How to Model, Evaluate and Generate Interaction Techniques?

Caroline Appert
LRI & INRIA Futurs
Bât. 490, Université Paris-Sud
91405 Orsay, France
appert@lri.fr

ABSTRACT
This article summarizes the process I have developed to describe, evaluate and facilitate the creation of novel interaction techniques. First, it presents the CIS model for describing interaction techniques and predicting their effectiveness in real contexts of use. CIS shows that there is no absolute best technique but that performance depends on the context of use. The article then shows how to improve a technique by optimizing subcomponents of its CIS structure. Finally it describes SwingStates, a toolkit designed to help develop novel interaction techniques by exploring different CIS structures.

ACM Classification: D.2.2 [Design tools and Techniques]: User Interfaces; H.5.2 [Information Interfaces and Presentation]: User Interfaces – Graphical User Interfaces.

General terms: Design, Human Factors, Performance.
Keywords: Model, Evaluation, toolkit, context.

INTRODUCTION
Research in HCI has produced many novel interaction techniques aimed at improving the usability of graphical applications. Yet very few make it into industrial products. This may be due to the difficulty of assessing the actual value of a technique and the required efforts to implement and test it quickly and efficiently. The goal of my Ph.D. is to develop a framework to describe, evaluate and create interaction techniques and facilitate their widespread use.

Commonly, measuring the effectiveness of an interaction technique consists in conducting a controlled experiment that compares several techniques on a given task. However, designing such experiments requires a large amount of work and provides results that are difficult to generalize due to the specificity of the chosen task. What designers need is a model that can describe interaction techniques and predict their performance in different contexts of use before conducting experiments. Such a model would ease the design of experiments and increase the validity of their results by helping researchers understand the effect of context on their performance. I have developed the Complexity of Interaction Sequences (CIS) model [2] to describe WIMP or post-WIMP interaction techniques and their contexts of use through the notion of *interaction sequences* in order to predict their performance in context.

Automatically generating interaction techniques can lead to interesting compositions of existing widgets but is unlikely to lead to very innovative techniques. The generative approach I am exploring instead consists of providing researchers and designers with tools to identify high-level properties of the techniques they create and to quickly prototype their ideas. CIS serves the first goal by exploring new techniques according to different properties. SwingStates [4] is a toolkit developed to serve the second goal. Since programming graphical user interfaces is notoriously difficult, developers stick to the existing set of standard widgets provided by GUI toolkits. SwingStates is an extension to the existing, widely used Java Swing toolkit to easily program non-standard interaction techniques without requiring to learn new development tools.

The rest of this article shows how to describe and evaluate interaction techniques in context with the CIS model and how to generate new interaction techniques. It concludes with future work and open questions.

DESCRIBING INTERACTION TECHNIQUES
The Complexity of Interaction Sequences (CIS) model can describe the structure of interaction techniques and address context through the notion of an *interaction sequence*.

The level of description of CIS is intermediate between the low level of input devices and the high level of user tasks. An example of a low-level description is the taxonomy introduced by Card et al [8] in which input devices are described as translators from physical properties to logical parameters of an application. An example of a high-level description is the family of descriptive and predictive GOMS models [13] based on task analysis. CMN-GOMS, NGOMSL and CPM-GOMS describe a task as a hierarchy of goals with operators as leaves. A goal can be reached by a method, described as a sequence of sub-goals and operators. The Keystroke-Level Model (KLM) is the exception in this family as it describes a task as a totally ordered sequence of operators, i.e., low-level tasks. Unlike CIS, it does not explicitly exhibit properties of the technique nor the context of use.

CIS describes an interface as a set of objects that users can manipulate. The *state of the interface* is defined by the set of objects and the values of their attributes. A *manipulation* is a creation, modification or deletion of objects. It is described by a tuple of the form (command, attributes). The *interaction space* is the set of manipulations available to the user in a given interaction state. An *interaction step* is a sequence of *actions* that progressively reduce the interaction space to a single manipulation that it executes, leading to a new state and therefore a new interaction space.

