Support for Input Adaptability in the ICON Toolkit

Pierre Dragicevic
LIIHS/IRIT
Université Paul Sabatier
F-31062 Toulouse Cedex, France
Tel: (+33) 561-556-965
dragice@irit.fr

Jean-Daniel Fekete
INRIA Futurs/LRI
Bat. 490, Université Paris-Sud
F91405 Orsay Cedex, France
Tel: (+33) 169-156-623
Jean-Daniel.Fekete@inria.fr

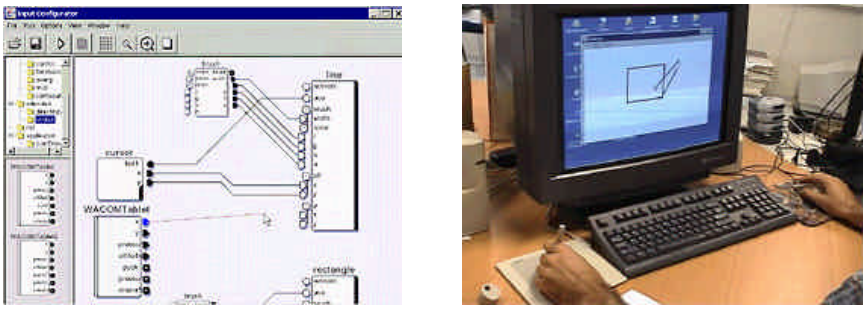

Figure 1: Configuring a Drawing Application for Bimanual Interaction using ICON

ABSTRACT
In this paper, we introduce input adaptability as the ability of an application to exploit alternative sets of input devices effectively and offer users a way of adapting input interaction to suit their needs. We explain why input adaptability must be seriously considered today and show how it is poorly supported by current systems, applications and tools. We then describe ICON (Input Configurator), an input toolkit that allows interactive applications to achieve a high level of input adaptability. We present the software architecture behind ICON then the toolkit itself, and give several examples of non-standard interaction techniques that are easy to build and modify using ICON's graphical editor while being hard or impossible to support using regular GUI toolkits.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *user interface management systems, input devices and strategies, prototyping, interaction styles*,

General Terms
Design.

Keywords
Interaction techniques, Input devices, Visual programming, Toolkits, Adaptability.

1. INTRODUCTION
For years, using a computer was all about typing on a keyboard and manipulating a mouse to select or drag objects on the screen. Today, we are facing radical changes in the computer industry that seriously throw back those standards into question.

A key evolution is the advent of mobile computing and the proliferation of heterogonous platforms such as palmtops, laptops, wearables, tablet PCs and smart phones. People are discovering new ways of interacting, and techniques such as gesture interaction or speech recognition are now becoming popular. Meanwhile, input hardware on traditional desktop computers is becoming more and more rich and complex: mice, keyboards and joysticks are continuously being augmented with new controls (see [29] for example), whereas alternative devices such as web cams, voice recognition microphones, tablets, and dedicated gaming devices are being more and more affordable and fashionable. One reason is that tasks performed on desktop computers, including web browsing, music listening or watching movies, have become extremely varied. The population of desktop computer users has also gained in experience, and their demands evolved from simplicity and ease of use to efficiency and speed.

Unfortunately, available applications and tools currently put a brake on the evolution of input interaction: they are still designed for a stereotyped set of standard input devices and interaction techniques, and are far from being able to adapt to a high diversity of input peripherals, interaction styles and platforms.

In this article, we introduce input adaptability as the ability of an interactive application to exploit alternative input devices effectively and offer users a way of adapting input interaction to suit their needs. We explain why input adaptability must be seriously considered and survey the different levels of support for input adaptability in today's systems. We then describe ICON (Input Configurator), a novel system that addresses main input

adaptability issues by making interactive applications fully input-configurable. We detail the underlying data-flow architecture behind ICON and we show how ICON configurations can be adapted to handle missing input devices or to support multiple input devices and advanced interaction techniques.

2. INPUT ADAPTABILITY

In this section, we define our terminology and explain the issues related to input adaptability.

