Designing Interaction, not Interfaces

Michel Beaudouin-Lafon[*]

Université Paris-Sud

LRI - Bât 490

91405 Orsay - France

+33 1 69 15 69 10

mbl@lri.fr

ABSTRACT

Although the power of personal computers has increased 1000-fold over the past 20 years, user interfaces remain essentially the same. Innovations in HCI research, particularly novel interaction techniques, are rarely incorporated into products. In this paper I argue that the only way to significantly improve user interfaces is to shift the research focus from designing interfaces to *designing interaction*. This requires powerful interaction models, a better understanding of both the sensory-motor details of interaction and a broader view of interaction in the context of use. It also requires novel interaction architectures that address reinterpretability, resilience and scalability.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *graphical user interfaces (GUI), interaction styles, theory and methods, user-centered design.*
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces.*
I.6.3 [**Computer Graphics**]: Methodology and Techniques – *interaction techniques.*

General Terms

Design, Human Factors, Theory.

Keywords

Interaction paradigm, Interaction model, Instrumental interaction, Design principles, Situated interaction, Interaction architecture.

1. INTRODUCTION

Over the past twenty years, the computing power of personal computers has increased by at least three orders of magnitude (Table 1), while their input/output devices have stayed essentially the same. In many cases, the hardware interface has become more limited, with the advent of PDAs and cell phones and their tiny screens and buttons.

The essential character of the user interface has not evolved either. Today's Macintosh users use the same WIMP interface

Table 1. Comparing two personal computers 20 years apart.

	original Macintosh	iMac 20"	comparison
date	January 1984	November 2003	+ 20 years
price	$2500	$2200	x 0.9
CPU	68000 Motorola 8 MHz 0.7 MIPS	G5 1.26 GHz 2250 MIPS	x 156 x 3124
memory	128KB	256MB	x 2000
storage	400KB floppy drive	80GB hard drive	x 200000
monitor	9" black & white 512 x 342 68 dpi	20" color 1680 x 1050 100 dpi	x 2.2 x 10 x 1.5
devices	mouse keyboard	mouse keyboard	same same
GUI	desktop WIMP	desktop WIMP	same

(Window Icon Menu Pointer) as in 1984: an iconic Finder, applications with a menu bar and overlapping windows, and a limited set of interaction techniques, including "widgets" (menus, buttons, dialog boxes, scrollbars), "drag&drop" and "copy-paste" to transfer data within and across applications.

WIMP interfaces have reached their limits in the face of three major challenges: the exponential growth in the amount of information each individual user deals with; the distribution of this information over multiple computers and devices, including mainframes, desktop computers, laptops, PDAs and cell phones; and the growing range of computer users, with their wide variety of skills, needs and expectations.

Why is it that the tremendous increase in computational power has not benefited user interfaces and made them much more powerful?

---

[*] PCRI Projet In Situ - http://insitu.lri.fr
Pôle Commun de Recherche en Informatique du Plateau de Saclay CNRS, Ecole Polytechnique, INRIA, Université Paris-Sud.
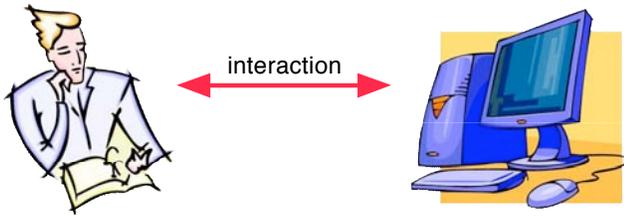
Figure 1. Interaction as a phenomenon.

HCI researchers have developed and evaluated a variety of innovative novel interaction techniques but few have been adopted in the marketplace. For example, pie menus, invented in the early 80s [13], are demonstrably more effective than linear menus and have a very low learning cost, yet only a handful of applications use them. Why? How is it that hardware innovations like mouse wheels, with their associated manufacturing costs, are adopted more easily than software innovations like pie menus, which could be added to user interface toolkits of all major platforms at practically no cost?

Two major barriers restrict wider dissemination of interaction research. First, although HCI researchers have created a variety of novel interaction techniques and shown their effectiveness in the lab, such "point designs" are insufficient. Software developers need models, methods and tools that allow them to transfer these techniques to commercial applications. Second, WIMP user interfaces have been so stable and so universally adopted over the past twenty years that the users' cost of change is very high. Before users switch to a radically new user interface paradigm, they must perceive the immediate benefits as dramatically outweighing the costs of learning and transfer to the new system. Macintosh users made such a jump 20 years ago; PC users followed 10 years later, from DOS to Windows.

In this paper I argue that, if we are to create the next generation of interactive environments, we must move from individual point designs to a more holistic approach. We need a solid theoretical foundation that combines an understanding of the context of use with attention to the details of interaction, supported by a robust interaction architecture. I characterize this shift as *designing interaction* rather than simply designing interfaces.

2. ANALYSING INTERACTION

Figure 1 illustrates interaction as a *phenomenon* between a user and a computer. This phenomenon is controlled by the user interface running on the computer. Designing interaction rather than interfaces means that our goal is to control the quality of the interaction between user and computer: user interfaces are the means, not the end.