An *interaction technique* is a set of interaction steps. CIS describes it with a directed graph, called the *interaction graph*. Figure 1 shows the interaction graphs for two techniques, a traditional fixed palette and a toolglass [7], in a simple interface that can create rectangles, ellipses and triangles of a predefined size. Nodes and edges of this graph correspond to atomic motor and perceptual actions such as pointing. We distinguish two types of actions, *acquisitions* and *validations*, described by arcs and nodes in the interaction graph:

- An *acquisition* (arc) identifies a subset of the current interaction space; it is usually achieved by moving an object, typically the cursor, over a tool, a work object or a position. In the interaction graph, the arc is labeled by the object being moved and its target.
- A *validation* (node) confirms the subset identified by an acquisition, which becomes the current interaction space; it is usually achieved by clicking a button or typing a key. In the interaction graph, each node (except for the root) is a validation, labelled by the element(s) of the manipulation it instantiates and the duration of the physical action.

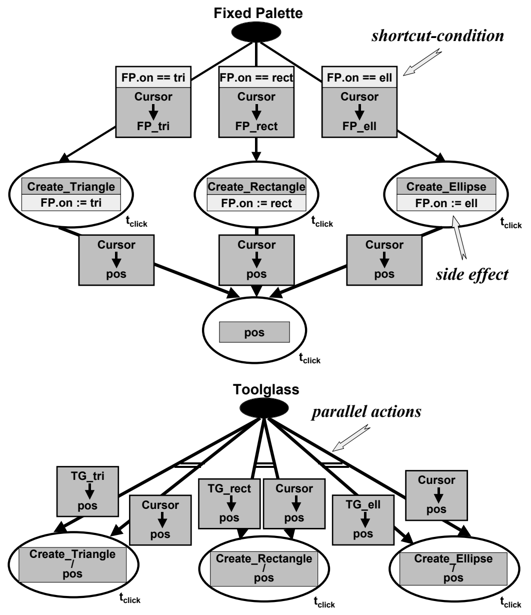


Figure 1: The CIS description of a fixed palette (top) and a toolglass (bottom).

Interaction graphs exhibit some high-level properties of interaction techniques that can be used to compare them qualitatively such as:

- *Order and Parallelism:* An interaction technique imposes a sequential and/or parallel organisation of its component actions, visualised by the structure of the interaction graph and the use of the parallel construct. For example, a toolglass is highly parallel while a palette is highly sequential.
- *Persistence*: Interaction techniques may have side effects such as setting attributes of tool objects. These side effects may affect how the interaction technique is used the next time, as described by the *shortcut-conditions*. For example, the tool selected when using a traditional palette is persistent, so, for example, creating two rectangles in a row only requires selecting the rectangle tool once.

How can we capture the context of use using this formalism? Mackay [14] conducted an experiment showing that users organize their interactions according to their cognitive context. For instance, in a copy context, users tend to create objects of the same type in sequence while in a problem solving context, they create objects according to their thought process. With CIS, we can operationalize such a context of use by *interaction sequences*, i.e. a sequence of manipulations, and evaluate the technique with respect to these sequences.

EVALUATING INTERACTION TECHNIQUES

In order to measure the efficiency of an interaction technique, we introduce a *measure of complexity*, inspired by the measure of complexity used in evaluating algorithms. We define a *problem* to be solved as a state to be reached using an interaction sequence. The *size* of the problem is the length of the sequence. The *actions* are the acquisition and validation actions used in an interaction sequence that solves the problem, i.e., which activate manipulations in the sequence. The *complexity* of an interaction technique for the given sequence measures the cost of the actions relative to the size of the problem when using this technique. As with algorithms, we can explore the best- and worst-case complexities, i.e. the interaction sequences that solve equivalent problems with the lowest and highest values.

