Towards a Unified Framework for Modeling, Dispatching, and Interpreting Uncertain Input

Julia Schwarz
HCII, Carnegie Mellon
5000 Forbes Ave, Pittsburgh, PA 15213 USA
julia.schwarz@cs.cmu.edu

ABSTRACT
Many new input technologies (such as touch and voice) hold the promise of more natural user interfaces. However, many of these technologies create inputs with some uncertainty. Unfortunately, conventional infrastructure lacks a method for easily handling uncertainty, and as a result input produced by these technologies is often converted to conventional events as quickly as possible, leading to a stunted interactive experience. Our ongoing work aims to design a unified framework for modeling uncertain input and dispatching it to interactors. This should allow developers to easily create interactors which can interpret uncertain input, give the user appropriate feedback, and accurately resolve any ambiguity. This abstract presents an overview of the design of a framework for handling input with uncertainty and describes topics we hope to pursue in future work. We also give an example of how we built highly accurate touch buttons using our framework. For examples of what interactors can be built and a more detailed description of our framework we refer the reader to [8].

ACM Classification:H5.2 [Information interfaces and presentation]: User Interfaces.- Graphical user interfaces.

General terms: Performance, Human Factors.

Keywords: Input Handling, Ambiguity, Recognition.

INTRODUCTION
Input handling in most modern interface toolkits depends on an established framework for modeling and responding to input that has been tuned over time to the needs of conventional graphical user interfaces (GUIs). However, this framework makes a number of assumptions about the nature of the inputs it deals with. For example, standard GUI toolkits implicitly assume inputs are certain to have occurred as reported. As we move to promising new technologies such as computer vision, free-space gesture recognition, pen input, and touch sensing, this assumption of certainty is being violated. It is no longer the case that reports about inputs are completely correct, without ambiguity or significant error. For example, in touch input the location of a touch ‘click’ is partly ambiguous simply because a user’s finger touches, not at a point, but an area (and the user cannot see how their contact area overlaps small objects underneath their finger). Similarly, a recognizer may produce one or more uncertain estimates of user intent.

If these inputs are processed using a conventional input framework, their uncertainty is resolved quickly and often simplistically. The result may seem arbitrary or unpredictable. Small errors in interpretation may lead to incorrect application actions that are difficult to recover from.

Our ongoing work aims to design a unified framework for modeling uncertain input, dispatching it to interactors, allowing developers to easily create interactors which can interpret uncertain input, giving the user appropriate feedback, and accurately resolving any uncertainty. This abstract presents an overview of the design of a framework for handling input with uncertainty, gives a concrete example of how our framework can be used to make better buttons for touch, and describes topics we hope to pursue in future work.

In the next section, we give a general overview of our framework. Our framework provides a model for probabilistic input and a means to make informed decisions on the basis of accurate probabilistic tracking of alternatives. Our
framework allows different interactors taken from a library to be used largely without regard to what other (probabilistic or conventional) interactors are being used in the same interface.

HANDLING INPUTS WITH UNCERTAINTY: OVERVIEW
Input frameworks can be broken into four areas: modeling the event, dispatching the event to interactors, interpreting and acting on these events. This section explains how our framework actually handles event modeling, dispatch, interpretation and action.

A proof of concept implementation of the ideas presented here has been implemented in C# on top of the .NET framework. Relevant implementation details are discussed within each subsection. Although our implementation is not a full toolkit, it constitutes a working demonstration of the ideas discussed below.

Modeling Probabilistic Events
To accurately model uncertainty, input event properties need to be expanded from a single fact to estimates representing a range of possibilities, which we will represent using a probability mass function (PMF). A PMF is a function which describes the relative likelihood that some discrete random variable has a particular value or that a continuous random variable falls within a finite range. For example, a Boolean variable can be represented as a PMF with probability of 0.2 of being true and 0.8 of being false and the PMF for a position indicates for each pixel the probability that the true position falls within the pixel. Since a PMF is derived from an underlying probability distribution, the integral over all possible values of the random variable must sum to 1. In our framework, all properties of an event may be represented by a PMF instead of a single certain value.

These PMFs can be instantiated in one of several forms, abstracted into an API. For example PMFs can be implemented as a table or histogram, can be implemented using an analytical function, or as a Monte Carlo sampling of the underlying distribution.

In addition, the input event as a whole is assigned a probability, indicating the likelihood that it (as opposed to a different input event) is what happened. The event probability is often 100%, however something like a recognizer that produces multiple possible alternatives might assign differing likelihoods to each alternative, each of which would be represented by a separate input event. Figure 1 illustrates a sample uncertain event.

