figure 2, bottom middle); and dragging the box (possibly to a trash can that appears in response to the drag; Figure 2, bottom right).

The reduction step takes a list of interface alternatives (weighted with likelihoods) as input and returns a new list of alternatives (weighted with merged likelihoods) representing a reduced set. We have implemented and experimented with two reduction algorithms, however reduction is implemented in a pluggable fashion, allowing for other algorithms to easily be added.

Our first method is to pick the *N* most likely alternatives and re-normalize the probabilities across those alternatives. While simple, this has the disadvantage of not representing a wide range of interfaces. For example, the top 10 alternatives in Figure 2 might all be lines with slightly different start and end points. Showing only the top alternatives in this case would fail to convey the other possibilities.

Another method we implemented is to group together interface alternatives for which the same interactor or set of interactors are *changed,* and then select or create a representative alternative interface for each group. The likelihood of the representative alternative for a group is the sum of the likelihoods of the samples it represents. In this method, all three alternatives at the bottom of Figure 2 would be present in the reduced set of interface alternatives. For example, a single representative line would be included which has a position that represents a merged version of the full set of lines, as described next.

Although within a group the same interactors are *changed*, the specific details of those changes (represented as property values associated with the interactors) may differ from one sample interface to the next. If we select a single sample interface, it may not have appropriately representative property values. Thus, we update each property value of each changed interactor using a pluggable merge function, which takes a set of (alternative_value, likelihood) pairs and reduces them into a representative value. For example, for a property whose value is numerical, one merge technique would be to take the weighted mean. We provide several basic merge functions for primitive variable types. Developers may provide new merge functions, as well as specify which merge function is used when.

Fusion of Representative Interface Samples for Feedback

Once a small number of representative interfaces have been created, the feedback system fuses these interface alternatives into a single interface. More specifically, feedback mechanisms take in a PMF over interface alternatives (generated by reduction), fuse them in some way, and return a single interface that can be rendered to the user. This fused interface will include interactors copied from the representative interfaces, as well as specialized container and wrapper interactors that implement specific feedback presentation techniques. Figure 5 shows several sample fused interfaces.

Our general approach to fusion involves defining what interactors should be displayed, and how they should be displayed. The system walks down the interface hierarchy of the last certain interface. In tandem, it walks down the representative interfaces producing a fused tree according to

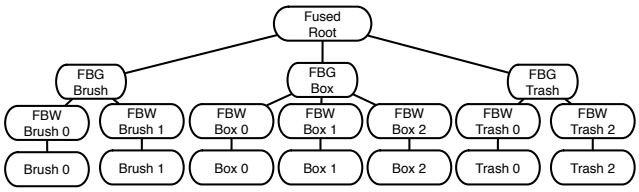

Figure 4. Interface hierarchy that would be generated as a result of grouping the alternatives in Figure 3. Numbers refer to the alternative the interactor came from. Alternative 0 is the most recent certain interface.

rules that differ depending on the specific type of fusion strategy used. For example, when an interactor is selected for display in the fused tree, it can be placed inside a container that implements the feedback mechanism in use. There are two primary types of containers that handle feedback. *Grouping* containers decide how to display a group of alternative interactors, while *Wrapper* containers modify the look and feel of a specific alternative that is being displayed to visually convey information (such as likelihood).

As an example of how this works, consider the interface hierarchy that would result from the alternatives shown in our running example (Figure 2). Figure 4 shows the new interface hierarchy that would be generated as a result of a very simple feedback system that overlays all three sample interfaces, each placed inside a type of feedback (FBG) container that simply displays them appropriately to indicate likelihood. The FBW containers in Figure 4 may change the opacity of their contents, for example.

Interface developers may use existing fusion strategies or write their own. In this paper we will describe two general types of fusion strategies available in our library (along with several extensions of each strategy) that can be used to create a wide range of specific feedback techniques. Many more fusion strategies are possible and may be built by extending the default fusion object and implementing a function that takes as input a list of (interface alternative, likelihood) pairs and returns a single interface.

An important note is that this architecture does not require developers of interactors to consider the uncertain state of the interactor (or the uncertain state of other interactors) when drawing themselves. Interactors draw themselves as with conventional interfaces. Communicating uncertainty is instead handled by special interactors inserted into the new interface hierarchy which modify how their children are rendered (by, *e.g.,* changing the opacity of all children based on the likelihood of an alternative).