CIS constructs the *sequence graph* that describes the overall interface by merging together the roots of all the interaction graphs and adding *return arcs* from each leaf to the new root. Any path starting and ending at the root of the sequence graph instantiates an interaction sequence (such paths will typically go through the root multiple times). CIS computes the path *P* that activates the manipulations of the sequence *Seq* and evaluates the action and time complexity. Time complexity is computed by summing the time taken by each non-return arc and node in *P*. When a shortcut-condition is true, the corresponding arc and end node are not counted.

The CIS predictive power stands on labeling actions by well-known predictive motor and perceptual laws. The time taken by an arc is the sum of the time taken to choose that arc at the parent node and the time taken to traverse the art, typically using a pointing action. The former is estimated by the predefined time of the physical action, e.g. a click, and Hick's law [10], which models the choice selection time: $k \log_2(1+n)$ for n arcs; the latter is estimated by Fitts' law [9], which models the pointing time: $a + b\log_2(1+\frac{D}{W})$ for a target of size W at distance D. The time taken by a node is the constant time that labels the node. When the shortcut-condition of an arc is true, the times of the arc and its end node are ignored. When two series of actions are parallel, the complexity of the whole is approximated by the maximum of the complexities of each branch.

In order to test the validity of CIS predictions, we ran a controlled experiment [2] comparing different techniques on different sequences. The sequences we used corresponded to the two cognitive contexts identified by Mackay [14]: copy and problem solving. We compared two techniques from the original experiment, the standard tool palette and the toolglass, and added a third one, the bimanual palette. The latter was implemented in the CPN2000 interface [6] and combines the persistence property of the standard palette and the parallelism of the toolglass. It uses two hands and two cursors: the non-dominant hand selects tools in the palette with the first cursor while the dominant hand selects objects in the work area with the second cursor.

The results revealed that although CIS underestimates the execution times, it predicts the pattern correctly, i.e. the relationship between interaction sequence and the efficiency of a technique. The results also confirmed the results reported by Mackay. This supports the fact that CIS can be used as an evaluation tool at least in the preliminary phases of designing a new technique or interface.

OPTIMIZING THE COMPONENTS OF A GRAPH

Starting from an interaction graph, i.e. the structure of the interaction technique, designers can improve it by optimizing the arcs and nodes actions. For example, replacing a clicking interaction with a crossing one [1] reduces the time of a node (clicking takes 200 ms while crossing does not take any extra time). Similarly, in order to reduce the time of an arc, we introduced a novel pointing interaction technique, the OrthoZoom Scroller [3].

The OrthoZoom Scroller is a continuous multi-scale pointing technique requiring only a mouse. It extends a traditional scrollbar into a 1D multi-scale navigation technique. It behaves like a traditional scrollbar when the mouse is moved within the bounds of the scrollbar. When dragging the mouse outside the bounds of the slider, it continuously changes the granularity/zoom of the slider. The granularity decreases as the mouse cursor gets farther away from the scrollbar bounds. In other words, moving the mouse along the scrollbar orientation performs a pan whereas moving it orthogonally performs a zoom.

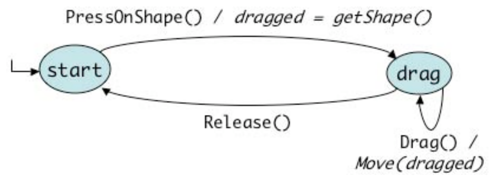
We conducted a controlled experiment showing that the OrthoZoom Scroller follows Fitts' law and is about twice as fast as Speed Dependant Automatic Zooming (SDAZ) [12] to perform pointing tasks with an index of difficulty in the 10-30 bits range. Until then, SDAZ was the fastest mouse-based multi-scale scrolling technique with continuous control that outperformed standard scrolling interfaces.