Event Dispatch
The goal of event dispatch is to select a candidate set of interactors that may represent the proper recipient(s) of an event, and then deliver that event to each recipient. Although position or interactor focus are most commonly used, other interesting policies are possible. For example, input could be dispatched based on its proximity to a target or whether it surrounds a target. Probabilistic selection should be equally flexible, with the added requirement that events are delivered to all interactors that may be plausible recipients.

Selecting which interactor should receive an event
We support a probabilistic notion of dispatch in which events are delivered to all interactors which are candidates for input. Candidacy is determined using a scoring mechanism that considers event properties and interactor state. These scores are provided by querying each interactor. The resulting probabilistic selection list can be seen as a PMF. Past tools have typically not provided structured support for uncertainty about who should receive an event (termed target ambiguity [7]).

Conventional dispatch usually determines candidacy based on one of two algorithms focusing on event type (focused dispatch) or position (positional dispatch). Both are handled in the same way in our framework. Each interactor examines the type, position, or any other property of an event to determine a selection score between 0 and 1 inclusively. Selection may be based on event types, as well as measures of “overlap”, “nearness” or other more radical spatial relationships. By loosening the requirements for “overlap” and using logic and contextual information to determine “nearness”, we can enable inexact interaction [2]. A nice property of the framework is that more exotic styles of dispatch can be supported simultaneously with more conventional styles because of the uniform way in which uncertain input is delivered.

As a simple example of this process, consider the buttons in Figure 3, which are smaller than the touch area. They are designed to calculate a selection score based on the integration of the location PMF over the button bounds. Because buttons look at the integration over the non-uniform location PMF, if the finger completely covers two or more buttons, the buttons closest to the center of the finger (where probability density is higher) will have the highest score. In contrast, an interactor used in a pen based system might return a high selection score for a circling gesture event which encloses it, rather than requiring overlap. Alternatively we might support underlining in a pen system using a combined measure of nearness (without overlap) and parallelism to calculate the selection score.

The result of the selection process is a candidate list of interactors with associated probabilities (selection scores) indicating the likelihood that they are the intended target of the user’s input. These selection scores are normalized across all interactors so that they sum to 1.

Figure 3: A probabilistic input scenario involving a touch over three very small square buttons.
Finally, for each interactor, the normalized selection score, and probability that the event actually occurred are multiplied together to determine a final probability to be associated with dispatch of the event to that specific interactor.

**Interpretation, Feedback and Action**

At this point, it is up to the interactor to interpret the meaning of the event based on its internal state, update that state, and decide whether there is a possible action it might take as a result. As indicated earlier, it is often valuable to provide immediate feedback to express a system’s current understanding of the user’s input – ambiguous or otherwise. As a result, feedback or other temporary, fully reversible actions, are modeled separately from actions that have permanent and/or irreversible consequences. In our framework we only allow interactors to modify their own appearance to avoid conflicting representations of feedback.

The interactor informs the system of any temporary actions that it will take using a temporary action object that encapsulates key information about the action useful for later reversing it. Similarly, possible (final) actions are encapsulated in a possible action object (also referred to as a final action request), which has an associated probability that this interactor is really in a state compatible with that possible action. This possible action is passed to the system along with a Boolean indicating whether the request must be finalized immediately, or can be handled in a lazy fashion.

**Mediation**

A mediator’s job is to choose between competing (and potentially conflicting) actions. Typically, the mediator will make this choice when it receives a finalize request. In that case, the mediator will make one of three choices: (1) it may select an action (in our implementation simply the most probable one) (2) it may decide that no action is probable enough to execute and cancel all actions (3) it may decide that it needs additional information and request some from the user, deferring mediation until the user responds e.g., while an N-best list style mediation dialog is offered to the user. Many other sophisticated interactive mediators are also possible [6]. In some cases, the mediator may decide to finalize even without a request to finalize. For example, if there is one possible action that is significantly more likely than any others (i.e., ambiguity is very low), the mediator can choose to automatically finalize that action.

Once a possible action is selected, the associated interactor is notified that the action has been finalized (and can act on it) and any other interactors with possible or temporary action requests are notified that their actions are canceled.

One nice property of the temporary/possible action split is that feedback is provided early, and may be visible up to and through the mediation process. This can help a user to adjust their input to implicitly mediate, or provide the user with needed context during more explicit cases of mediation. When interactors are notified that an action has been canceled, feedback can be removed. Any pending actions (temporary or possible) registered since the last time an action was finalized are canceled. The interactor associated with each action is expected to roll back its state and undo any temporary changes such as displaying feedback.