In the natural sciences, analyzing, understanding and controlling a phenomenon requires *theory*. Unfortunately, HCI research is far from having solid (and falsifiable) theories of interaction. In particular, the situation illustrated in Figure 1 is overly simplified and naive, abstracting out everything but the user and the computer. In real-world interaction (Figure 2) users interact in an environment that includes physical artifacts and networks of diverse computers and whose physical, social, organizational and cultural characteristics affect their actions. I identify two levels for analyzing and designing interaction: *interaction paradigms* offer a high-level conception of interaction phenomena and *interaction models* are operational descriptions of how interaction proceeds.
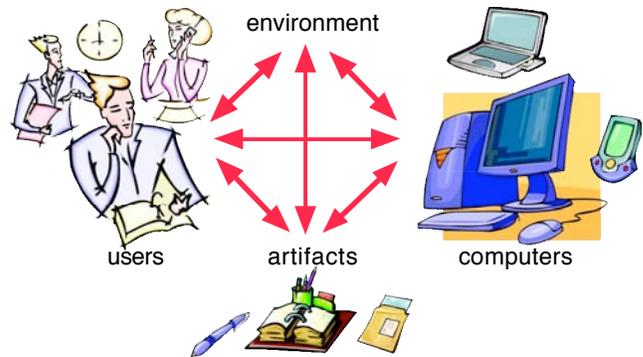

Figure 2. Interaction in the real world.

2.1 Interaction Paradigms

The three primary interaction paradigms are: *computer-as-tool*, *computer-as-partner*, and *computer-as-medium*. The computer-as-tool paradigm extends human capabilities through a (very sophisticated) tool, just as the invention of the wheel allowed us to transport heavy loads over long distances. Direct manipulation and WIMP interfaces fall into this category. The computer-as-partner paradigm embodies anthropomorphic means of communication in the computer, such as natural language, so that users can delegate tasks. Agent-based interaction and speech-based interfaces fall into this category. Finally the computer-as-medium paradigm uses the computer as a medium by which humans communicate with each other. Email, chat and videoconferencing fall into this category.

These paradigms are addressed separately by different research communities: Human-Computer Interaction focuses on computer-as-tool, Artificial Intelligence focuses on computer-as-partner and Computer-Supported Cooperative Work focuses on computer-as-medium. However, no paradigm can subsume the others and I believe that ultimately, all three paradigms must be integrated into a single vision. Note, too, that these paradigms rely on two sets of skills that distinguish humans from other species: the ability to create and use artifacts (computer-as-tool and computer-as-medium), and the ability to communicate with each other through language (computer-as-partner and computer-as-medium).

The rest of this paper focuses on the computer-as-tool paradigm. The unique challenge here is to create new tools that both augment and complement human capabilities. For example, Information Visualization [14] illustrates the benefits of complementing human capabilities: it combines the power of computers to create visualizations and human perceptual abilities to extract patterns. In the computer-as-tool paradigm, designing interaction requires understanding of both computer algorithms and human perception-and-action in order to harness the power of the user+computer ensemble.

2.2 Interaction Models

The purpose of an interaction model is to provide a framework for guiding designers, developers and even users (in the context of participatory design) to create interactive systems. An interaction model is thus more operational than an interaction paradigm and can be used directly by designers. Unlike ergonomic rules, which are often limited to post-hoc evaluation of a design, an interaction model is used from the early stages of the design and is therefore proactive.

Interaction models can be evaluated along three dimensions:

1) *descriptive power*: the ability to describe a significant range of existing interfaces;
2) *evaluative power*: the ability to help assess multiple design alternatives; and
3) *generative power*: the ability to help designers create new designs.

Generative power does not mean allowing the computer to automatically generate canned solutions, but rather helping human designers create richer and more varied design spaces from which to develop innovative solutions.

An interaction model can take many forms, from high-level design guidelines, such as the four principles of direct manipulation [30], to detailed rules such as those described in style guides, e.g. the Apple Human Interface Guidelines [3]. High-level models tend to have good descriptive power but poor evaluative and generative power. Low-level models tend to have poor descriptive and evaluative power, but higher generative power. A good interaction model must strike a balance between generality (for descriptive power), concreteness (for evaluative power) and openness (for generative power). Finally, the quality of the interaction model itself does not guarantee the quality of the resulting designs. As with programming languages, where one can write terrible programs with a good language, one can create terrible interfaces with a good model.

3. INSTRUMENTAL INTERACTION

Instrumental Interaction [4] is an interaction model that operationalizes the computer-as-tool paradigm. Building on direct manipulation [30], it introduces the notion of *instruments* as mediators between users and domain objects. It is inspired by our everyday experience of using tools, instruments and devices to operate on the physical world rather than using our bare hands. We often use secondary objects to extend our abilities and achieve the desired effect on a physical artifact. For example, undoing a screw requires a screwdriver, writing is easier with a pen, and swimming is more efficient with fins.