Our architecture provides a simple yet powerful method for generating feedback about uncertain interfaces. In the next section we describe a range of feedback techniques provided in our library.

SPECIFIC FEEDBACK TECHNIQUES

Our architecture enables a large number of feedback techniques, ranging from simply overlaying alternative versions of interactors to showing an N-best list of alternative interfaces that allows a user to disambiguate their intent.

These techniques can be grouped into three categories of fusion strategies. First, we discuss a technique that groups and alters the appearance of alternatives. Next, we discuss interactive feedback techniques that not only show alternatives but also allow users to disambiguate intent. Finally we present contextual feedback selectors: a technique that picks which feedback object to use based on context.

Grouping Alternate Interactors and Altering Appearance

The first feedback category we explored centers around organizing similar interactors in the same grouping container and then applying wrappers to the alternatives that encode a wide range of visual feedback techniques. By default, these grouping containers are placed within a tree consisting of all of the interactors from the alternatives which are not *changed*. This ensures that the basic structure of the interface is visible in addition to feedback about uncertain alternatives. The *Feedback Group* container (FBG) makes decisions about which alternatives to display when. This can include context and timing dependent selection of what to display.

The *Feedback Wrapper* container (FBW) decides how to display alternatives. For example a wrapper might modify the transparency used when rendering an interactor (but otherwise leave its appearance unmodified), or it might extract some summary rendition (such as text or an icon) and display that instead of the normal presentation. Consider our running example of a draggable and resizable box (Figure 2). Figure 4 shows the new interface hierarchy that would be generated as a result of grouping alternatives. Figure 5 shows a few of the variations that are possible in this approach, including overlaying all of the alternatives using opacity or blur to indicate likelihood and adding a decoration to indicate likelihood (progress bars).

Our system provides generic versions of each type of container, which can be subclassed to provide a range of feedback. For example, our library includes containers implementing a variety different grouping techniques ranging from simply showing the most likely interface to overlaying all possible interfaces into a single interface. Below we describe some examples.

Our library includes *Feedback Wrapper* objects that adjust the opacity, scale, blur, contrast, and saturation of interactors based on their likelihood (Figure 5 shows a few examples using the running example in this paper). In addition to image adjustments, we can also render additional information using the wrapper. For example, we implemented a 'progress bar' style display that renders a bar indicating likelihood at the top left of its child.

We used an opacity wrapper to implement a demonstration of Octopocus [1]. In our implementation, each recognized gesture alternative is delivered as an event alternative and

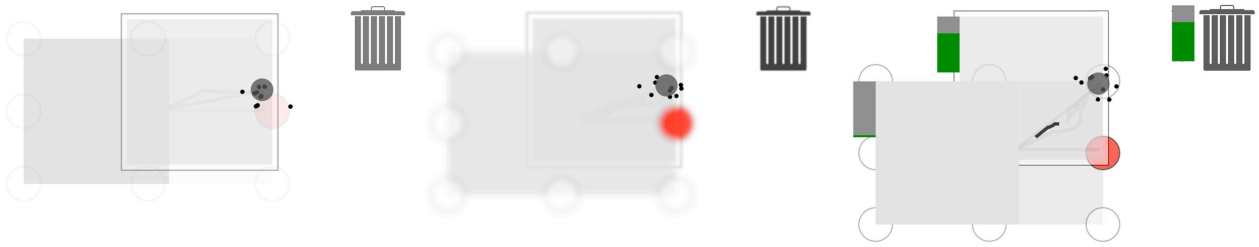

Figure 5. Three example renderings of the running example (Figures 2-5). Grey dots represent touch area, small dots represent touch samples. Alternative interfaces are rendered using: Left – opacity (the line is very unlikely, not visible); Center – blur; Right –progress bars (with all three interfaces overlaid). The draggable item and trash are most likely here.

creates an interface alternative as a result. A 'gesture interactor' in each alternate interface shows both the gesture's past trail and its predicted completion path (as computed using the fitting algorithm described in [1]). Each of these alternate interfaces (one for each alternatively recognized gesture input) is fused using the grouping technique, where the group simply overlays the interfaces and the wrapper simply adjusts opacity to reflect probability. This demonstration was implemented using a total of 140 lines of JavaScript, not including gesture recognition code.

Animating among multiple versions of an interactor is another way to indicate the range of uncertainty (this is briefly discussed in [13], but no implementation is presented). Our library includes *Feedback Group* containers that jitter or pulsate according to the number of alternatives in a group. Another animation technique fades between alternatives.