**IMPROVING BUTTONS FOR TOUCH: A COMPLETE EXAMPLE**

To illustrate the complete flow of input under our framework, we illustrate how our framework enables the creation of highly accurate buttons. We illustrate our example with the interface shown in Figure 5. This example adds realism to the example associated with Figure 3. Although both examples are fairly basic, our implementation enables something very powerful: By allowing users to select among very small buttons using a finger, we open the door to mobile applications that can take advantage of increasing screen quality rather than being limited by finger size.

1. The user touches the screen and a probabilistic touch event is generated and passed to the dispatcher. As described before, a single event with 100% probability is generated. The position information within the event is uncertain and is provided as a PMF derived from a 2d Gaussian function centered about the touch center with standard deviation equal to the width/height of the touch area divided by 4.

2. Each interactor in the interface is asked for a selection score which indicates whether it should be in the candidate dispatch list. It is obvious in Figure 3 that this includes all three buttons, but note that in the case of Figure 5, this includes the “yes”, “no”, “cancel” and “close” (“X”) buttons, as well as the dialog box title bar. Each interactor computes its selection score by integrating the PMF over the button area. After receiving the selection scores, the dispatcher delivers a touch down event to the ‘yes’ and ‘no’ buttons, as these are the only buttons which have returned nonzero selection scores.

3. Our button class acts as soon as it receives the initial touch (without waiting for a release event), so each button’s tracking of state and interpretation of the input is trivial. The response of the “yes” and “no” buttons is to each emit a possible action and request finalization. Since the interpretation of events is very simple in this example, the incoming event score is simply passed along as the action probability.
4. When the mediator receives the possible actions, it notes the presence of a finalization request and immediately makes a decision among the possible actions. The mediator we implemented selects the action with the highest probability to go forward. In the case of Figure 5, this means that it calls finalized() on the “yes” button, passing its action back so that it can respond, and calls cancel() on the “no” button. All other buttons are ignored, since they did not submit possible or temporary actions.

5. The selected button notifies the application that it should execute its action. If any other button had displayed any feedback, at this point it would remove that feedback.

This example has the practical effect of making the effective target size larger. This is one of the central ways in which one can make it easier to select a target (“beat” Fitts’ law [Error! Reference source not found.]) and so it is not surprising that many advanced interaction techniques take a similar approach. Our framework’s approach to identifying candidate targets makes adaptive versions of techniques for beating Fitts’ law essentially the default (having an effect analogous to adaptive area cursors [5] or bubble cursors [3]).

FUTURE WORK

Our framework presents a new way of thinking about and handling user input. As such, it opens a wide range of possible directions for future work, some of which we touched upon in our presentation thus far.

Specifically, we mentioned that during the interpretation phase, interactors could probabilistically track their state based on probabilistic inputs and act accordingly. The details of this are fairly complex and are a topic we plan to cover in future work. The challenge here is to design a framework which probabilistically tracks interactor’s state as in [4] without requiring the developer to describe a probabilistic finite state machine.

A second problem we need to address involves tracking and resolving complex events. For example, a touch gesture such as a swipe is made up of many smaller touch events. If a swipe event is accepted, which interactors should the mediator cancel?

Finally, we’d like to formally evaluate the impact of our toolkit on end-users. We’ve begun to do this in [8] in our evaluation of how our toolkit helps people with motor impairments, however we’d also like to formally evaluate the impact of our framework in areas such as touch input and voice.

Once all of the details of our framework are refined, we plan to integrate our contributions into a toolkit which developers can use to handle inputs with uncertainty. There are many questions involving usability and performance of this toolkit which we hope to answer. First, we want to understand how easy it is for developers to write interactors using our toolkit. Second, we want to evaluate the performance impact of our toolkit on modern computers.

Through our contributions we hope to introduce a new paradigm for handling user input.

CONCLUSION

The advent of new input technologies introduces a new type of input: input with uncertainty, which conventional input frameworks deal with poorly. We presented a framework for the robust and flexible handling of inputs with uncertainty as an extension of the conventional event-based input handling framework, and showed how our framework supports existing interaction improvements, enables new interactions, and can be used to create techniques improving user experience for people with motor impairments. This abstract highlights the importance of properly dealing with uncertain input and presents an overview robust and flexible framework for handling inputs with uncertainty, as well as directions for future work.

ACKNOWLEDGMENTS

This work was funded in part by grants IIS-0713509, IIS-0803733, and IIS-0840766 from the National Science Foundation and a grant from the Intel Research Council. This project was also supported by a National Science Foundation Graduate Research Fellowship and an ARCS Foundation Fellowship.

REFERENCES