Figure 3 illustrates a simple navigation instrument: the scrollbar. This instrument is composed of the physical mouse and the on-screen scrollbar. The user interacts with it through direct action, and receives direct feedback from the mouse (touch) and the scrollbar (highlight). The scrollbar then translates user actions into scrolling commands for the document. The document provides two kinds of response: towards the instrument (updating the thumb of the scrollbar) and towards the user (scrolling the contents of the document). Interaction with the document is therefore mediated by the instrument.
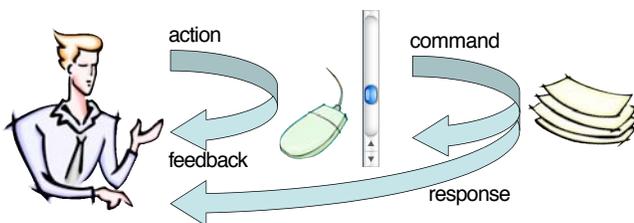

Figure 3. Instrumental interaction: the instrument (center) mediates interaction between user (left) and document (right)
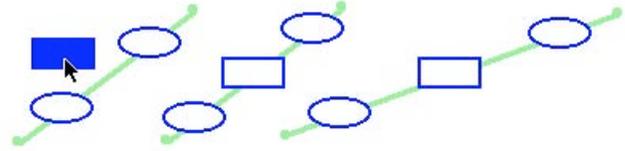

Figure 4. Distribution guideline: objects are re-distributed when an object (left) is added to the guideline (center) or when the guideline is reshaped (right).

Descriptive power - Instrumental interaction describes a wide spectrum of interaction techniques, from traditional GUIs to advanced techniques such as toolglasses [9], tangible interfaces [34] and augmented reality [37]. (See [4] for details).

Evaluative power - I have introduced three properties [4] to compare instruments with similar functions. The *degree of indirection* is the distance, in time or space, between the instrument and the object it operates on. The *degree of integration* measures degrees of freedom of the input device that are used by the instrument. The *degree of conformance* is the extent to which the user's action on the instrument is similar to the instrument's action on the object. Additional properties could be defined, e.g. in relation with the taxonomies introduced below.

Generative power - Thinking in terms of instruments helps designers focus on interaction and create more innovative design solutions. For example, we designed an alignment instrument for a graphical editor [5] that is more powerful than the traditional alignment dialog box. A *dialog box* answers the question: What interface should I build that specifies the parameters of the alignment function in my application? In contrast, an *alignment instrument* answers the question: How can I create and maintain alignments in a document? The first approach focuses on the interface, the second focuses on interaction. Our solution was the *magnetic guideline:* an instrument that can be inserted as a first-class object into a document. Graphical objects can then be attached to or detached from the guideline. Moving the guideline moves all the objects attached to it, maintaining the alignment. Figure 4 shows an extension which evenly distributes all objects that are attached to the guideline.

Both generative and evaluative power can be enhanced by creating instrument taxonomies. One taxonomy might concern the relationship between instruments and the objects they operate on: some instruments *create* objects, others *transform* them while others *add constraints*, such as the alignment instrument above. Still others *select* objects or *navigate* the object space and some even operate on other instruments, such as a tool palette with a set of tools. Another taxonomy might describe how the user operates the instrument: some instruments, like paint brushes, are held in the hand. Others, like scrollbars, sit on the screen. Still others, like magnetic guidelines, become domain objects in their own right. Taxonomies are useful for cataloguing existing techniques, identifying gaps and looking for possible new candidates.

3.1 Design principles

Another way to increase the generative power of an interaction model is to define *design principles*. For instrumental interaction, we have defined three design principles inspired by programming languages: reification, polymorphism and reuse [6].

Reification turns concepts and abstract commands into user interface objects that the user can manipulate directly. For example, a scrollbar is the reification of the concept of navigating a list or document. *Polymorphism* states that tools should operate in as many different contexts as possible, as long as this makes sense to the user. For example, copy and paste work across a wide range of object types, including text, drawing and sound. Finally *reuse* may involve either user input or system output. *Redo* and *macros* are examples of input reuse, enabling users to replay earlier commands in a new context. *Copy-paste* is an example of output reuse, enabling users to reuse results from earlier command sequences.

Building on the parallel with programming languages, other principles can be defined. For example *currying* (or partial application) might be used when a command requires multiple parameters, such as applying a color to an object. Currying the command means creating a family of instruments that apply a color (one per instrument) to an object, i.e. a set of color swatch tools. Side Views [33] is an example of a sophisticated use of currying: it displays a preview of the current command applied to the current object, and expands the preview when the command uses extra parameters.

Our extensive use of the first three principles in the CPN2000 project [5] led to a graphical application with no menus, no dialog boxes, no title bars, no scrollbars and no notion of selection, yet it was demonstrably more powerful and simpler to use than the earlier WIMP version. These principles are complementary to each other when designing instruments and have proven extremely effective as design tools.