To summarize, by grouping alternatives and wrapping them in a variety of pluggable containers, we can achieve a range of feedback effects. From the user's perspective, this creates a feedback loop in which they can change their input (for example the shape of the gesture they are drawing) and monitor the system's response, ensuring that the correct interpretation is primary.

N-Best Lists

The methods described above communicate uncertainty and allow the user to respond, but do not directly support interaction. In contrast, a traditional n-best list allows the user to directly select an alternative.

This class of "N-Best List" feedback can take many forms. A simple and relevant example scenario is with gesture (Figure 6, Top). After the user completes a gesture, the gesture recognizer may have produced several likely alternatives. In this case, the interface may show an N-Best list illustrating the resulting interfaces that may occur as a result of her gesture (*e.g.,* adding several shapes, or drawing to the screen in red ink). Note that this example shows just the immediate resulting shape. Many rendering choices are possible, such as showing the entire resulting interface (Figure 6, bottom left) or highlighting only the portions that would change (Figure 6, bottom right).

Figure 7 shows the hierarchy generated as a result of fusing interface alternatives in our running example using an N-Best list. Note that the root interface into which the N-Best list is fused is by default the most recent certain interface, although this may be modified by the developer (*e.g.,* to show the alternative with the highest likelihood).

The *N-Best Item Group* is responsible for rendering the remaining alternatives (*i.e.,* the relevant portions of the N most probable interface alternatives) and providing a means for the user to select from among them. The display of each individual item is determined by the wrapper object placed over the subtree for the item. The N-Best Item Group handles layout and scaling, so that for example a rendering of an entire alternative interface may fit within a small box within the N-Best list. The N-Best Item Group also provides a selection mechanism for picking from the items.

Our implementation provides generic versions of the N-Best list container, which can be subclassed to achieve a variety of effects. Our library of N-Best lists varies along three dimensions: how uncertainty is communicated, how alternatives are presented, and how alternatives are interacted with. Below we give examples of demonstrations implementing each of these design dimensions.

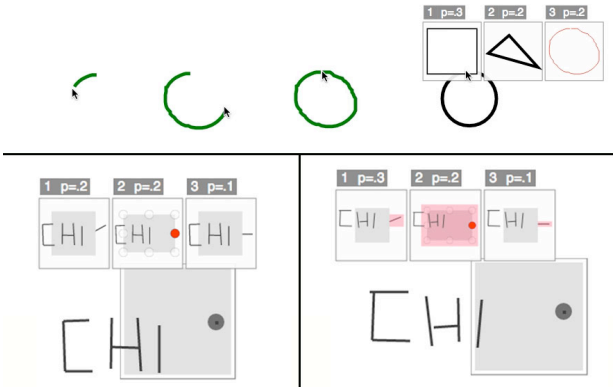

Figure 6. Examples of different N-Best lists implemented using our system. Top: After executing a gesture, show possible alternatives. Bottom Left: Show entire interface. Bottom Right: Show entire interface and highlight changes.

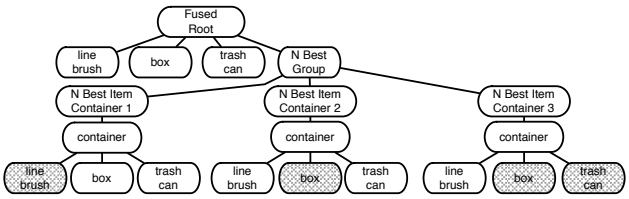

Figure 7. Interface hierarchy that would be generated as a result of making an N-Best list of alternatives in Fig 4.

Adjusting Appearance to Communicate Uncertainty: As with the first category of feedback we described, our library includes subclasses of the N-Best Item Group and associated wrapper containers which cover a range of examples of how this approach may be used. By adding a Feedback Wrapper above each *item*, in an N-Best list can indicate likelihood using any of the methods described earlier (opacity, blur, contrast, progress bar, *etc.*).

What Information to Show: As mentioned above, the tree processing phase determines what elements are placed under a given N-Best Item Group. The group then determines how each interface alternative is rendered. For example, our library of generation classes for N-Best lists include placing in the group: 1) the entire interface of each alternative (Figure 6, bottom left), 2) only the portion of the interface that has changed (Figure 6, top), 3) the entire interface, but using wrappers to highlight changes (Figure 6, bottom right), or 4) a compressed view of only the leaf interactor that has changed (Figure 9, center). Note that in the case of (4) we currently require the interactor to provide a function for drawing itself in compressed form, though in future work, aspects of this could be automated.