2.1 Matching Input with the Context

An *interaction technique* is a way of using a physical input device to accomplish a given task [17]: moving a cursor, clicking on buttons, dragging objects are interaction techniques commonly associated with pointing devices. We call *input technique* the combination of *physical* input devices with an interaction technique that describes the way they are *logically* used. The numerous input techniques described in the HCI literature clearly demonstrate that the quality of user interfaces strongly depends on the choice of appropriate input techniques according to application's tasks, users and working environment [27]:

Application. Input techniques are strongly related to tasks. Buxton gave a good idea of how subtle the notion of device/task compatibility may be [11]. He showed how each input device, by its intrinsic properties, is unique and appropriate only for a limited set of tasks, making it difficult to categorize devices into classes of equivalence [11]. Task-dedicated input devices are widely used in geometric modelling and video games, and can even be part of the application design [25].

Users. "Designing for the user" is the motto of the HCI community. In particular, taking user's abilities and skills into consideration can dramatically improve interaction. Prolific work in the field of "informal interaction" describe interaction techniques and input devices that take advantage of ordinary skills such as handwriting or speech, for achieving a more natural interaction [5],[9]. Other work describe input techniques designed for special skills such as tape drawing [3] or puppet animation [25]. Common skills can also include, for example, driving a car or playing an instrument. On the other side, taking user's disabilities into account is even more important while designing input techniques. Significant work in the field of accessibility describe various input techniques for making computers usable by people with special needs [32],[12]. But again, no user taxonomy can capture all individual differences, and taking user preferences into consideration is also a central concern [36].

Working environments. Just like users, working environments can set important constraints or offer high potential that must be taken into account. Large desktops allow the use of multiple devices and controllers, and alternative devices such as microphones, web cams, tablets, or dedicated gaming devices are increasingly available. Conversely, noise, limited footprint, or mobility are characteristics of the environment that restrict the set of usable input techniques. Tablet PCs and PDAs, for example, make extensive use of pen-based interaction techniques, while new techniques are regularly being proposed [37], [34].

2.2 Limitations of Standard Input Techniques

As opposed to previous examples, most real-world applications follow the "standard input techniques" approach. On desktop computers for example, applications are essentially controlled using a mouse, a keyboard, and a fixed set of interaction techniques such as clicking, dragging, or text input. Those "WIMP" input techniques provide cross-application reusability and consistency, and they can be carefully designed and tested. However, this model has serious shortcomings in terms of input adaptability:

Standard input techniques are never fully appropriate. Standard input techniques target the average user, an average environment, and a set of average tasks (i.e. controlling standard widgets, such as clicking on a button or selecting a menu item). Because those techniques are too generic, they are never fully appropriate for the context and the application tasks [11].

Standard input techniques can be fully inappropriate. Standard input techniques can become totally inappropriate, for example when one or both standard devices are not available. On alternative platforms like mobile computing, interaction has been redesigned, but applications are still developed on top of a new set of generic input techniques. Although it is important for users and software companies that the same application runs on the most popular platforms, "standard input techniques" approaches do not address cross-platform portability.

2.3 Defining Input Adaptability

For a better accessibility and portability, it is critical for interactive applications to be able to exploit modified sets of input devices. For optimised usability, they must make use of appropriate interaction techniques, according to the available devices but also to the specific application tasks and the end user preferences. In this paper these input-related issues will be denoted by the single term of *input adaptability*. Note that there already exists a large terminological literature on adaptability in HCI (see [45] for example). With respect to this body of work, we aim at providing a simple and operational term for making our position clearer, rather than trying to establish a new comprehensive definition. We therefore introduce *input adaptability* as the simple combination of controllability, accessibility and configurability:

1. Controllability is the application's ability to use *enriched input* or to use standard input *more effectively*. Enriched input can range from additional controls on standard devices to multi-dimensional, high-bandwidth devices, and multiple devices and modalities. Effective use of rich input implies using interaction techniques making smart use of the full input bandwidth and device/task compatibilities. Standard input devices can also be better exploited by new interaction techniques such as gesture recognition.