3.2 From models to systems

Even though instrumental interaction helps designers think in terms of interaction rather than just interfaces, it only addresses one level of interaction design. Interaction can be analyzed at many levels and, to be successful, interaction design must take them all into account. The following sections focus on two levels: *situated interaction,* where the concern is to understand how the system can support the users' activities in the context of use, and interaction as a *sensory-motor phenomenon* where the concern is to match the interface with the users' capabilities. Next, I address how operationalizing the design of interaction requires appropriate tools and frameworks. Beyond user interface toolkits, we need proper *interaction architectures* that can support the creation and evolution of interactive systems.

4. SITUATED INTERACTION

Traditional HCI views interactive software in terms of user tasks, with the assumption that these systems will be used for specific purposes in specific ways. This approach emphasizes operational rather than creative tasks, yet the most interactive applications, e.g. graphical editors, are the least amenable to operational task descriptions. Increasingly, computers are used by knowledge workers and other creators who typically develop their own practices and patterns of use, despite using the same software on similar computers for similar projects. Even in cases where the computer supports operational tasks, users do not always play by the rules and adapt the system to their needs and work habits [22].
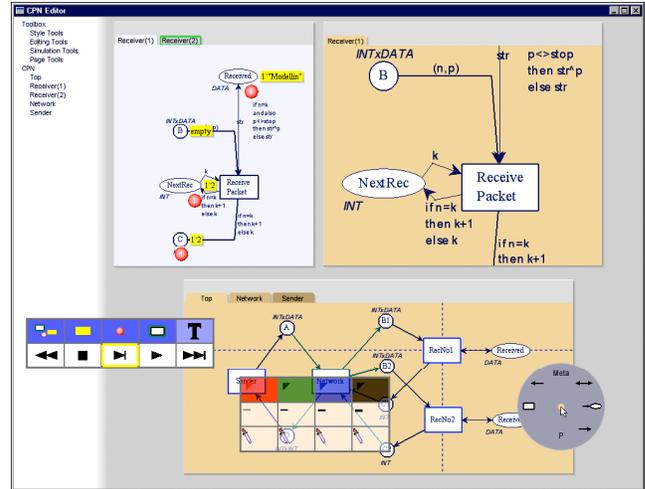

Figure 5. The CPN2000 interface, with multiple-page windows, a tool palette, a toolglass and a marking menu.

If we abandon the notion of task, on what grounds can we build better interactive systems? Suchman's concept of situated action [31] gives an anthropologist's account of how users interact with information technology in real-world settings. Her work shows how ethnography can help characterize the design problem. Usability testing on the other hand helps identify problems once a solution has been developed. But ethnography does not provide much help in choosing a design direction, and user testing does not provide much help in how to fix design problems. What is needed is a more generative approach, one that can create solutions or pieces of solutions that are then evaluated and improved upon.

Situated interaction is an approach I am developing with Wendy Mackay in the *In Situ* lab (http://insitu.lri.fr). The goal is to combine a range of techniques for designing interactive systems that are better adapted (and adaptable) to their context of use. It builds upon participatory design [18] by heavily involving end users throughout the design process. Generative techniques include well-known creativity techniques such as brainstorming and paper prototyping [7], and the heavy use of video [23]. We rely on interaction models and design principles to guide the design process and help reflect on the solution being created.

The CPN2000 project [5] illustrates this design approach with a graphical editor for Colored Petri Nets (CPN). The first key to the successful design of CPN2000 was to recognize, after detailed observational work, that different users work differently and that the same user applies different interaction patterns according to the context of use [24]. A CPN designer may be copying a design drawn on paper, creating it directly on-line, beautifying a diagram or fixing a bug uncovered during a simulation. Even though the elementary actions of creating and editing objects in the diagram may be the same, the different contexts of use lead to different interaction patterns. Some patterns are object-centered, i.e. applying multiple commands to the same object, while others are command-centered, i.e. applying the same command to different objects. As a result no single interaction technique works best in all contexts, and the best solution is to provide a range of interaction techniques and let users decide which to use.
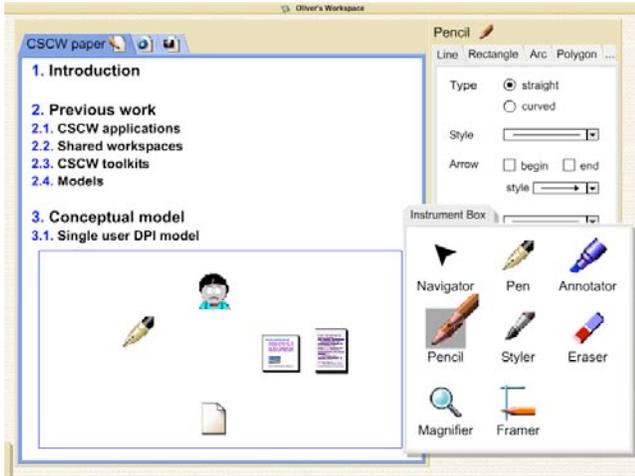
Figure 6. A mock-up of the DPI environment, with a set of editing instruments that can be used across documents.

Figure 7. Semantic pointing: a dialog box as seen on the screen (left) and as perceived in motor space (right). Enlarging the *Save* button in motor space facilitates its selection, while shrinking the *Don't Save* button in motor space makes it harder to select.