How to Disambiguate: As mentioned above, a fused feedback interface can handle input events to allow users to disambiguate intent. We have implemented techniques for selecting interface alternatives by: 1) selecting the alternative directly, 2) when dragging, crossing over an interface alternative to select it, and 3) using keyboard keys to select an alternative. When an alternative is selected, all competing alternatives are rejected.

Contextual Feedback Selectors

The feedback a developer may want to show may vary based on things like the degree of ambiguity between interfaces, the number of alternatives, which interactors are ambiguous, the source of user input, and user behavior. For example, a developer may wish to show the single most likely interface when the system is confident, but show a list of alternate interpretations when several alternatives are likely. Similarly, a developer may wish to only show the top choice (a trivial form of feedback) unless the user hesitates. Finally, feedback may be selected based on the type or ID of an interactor.

Figure 8. This touch interface disambiguates between when a user is scrolling or selecting text. Both alternatives and their likelihoods are tracked (right). Likelihood is determined by finger motion. Initially, the most likely interface (text selection) is shown. After the user pauses, an option to scroll instead appears in the upper left.

To facilitate this, we developed several feedback generator objects of a class we call *Contextual Feedback Selectors* which make decisions about all or part of how they generated feedback based on contextual information. When asked to generate feedback, these objects first check contextual information, then select which of one or more actual feedback types to display. In other words, these feedback generator objects select among other feedback generator objects based on context. Of course, developers may extend and develop their own feedback generator objects, as with the rest of the components of the system.

As part of the feedback library of our system, we implemented a feedback selector object for each contextual piece of information mentioned above. One particularly useful selector is an object that only shows feedback when a user hesitates. This is sensed by looking for a pause in either finger or mouse motion (*e.g.,* by comparing the current mouse position to a temporally smoothed position). By default, this feedback mechanism shows only the most likely interface. When a pause is detected, the feedback selector sends a synthetic input event to force interface update. The feedback selector then shows other possible interpretations.

One example demonstration of this technique is probabilistic text selection (Figure 8). Text is cumbersome to select on a mobile phone because of ambiguity between whether a press and drag indicates text selection or scrolling. We built a simple model measuring the likelihood of an intended scroll or text selection using simple gestural features (amount of vertical motion), and used this model to adjust the likelihood of action requests from the scroll view and underlying text view. As a result, the interface tracks both alternatives along with their likelihoods. This demonstration was implemented in 350 lines of code (including logic for tracking & rendering highlighted text, and scrolling).

The feedback we developed is shown in Figure 8 at the top left. In this case, we have combined an N-Best list with a *Context Feedback Selector* object that only shows feedback when the user pauses (possibly indicating an incorrect

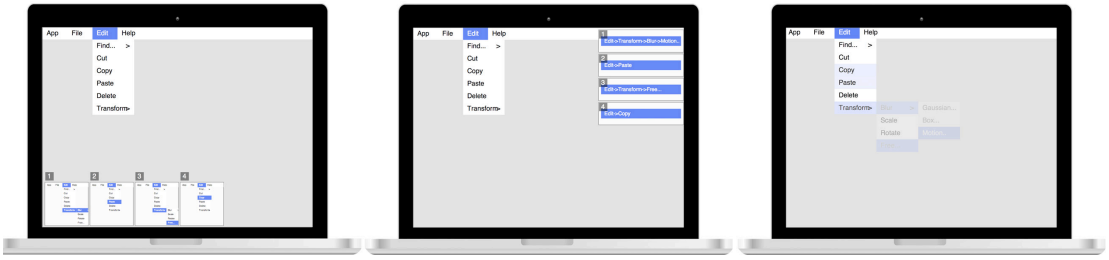

Figure 9. Demonstration of several forms of interactive feedback in a predictive menu system built with our architecture. The user has moved his mouse over the Edit item; a predictive model computes likelihoods of subsequent menu items. Left: An n-best list of the four most likely menu items is displayed. Users may select an alternative to jump to the interface alternative presented. Middle: A more compressed n-best list. Right: Every interface alternative is overlaid, opacity reflects likelihood.

guess). The N-Best list is showing only one alternative (the other choice) and renders a small cue in the upper left, allowing the user to switch to the alternate mode (scrolling).