For lack of a standard, the various multi-scale navigation techniques that have been reported in the literature recently are difficult to compare. In order to help designers assessing the effectiveness of different pointing techniques, we are currently developing an experimental platform [9] dedicated to the comparative evaluation of multi-scale navigation techniques using Fitts' methodology.

GENERATING NEW INTERACTION TECHNIQUES

Creating novel interaction techniques often requires a more radical approach than optimizing an existing one, resulting in a different structure of the CIS graph. In order to facilitate the prototyping and testing of novel techniques and their integration into real applications, we developed SwingStates [4], a library that adds state machines and a canvas widget to the Java Swing user interface toolkit. SwingStates tightly integrates state machines, the Java language and the Swing toolkit. We chose to extend this widely used tooliit instead of creating a new onw in order to facilitate its adoption. Since many developers are familiar with Java/Swing, SwingStates capitalizes this knowledge and brings novel interaction techniques closer to real applications.

Unlike traditional approaches, which use callbacks or listeners to define interaction, state machines provide a powerful control structure that localizes all of the interaction code in one place. Figure 2 illustrates how SwingStates takes advantage of Java's inner classes, providing programmers with a natural syntax and making it easy to follow and debug the resulting code. It reduces the potential for an explosion of states by allowing multiple state machines to work together.



```
StateMachine sm = new StateMachine("DnD") {
  SMShape dragged = null;
   public State start = new State() {
     Transition  dragOn =
       new PressOnShape(BUTTON1, "drag") {
        public void action() {
          dragged = getShape();
        }
    };
  };
   public State drag = new State() {
    Transition drag = new Drag(BUTTON1, "drag") {
      public void action() {
        move(dragged);
      }
     };
     Transition  dragOff =
      new Release(BUTTON1, "start") { } ;
  };
};
```

Figure 2: The graphical representation and the SwingStates' code of a simple state machine for dragging objects. (State=circle; transition=arrow, events in roman font and actions in italics).

The SwingStates canvas manages a display list of shapes, including simple and arbitrary paths, text strings, images, etc. Each shape has a geometric transform that combines translation, scaling and rotation. Shapes may have a parent shape, with transformations relative to the parent, and a clipping shape. Shapes can be tagged to group them and state machines can be attached to a canvas, to individual shapes, and even to tags. Several state machines can be active at once, running in parallel.

SwingStates allows developers to easily implement post-WIMP techniques. SwingStates can be used to change the behavior of standard Swing widgets, e.g., to customize buttons so that crossing rather than clicking activates them. Figure 3 shows two other examples of modified Swing widgets: an entry field to enter a numeric value using a joystick-like interaction and a pie menu to change the background color of other widgets.

We used an earlier version of SwingStates' Canvas [5] in a Master's level computer science course where students implemented a wide variety of interaction techniques, including toolglasses, magnetic guidelines and side views. Unlike our attempts in previous years with other toolkits, all students completed their projects with little or no help, demonstrating the power and simplicity of SwingStates to implement advanced interactions.

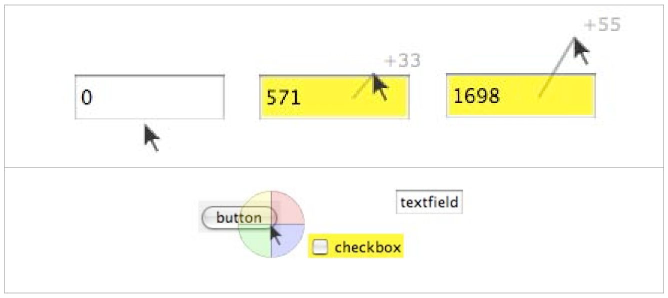


Figure 3: A joystick text entry (top) and a pie menu (bottom) that changes the color of other arbitrary Swing widgets: button, checkbox, text field.