The second key to the design of CPN2000 was the holistic approach to interaction design. Rather than juxtaposing a set of interaction techniques and hoping for the best, we systematically studied how best to combine them, using instrumental interaction and the design principles.

A similar approach was taken by Beaudoux to design DPI (Document-Presentation-Instrument) [8], a document-centered system based on instrumental interaction (Figure 6). Here too, the design process focused on interaction as a whole, resulting in a system that completely decouples documents from the tools used to edit them and supports various types of sharing of document and tools.

I call systems such as CPN2000 and DPI *post-WIMP interfaces* because, even though they are graphical, direct manipulation interfaces, they are radically different from current systems. In particular, current systems intertwine the data being manipulated and the tools used to manipulate it while post-WIMP interaction separates them. This opens the door to interactive environments that are centered on documents rather than applications, as in the original Xerox Star [21].

By decoupling documents from the tools used to create, edit and navigate them, users gain a sense of freedom and control in organizing their electronic world: they can select the tools that fit them best. This approach would have a tremendous effect on the market for interactive software. Instead of using proprietary formats to lock their users away from the competition, software vendors would compete in a market of interaction instruments and document viewers. They would be compatible by necessity, like plug-ins for systems like Adobe Photoshop. Competition and openness would facilitate the transfer of novel interaction techniques from research to the market.

5. INTERACTION AS A SENSORY-MOTOR PHENOMENON

Interaction can be viewed as a sensory-motor phenomenon: the user acts on the system, which generates output perceived by the user. Since human sensory-motor capabilities are fairly limited and constant across users and over time, it is important to develop interactions that are adapted to these capabilities.

Although the psychology of perception and action has a vast literature, the advent of computers has created new challenges and opportunities. With computers, humans are no longer exposed to the physical world governed by the laws of physics, but to a synthetic world whose laws can be programmed at will. For example, overlapping windows with scrollable contents do not correspond to anything in the physical world, yet most users understand them easily. Pointing with a mouse is also unnatural because of the indirection between mouse and cursor and the non-linear control-display ratio, i.e. the mapping between mouse movement and cursor motion. Controlling interaction opens a wide design space not covered by traditional Psychology.

Fitts' law [16] is one of the rare examples of a robust empirical law in Psychology and has generated a significant amount of HCI research [25]. It is extremely relevant to graphical interaction because it models the movement time to acquire a target, i.e. the basic act of pointing. Fitts' law specifies that movement time MT is a simple function of the index of difficulty ID of the task, defined as the logarithm of the ratio of target distance D to target width W: $MT = a + b\ ID$ where a and b are determined empirically and $ID = \log_2(1+D/W)$.

Until recently, Fitts' law had been applied to graphical interaction for pointing tasks that were similar to pointing in the real world. Not surprisingly, performance was similar to that in the real world. Recently, several researchers have tried to "beat Fitts' law", i.e. to obtain pointing performance that is better than in the real world, or to see if the law applies to pointing tasks that are impossible in the real world. For example, McGuffin & Balakrishnan [26] show that when the target expands as the cursor approaches it, the performance is that of the expanded target size even if expansion occurs as late as 90% of the distance to target. The MacOS X *dock* uses expansion. Unfortunately it does not take advantage of this phenomenon because of the relative motion of its targets [38].

Blanch et al. [11] present *semantic pointing*: by manipulating the control-display ratio, the visual size of targets can be independent from their size in motor space (Figure 7). A visually-small target can be made easier to point to, for example, without the user even noticing the manipulation. A simple version of this decoupling occurs when targets are along the sides of the display. For example, the menu bar on the Macintosh is at the top of the screen, so targets have an infinite vertical size in motor space since the cursor cannot go further than the top. On Windows however, the menu bar is separated from the top of the screen by a thin border, so the height of the targets in the menu bar is the same in visual and motor space, resulting in a significantly higher index of difficulty than on the Macintosh. This shows how the details of even the simplest interaction technique (pointing) can be critical to its performance.

In joint work with Yves Guiard, we have pushed the limits of Fitts' law even further. Pointing with one's arm in physical space is limited to an index of difficulty of about 10 bits (1mm at a 1m distance). However on a computer, with the help of zooming, one can point to targets that are arbitrarily small at an arbitrarily-long distance. We experimented with indices of difficulty up to 30, i.e. a target of 1mm at a distance of 1000km, and found that Fitts' law still applied [19].

Fitts' law has proven to be an invaluable tool in studying interaction as a sensory-motor phenomenon. Its scope, however, is limited to a single type of action: pointing. Other laws exist that can be used to model a wider range of techniques, including the steering law [1] for moving the cursor through a constrained path and Hick's law [20] for visually searching a target. But more work is required to fully understand the sensory-motor phenomena of interaction. For example, information visualization relies on preattentive processing of visual information to extract patterns [35]. Our knowledge of preattentive processing is still limited and amounts to a set of rules rather than a unifying theory. Perception of sound is even less well understood [12], despite the obvious potential for reducing the load on the visual channel and conveying temporal information.