CASE STUDY APPLICATIONS

In addition to the feedback techniques demonstrated above, we performed two case studies to demonstrate the value and test the feasibility of our approach. The first study is an exploration of feedback techniques in a predictive menu system, while the second is an implementation of a diagramming application inspired by the ideas of Igarashi's Interactive Beautification [7].

Predictive Menus

Our predictive menu is an example of an adaptive user interface. It uses a simple conditional probability table to guess which menu item a user intends to select, and allow him/her to more rapidly select the predicted item. Findlater and Gajos cover the design space of such adaptive interfaces and discuss evaluation issues in [5]. We used our feedback system to explore one design covered in [5] as well as two new designs. First, we overlay possible future menu selections on the menu (Figure 9, Right), adjusting the opacity so that likely items are highlighted. We also use an N-Best list showing possible menu items ordered by likelihood (Figure 9, Left). A third design shows a compressed N-Best list (Figure 9, Center).

Although the capabilities of predictive menus are fairly complex, our implementation is no more complex than that of a normal hierarchical menu. In fact, more complexity is devoted to the implementation of the hierarchical menu itself than to rendering predictions. In our demonstration, about 200 lines of code are used for interface setup and configuration, while 360 lines of code were needed for the hierarchical menu implementation (rendering, tracking which menus should be open, etc.). The drawing code for a menu item simply draws itself; it does not consider its likelihood, nor the likelihood of other alternatives.

When a menu item is selected, the menu interactor determines a list of predicted future events (and likelihoods) and dispatches a new probabilistic event, which in turn generates a list of interface alternatives representing possible future menu selections. These alternatives are then displayed using one of the feedback methods we developed.

This example demonstrates how complex interactions such as predicting future actions and showing possible completions can be accomplished using methods that are no more complex than those in existing interfaces. Our architecture handles the complexity of both tracking alternatives and (more relevant to the contribution of this work) fusing alternatives to render feedback.

Interactive Beautification of Touch Interfaces

Inspired by Igarashi [7] and Lunzer [12] we built a touch interface for drawing diagrams (Figure 10). Users may draw rectangles or ellipses using free form gestures (*e.g.,* draw a circle to make a circle) or by dragging a bounding box. They may also draw straight lines. Shapes and lines may be dragged and resized. When an item is dragged, a 'remove' drag target appears on the bottom of the screen, but only for the interface alternatives containing a shape being dragged. A line's endpoints snap to control points on shapes. If multiple snap points are possible, all alternatives are tracked. Items can be removed by being dragged to the bottom of the screen. When being dragged, appropriate feedback shows up indicating the remove region as a possible destination. Of course, the user may simply wish to place an item near the bottom of the screen. When multiple actions are possible the application shows the most likely action by default. When the user hesitates, the application shows an N-Best list. The user can then select one of the alternatives to disambiguate. This type of interaction would be very difficult to implement in a conventional user interface framework. In fact, one would need to build something almost as complex as our general feedback architecture to properly handle all cases (such as, the remove button appearing only when draggable items are selected). In contrast, this application was written largely without regard to probability, and its fairly complex operation was described about 800 lines of JavaScript.

CONCLUSION AND FUTURE WORK

This paper presents an architecture for fusing a probability distribution over possible interfaces into a single interface that communicates uncertainty and allows for disambiguation. We demonstrate the flexibility of our architecture with a collection of new and existing interaction techniques and two case study applications. Our work abstracts away the task of analyzing interface differences, selecting alterna-

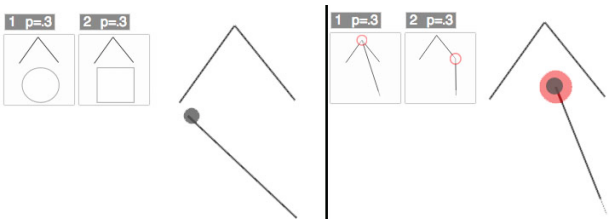

Figure 10. N-Best list in drawing application. Left: User drags on an empty area; she may create a line, rectangle, or ellipse. Right: When adjusting endpoints, several snap points are possible (hollow circles).

tives, and fusing alternatives, allowing interface developers to focus on interaction logic and presentation.

The examples in the previous sections demonstrate some of the feedback techniques that our architecture supports. In the future we hope to expand our library of feedback techniques, incorporating more work from the data visualization literature on portraying uncertainty, and allowing for interface alternatives to be modified. Additional future work includes developing more sophisticated algorithms for interface reduction (*e.g.,* using clustering techniques to identify similar interfaces) and identifying differences between alternatives. By providing an architecture that enables development of sophisticated feedback techniques for uncertain user interfaces, this work is helping to lay the foundation for a new era of nondeterministic user interfaces that leverage probabilistic models to better infer user intent.