FUTURE WORK AND DISCUSSION

This article has presented the current state of my Ph.D., whose goal is to provide a framework for describing, evaluating and creating interaction techniques. I introduced CIS, a model that describes the structure of interaction techniques and predicts their relative performance for different interaction sequence, and I presented how to generate new interaction techniques by optimizing the components of existing ones as well as by creating new ones with the SwingStates extension to Java/Swing.

I am currently working on a stronger validation of the effect of context on performance, by studying different navigation techniques in different context of use, e.g. when the user knows the location of the target he wants to reach as opposed to when he is looking for a target. I am also working on bridging the gap between CIS and SwingStates so that the CIS model can be derived from the SwingStates code by adding some metadata to it. This allows comparing predictions with 'live' results to improve CIS.

I am looking forward to the UIST Doctoral Consortium for presenting this work and gathering feedback. I am particularly interested in discussing ways to validate the various components of my work. Also, while CIS is a model dedicated to graphical interaction, it would be interesting to study how a similar approach could apply to other interaction styles such as tangible interfaces and Ubicomp.

REFERENCES

1. Accot, J. and Zhai, S. 2002. More than dotting the i's - foundations for crossing-based interfaces. *In Proc. Conference on Human Factors in Computing Systems*. CHI'02. pages 73-80.

2. Appert, C., Beaudouin-Lafon, M. and Mackay, W.E. (2004). Context matters: Evaluating interaction techniques with the CIS model. In *Proc. People and Computers*. HCI'04. pages 279-295

3. Appert, C. and Fekete, J.D. (2006). OrthoZoom Scroller: 1D Multi-Scale Navigation. *In Proc. ACM Conference on Human Factors in Computing Systems*. CHI'06. pages 21-30.

4. Appert, C. and Beaudouin-Lafon, M. SwingStates: Adding State Machines to the Swing Toolkit. *Accepted to ACM Symposium on User Interface Software and Technology*. UIST'06. 4 pages.

5. Appert, C. and Beaudouin-Lafon, M. (2006) SMCanvas : augmenter la boîte à outils Java Swing pour prototyper des techniques d'interaction avancées. In *Proc. conférence francophone sur l'Interaction Homme-Machine*. IHM'06. pages 99-106.

6. Beaudouin-Lafon, M. & Lassen, H.M. (2000). CPN2000: A Post-WIMP Graphical Application. *In Proc. ACM Symposium on User Interface Software and Technology* . UIST'00. pages 181-190

7. Bier, E.A., Stone, M.C., Pier, K. & Buxton, W. (1993). Toolglass and Magic Lenses: the See-Through Interface. *In Proc. ACM Siggraph*, pages 73-80.

8. Card, S.K., Robertson, G. & Mackinlay, J. A. (1991). Morphological Analysis of the Design Space of Input Devices. *In Proc. ACM Transactions on Information Systems*, 9(2), pages 99-122.

9. Fitts, P. M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391, 1954.

10. Y. Guiard, J.D. Fekete, Y. Du, C. Appert, M. Beaudouin-Lafon and O. Chapuis. (2006). Shakespeare's Complete Works as a Benchmark for Evaluating Multiscale Document-Navigation Techniques. In *Workshop BELIV'06.* AVI'06, 6 pages.

11. Hick, W.E. (1952). On the Rate of Gain of Information. *Quarterly Journal of Experimental Psychology*, 4, pages 11-26.

12. Igarashi, T. and Hinckley, K. (2000). Speed-dependent automatic zooming for browsing large documents. *In Proc. ACM symposium on User interface software and technology*. UIST'00, pages 139-148

13. John, B. E. & Kieras, D. E. (1996). The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *Proc. ACM Transactions on Computer-Human Interaction*. 3(4), pages 320-351.

14. Mackay, W.E. (2002). Which Interaction Technique Works When? Floating Palettes, Marking Menus and Toolglasses support different task strategies. *Proc. ACM Conference on Advanced Visual Interfaces.* AVI'02. pages 203-208.