More generally, few theories in Psychology can be easily applied by interaction designers. This is not surprising, since HCI researchers generally draw from descriptive Psychological theories. (Exceptions, such as Behaviorism, are rarely used in interaction design). Gibson's ecological approach to perception [17] has been popularized in HCI by Don Norman through the (often misunderstood) notion of affordances [29]. Unfortunately it, too, is a descriptive theory that is hard to use by designers.

If we are to foster the design of better interaction, we need not only sets of techniques with proven effectiveness, but also rules and principles for combining them without losing their advantages. We are defining a model that predicts the cost of combining interaction techniques according to the context of use [2], moving up from techniques to interaction sequences. This is a first step toward a generative theory of interaction.

Studying interaction at the level of a sensory-motor phenomenon has led to important advances in HCI by providing a scientific basis to evaluate the performance of interaction techniques. However, the results must be taken with care since they are easy to misuse or overgeneralize. Controlled experiments abstract real situations in order to operationalize the phenomenon being observed. They allow measurement of the peak performance of a technique, which may differ significantly from its actual performance in context. This is why designing interaction must associate the holistic level of interaction in the context of use with the details of interaction as a sensory-motor phenomenon.

6. INTERACTION ARCHITECTURES

The last piece in promoting the design of interaction rather than interfaces is *interaction architectures*, i.e. tool and middleware support for creating interactive systems. Wegner argues that interactive systems are inherently more powerful than algorithmic systems [36]. Yet most interactive systems are implemented with traditional, algorithmic programming languages. This partially explains the high cost of developing and maintaining interactive software, and why the user interface represents 50% to 80% of an application [28].

Research in user interface toolkits and user interface management systems was active 10 years ago, but has almost disappeared. If we are to promote new interaction techniques and new interaction models, it is critical that they are embedded into toolkits that can be adopted by developers.

Virtually all of today's toolkits are based on an event-based model where input (and other) events are read by a top-level loop and dispatched to widgets through a callback mechanism. The resulting programming style breaks a conceptually linear interaction into a set of chunks called by the main loop independently from each other [27]. Programmers resort to tricks such as global variables and unsafe narrowing to share state between chunks (Figure 8). The result is brittle code that is hard to debug and hard to maintain.

Another problem is that almost all toolkits are based on the notion of a widget, i.e. a software component that encapsulates a presentation (how the widget is displayed), a behavior (how it reacts to user input) and an application interface (how it signals its state changes and how its state can be changed by the application). Widgets work well for simple interaction objects such as buttons or scrollbars. They work less well for dialog boxes and inspector windows and do not work at all for more direct manipulation techniques such as drag and drop or toolglasses [9]. We need to break away from the widget model, which results in "boxy" interfaces that do not exploit direct manipulation and instrumental techniques.

```
segment s;    // global variable
bool drawing; // global variable
void HandleButtonPress (point p) {
    s.p1 = s.p2 = p; drawing = false;
}
void HandleMouseMove (point p) {
    if (! drawing && distance (s.p1, p) > 3) {
        drawing = true; s.p2 = p; Draw (s);
    } else {
        Erase (s); s.p2 = p; Draw (s);
    }
}
void HandleButtonRelease () {
    if (drawing) {
        Erase (s);
        AppNewSegment (s); // notify application
    }
}
```

Figure 8. Event-driven programming of rubber-band interaction (event-loop omitted).

```
interaction RubberBand {
    segment s;
    state start {
        when ButtonPress(p) {
            s.p1 = s.p2 = p;
        } -> waitMove
    }
    state waitMove {
        when MouseMove(p)
          && distance(s.p1, p) > 3 {
            s.p2 = p; Draw(s);
        } -> track
        when ButtonRelease -> start
    }
    state track {
        when MouseMove(p) {
            Erase(s); s.p2 = p; Draw(s);
        }
        when ButtonRelease {
            Erase(s); App.NewSegment(s);
        } -> start
    }
}
```

Figure 9. Interaction machine for rubber-band.

I propose making interactions (not widgets) the first-class objects of a toolkit. Figure 9 shows a simple example of an interaction machine for rubber-banding (compare with Figure 8). One of the benefits of this approach is the ability for multiple interactions to run in parallel. For example, with bimanual input, the left hand can pan and zoom a document while the right hand edits the document, as in CPN2000 [5]. Or multiple users can work on the same screen with single-display groupware [32]. Such interactions are likely to become more common as we move towards pervasive computing and its multiplicity of devices. Another advantage of reifying interactions is the ability to externalize them, as in the ICON system [15]: interactions can be loaded and unloaded at run-time, like plug-ins. This allows both developers and end-users an unprecedented level of control over their interaction with the system. We have started to experiment with programming interactions. Interactions are described by hierarchical state machines [10], are independent from the objects they operate on, and can be loaded dynamically.

Interactive systems are by definition open: they interact with the user (or users) and often with other programs. They must therefore adapt to various contexts of use, both on the users side and on the computer side. In order to address the challenges described at the beginning of this paper, I believe it is critical that we define interaction architectures that give more control to end users, that are more resistant to changes in the environment, and that scale well. I call these three properties *reinterpretability*, *resilience* and *scalability*.