ACKNOWLEDGMENTS

This work was funded by NSF Grant IIS1217929, and fellowships from Microsoft Research, Google and Qualcomm.

REFERENCES

1. Bau, O., Mackay, W., OctoPocus: A dynamic guide for learning gesture-based command sets. Proc. *UIST 2008*, 37-46.
2. Baumgarten, Barksdale, and Rutter. IBM® ViaVoice™ QuickTutorial®, 2000.
3. Bi, X., & Zhai, S. Bayesian Touch: A statistical criterion of target selection with finger touch. Proc. *UIST 2013*, 51-60.
4. Bier, E., Stone, M., Pier, K., Buxton, W., DeRose, T. Toolglass and magic lenses. Proc. *SIGGRAPH 1993*, 73 – 80.
5. Findlater, L. and Gajos, K. Design space and evaluation challenges of adaptive graphical user interafaces. *AI Magazine*. 30(4): 68, 2009.
6. Hudson, S.E., Newell, G. L. Probabilistic State Machines: Dialog management for inputs with uncertainty. Proc. *UIST 1992*, 199 – 208.
7. Igarashi, T., Sachiko, K., Hidehiko, T., Matsuoka, S. Pegasus: A drawing system for rapid geometric design. Proc. *CHI 1998*, 24 – 35.
8. Kristenson, P., Zhai, S. SHARK^2: a large vocabulary shorthand writing system for pen-based computers. Proc. *UIST 2014*, 43 – 52.
9. Lalanne, D., Palanque, P., Robinson, P., Vanderdonckt, J., Ladry, J. Fusion engines for multimodal Input: a survey. Proc. *ICMI 2009*, 153 - 160.
10. Li, Y. Beyond pinch and flick: enriching mobile gesture interaction. *Computer*. 42(12):87-89, 2009.
11. Li, Y., Lu, H., Zhang, H. Optimistic programming of touch interaction. *TOCHI*. 21(4): 24, 2014.
12. Lunzer, A., Hornbæk, K. Subjunctive interfaces. *TOCHI*. 14(4): 1-44, 2008.
13. MacEachren, A.M. Visualizing uncertain information. *Cartographic Perspective*. 13(13):10–19, 1992.
14. Mankoff, J., Hudson, S. E., Abowd, G. D. Interaction techniques for ambiguity resolution in recognition-based interfaces. Proc. *UIST 2000*, 11 – 20.
15. Mankoff, J., Hudson, S. E., Abowd, G.D. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. Proc. *CHI 2000*, 368-375.
16. Martin, J. Veldman, R., Béroule, D. Developing multimodal interfaces: a theoretical framework and guided propagation networks. Proc. *Multimodal Human-Computer Comunication, Systems, Techniques and Experiments 1998*, 158-187.
17. Oviatt, S. Mutual disambiguation of recognition errors in a multimodal architecture. Proc. *CHI 1999*, 576-583.
18. Schwarz, J., Mankoff, J., Hudson, S.E., Monte Carlo methods for managing interactive state, action and feedback under uncertainty. Proc. *UIST 2011*, 235 – 244.
19. Terry, M., Mynatt, E., Nakakoji, K., Yamamoto, Y. Variation in element and action: supporting simultaneous development of alternative solutions. Proc. *CHI 2004*, 711-718,
20. Terry, M., Mynatt, E. Side Views: persistent, on-demand previews for open-ended tasks. Proc. *UIST 2002*, 71 – 80.
21. Weir, D., Rogers, S., Murray-Smith, R., Löchtefeld, M. A user-specific machine learning approach for improving touch accuracy on mobile devices. Proc. *UIST 2012*, 465-476.
22. Wigdor, D., Williams, S., Cronin, M., Levy, R., White, K., Mazeev, M., Benko, H. Ripples: Utilizing per-contact visualizations to improve user interaction with touch displays. Proc. *UIST 2009*, 3 – 12.
23. Williamson, J., Continuous uncertain interaction. PhD thesis. PhD thesis, University of Glasgow.
24. Williamson, J. Hex: Dynamics and probabilistic text entry. Proc. *Switching and Learning in Feedback Systems 2003*. 333-342.
25. Wilson, A., Shafer, S. XWand: UI for intelligent spaces. Proc. *CHI 2003*, 545-552.