Reinterpretability is the ability of a system to be used in contexts and by users it was *not* designed for. Like any other technology, computer systems are inevitably reinterpreted by their users. Users not only passively adapt to new technology but also adapt and appropriate it for their own needs [22]. Current systems are not flexible enough to support, even less encourage, such co-adaptation. One way to look at co-adaptation is that the development of an interactive system continues after it has been released, in the hands of users. Reinterpretability requires that users be able to redefine functions, change input/output devices, add or remove interaction techniques, maybe even program their own functions. Interaction as first-class object is a step in this direction, but it must be complemented by other mechanisms, such as those studied in the context of End-User Development (see, e.g. http://giove.cnuce.cnr.it/EUD-NET).

Resilience is the ability of a system to resist to change. This might seem contradictory with reinterpretability, but in fact is a condition for it: in order to withstand changes and reinterpretation, critical components of the architecture must resist changes in their environment. Web browsers are a good example: they are very tolerant of incorrect syntax in the HTML documents they display. Almost any subset of an HTML page is HTML, i.e. understandable by a browser. This encourages users to compose new pages by editing existing pages or page fragments, and has been a major factor in the development of the web. To date, there is no general approach for creating resilient interactive systems, especially since it is difficult to automate testing of interactive software.

Scalability is the ability of a system to withstand scale effects. How does a file manager perform with a million files? A text editor with a thousand page document? A shared editor with one hundred users? Given the exponential growth of on-line data, scale effects cannot be ignored when designing an interactive system. Maintaining interactive response time

under 0.1s under any circumstance means that performance should be constant with data size. Strategies such as caching, level-of-detail and multi-threading should be employed to guarantee response time and built into the architecture. Until this is achieved, we will continue to observe that despite the growth in computational power of our computers, the performance as perceived by end users degrades quickly as the number or size of documents grows.

In summary, more work is needed on software tools to create better infrastructures for interactive systems. This not only means software toolkits and middleware, but also programming languages and operating system support.

7. CONCLUSION

Despite the tremendous increase in computing power of personal computers and the large number of novel interaction techniques published in the literature, today's commercial user interfaces perform poorly. They face a three-fold challenge: an increasing amount of data, number of devices and diversity of users. I have argued that HCI research needs to shift its focus from individual point designs to full-fledged interactive environments and their supporting methods and tools.

I have shown how interaction models and design principles can help create better interactive systems. I have argued that design should focus both on the higher level of context of use and the lower level of sensory-motor phenomenon, and that implementation should be supported by robust interaction architectures. These are of course only the first steps toward a generative "theory" of interaction.

A paradox of HCI is that the best interface designs are invisible: they are not noticed by users, who find the interaction "natural", and once invented, they seem obvious to other designers. Our measure of success is therefore elusive: making systems that disappear, creating the invisible.

HCI is not the science of user interfaces, just as astronomy is not the science of telescopes. HCI needs interfaces to create interaction, and we should focus on describing, evaluating and generating interaction, not interfaces.

8. ACKNOWLEDGEMENTS

9. REFERENCES

1. Accot, J. and Zhai, S. (1997). Beyond Fitts' law: Models for trajectory-based HCI tasks. *Proc. ACM Human Factors in Computing Systems, CHI'97*. ACM Press. pp. 295-302.

2. Appert, C., Beaudouin-Lafon, M. & Mackay, W.E. (2003). Context matters: Evaluating Interaction Techniques with the CIS Model. *Technical report 1372, Laboratoire de Recherche en Informatique*, Université Paris-Sud, France.

3. Apple Computer, Inc (1987). *Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley.

4. Beaudouin-Lafon, M. (2000). Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. *Proc. ACM Human Factors in Computing Systems, CHI'2000*. CHI Letters 2(1):446-453. ACM Press.

5. Beaudouin-Lafon, M. and Lassen, M. (2000). The Architecture and Implementation of a Post-WIMP

Graphical Application. *Proc. ACM Symposium on User Interface Software and Technology, UIST '00*. CHI Letters 2(2):181-190. ACM Press.

6. Beaudouin-Lafon, M. and Mackay, W. (2000). Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. *Proc. Conference on Advanced Visual Interfaces, AVI 2000*. ACM Press. pp.102-109.

7. Beaudouin-Lafon, M. & Mackay, W.E. (2002). Prototyping Methods and Tools. In Jacko, J.A. & Sears, A. (eds) *The Human-Computer Interaction Handbook*. Lawrence-Erlbaum Associates. pp. 1006-1031.

8. Beaudoux, O. & Beaudouin-Lafon, M. (2001). DPI: A Conceptual Model Based on Documents and Interaction Instruments. *People and Computer XV - Interaction without frontier* (Joint proceedings of HCI 2001 and IHM 2001). Spring Verlag. pp. 247-263.

9. Bier, E., Stone, M., Pier, K., Buxton, W., De Rose, T. (1993) Toolglass and Magic Lenses : the See-Through Interface. *Proc. ACM SIGGRAPH.* ACM Press. pp. 73-80.

10. Blanch, R. (2002). Programmer I'interaction avec des machines à états hiérarchiques. *Proc. Conférence Francophone sur l'Interaction Homme-Machine, IHM '01*, ACM International Conference Series, pp 129-136.

11. Blanch, R., Guiard, Y., & Beaudouin-Lafon, M. (in press). Semantic pointing: Improving target acquisition with control-display ratio adaptation. *Proc. ACM Human Factors in Computing Systems, CHI '04*. ACM Press.

12. Bregman, A.S. (1990). *Auditory Scene Analysis: The Perceptual Organization of Sound*. MIT Press.

13. Callahan, J., Hopkins, D., Weiser, M. & Shneiderman, B. (1988) A Comparative Analysis of Pie Menu Performance, *Proc. ACM Human Factors in Computing Systems, CHI'88*. ACM Press. pp. 95-100.

14. Card, S.K, Mackinlay, J.D. & Shneiderman, B. (1999). *Readings in Information Visualization: Using Vision to Think*. Morgan-Kauffman.

15. Dragicevic, P. & Fekete J. D. (2001). Input Device Selection and Interaction Configuration with ICON. *People and Computer XV - Interaction without frontier* (Joint proceedings of HCI 2001 and IHM 2001). Spring Verlag. pp. 543-558.

16. Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology, 47*, 381-391.

17. Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin.

18. Greenbaum, J. and Kyng, M., eds (1991) *Design at Work: Cooperative Design of Computer Systems*. Hillsdale NJ: Lawrence Erlbaum Associates.

19. Guiard, Y., Bourgeois, F., Mottet, D., & Beaudouin-Lafon, M. (2001). Beyond the 10-bit barrier : Fitts' law in multi-scale electronic worlds. *People and Computer XV - Interaction without frontier* (Joint proceedings of HCI 2001 and IHM 2001). Spring Verlag. pp. 573-587

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

21. Johnson, J. et al (1989). The Xerox "Star": A Retrospective. *IEEE Computer*, 22(9):11-26.

22. Mackay, W.E. (1990). *Users and Customizable Software: A Co-Adaptive Phenomenon*. Ph.D. Thesis. Massachusetts Institute of Technology.

23. Mackay, W.E., Ratzer, A., and Janecek, P. (2000) Video artifacts for design: Bridging the gap between abstraction and detail. *Proc. ACM Designing Interactive Systems, DIS'00.* ACM Press. pp. 72-82.

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

25. MacKenzie, I.S. (1992). Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction, 7*, 91-139.

26. McGuffin, M. and Balakrishnan, R. (2002) Acquisition of Expanding Targets. *Proc. ACM Human Factors in Computing Systems, CHI '02*. CHI Letters 4(1):57-64. ACM Press.

27. Myers, B.A. (1991). Separating application code from toolkits: Eliminating the spaghetti of call-backs. *Proc. ACM Symposium on User Interface Software and Technology, UIST '91*. ACM Press. pp. 211-220.

28. Myers, B.A. and Rosson, M.B. (1992). Survey on user interface programming. *Proc. ACM Conference on Human Factors in Computing Systems, CHI'92*. ACM Press. pp. 195-202

29. Norman, D.A. (1999). Affordances, Conventions and Design. *Interactions* 6(3):38-43. ACM Press.

30. Shneiderman, B. (1983). Direct Manipulation, a Step Beyond Programming Languages. *IEEE Computer*, 16(8):57-69, August 1983.

31. Suchman, L.A. (1987). *Plans and Situated Actions*. Cambridge University Press.

32. Stewart, J., Bederson, B., & Druin, A. (1999). Single Display Groupware: A model for co-present collaboration. *Proc. ACM Human Factors in Computing Systems, CHI '99*. ACM Press. pp. 286-293.

33. Terry, M. & Mynatt, E.D. (2001). Side Views : Persistent, On-Demand Previews for Open-Ended Tasks. *Proc. ACM Symposium on User Interface Software and Technology, UIST 2001*, CHI Letters 3(2):71-80, ACM Press.

34. Ullmer, B. & Ishii, H. (2001). Emerging Frameworks for Tangible User Interfaces. In Carrol, J.M. (ed.) *Human-Computer Interaction in the New Millenium*. Addison-Wesley. pp. 579-601.

35. Ware, C. (1999). *Information Visualization, Perception for Design*. Morgan Kaufmann.

36. Wegner, P. (1997). Why Interaction is More Powerful Than Algorithms. *Comm. ACM* 40(5):80-91. ACM Press.

37. Wellner, P., Gold, R. & Mackay, W. (1993) Special issue on computer-augmented environments, *Communications of the ACM* , 36(7), July 1993. ACM Press.

38. Zhai, S., Conversy, S., Beaudouin-Lafon, M., Guiard, Y. (2003). Human On-Line Response to Target Expansion. *Proc. ACM Human Factors in Computing Systems, CHI 03*. CHI Letters 5(1):105-112. ACM Press.