2. Accessibility is the application's ability to use *impoverished input*. This can range from a missing standard device to very low-bandwidth input such as a single button. Supporting impoverished input effectively implies using richer interaction techniques to compensate for the decreased bandwidth and device/task compatibility.

3. Configurability expresses user's ability to freely *choose* how to use his input devices for controlling the application. This ranges from device selection and basic action mapping customisation to specification of rich interaction techniques. Ease of specification also plays a crucial role in input configurability.

2.4 Today's Support for Input Adaptability

In this section we describe the standard level of input adaptability provided by operating systems and WIMP graphical toolkits. We then explain how some applications try to achieve a better level of input adaptability by hacking this standard interaction model.

2.4.1 The Standard Level of Input adaptability

Most conventional desktop applications exclusively rely on traditional WIMP input systems. We describe the minimum level of input adaptability they are able to achieve.

Controllability. Traditional WIMP input systems make use of data coming from standard devices but ignore any additional input bandwidth provided by other devices. Drivers for alternative devices sometimes allow some generic control by sending keyboard keys or "pushing" the system cursor. But this compatibility-based control implies ignoring specific capabilities of the device, e.g. when a pressure sensitive tablet is used as a mouse, its high resolution and pressure information are not used.

Accessibility. Keyboard shortcuts from window managers and toolkits provide some redundancy that can sometimes replace a missing mouse. Windows accessibility allows mouse cursor control with keyboard, and provides alternative keyboard techniques for physically-challenged users. However, because input services from the system to the toolkit have all been built with standard input techniques in mind, there is no well-designed accessibility interaction technique. At best, missing input devices dramatically degrade the usability of applications.

Configurability. Device drivers allow minimal configuration of input hardware and low-level input handling, like mouse cursor acceleration or double-click speed. At higher levels, minimal input customisation is sometimes available through form-filling dialogs, often for configuring keyboard shortcuts. However WIMP systems do not allow rich configurability, because input handling is split up among several monolithic black boxes.

2.4.2 Enhanced Input Adaptability

Some applications provide a better controllability by accessing alternative input devices at a low level. We describe them in this section, as well as the tools and strategies commonly used for enhancing applications' accessibility and extensibility.

Musical and graphical design applications. Several semi-professional applications make use of non-standard devices by accessing them at a low level. Most applications for musical composition can read data from musical instruments through the MIDI protocol. Adobe Photoshop [1] supports tablet devices and exploits accurate positional values of the stylus for drawing; additional dimensions such as pressure or tilt can be assigned to any brush attribute through a dialog box.

Computer games. Computer games support increasingly sophisticated gaming devices such as joysticks with multiple controls or simulation-oriented devices. They traditionally give the user choice between keyboard, mouse or joystick to control the game. Most of them also provide free assignment of two-state controls – such as buttons or keys – to actions. Configurability of continuous channels is more limited, and the user is at best allowed to play with a few parameters such as sensitivity or Y axis inversion. DirectInput 8 adds built-in configurability with a standard but nonetheless limited input configuration dialog [30].

Hardware and Software Accessibility. Some add-on tools provide generic and sometimes configurable control of existing applications with alternative devices, usually for accessibility purposes. Accessibility can be achieved by hardware emulation, software emulation or by support inside specific applications. Some O.S. and toolkits such as Java/Swing allow external applications to access the internal states of the application's widgets and control them from the outside. But this approach leads to poor interfaces since interaction techniques provided by the components are not suited to alternative modes of control.

Scripting languages. Some applications use scripting languages to provide extensibility such as for configuring keyboard actions. With Quake3's scripting language, elaborate action bindings can be built, that use conditional action triggering, sequential actions, and dynamic reassignments [20]. New interaction techniques are sometimes implemented through scripting, e.g. the ViaVoice voice recognition system is integrated with Microsoft Word via scripts. But scripts are mainly designed for creating new functions, not new interactions.

2.4.3 Systems and Tools from Research

Compared to the prolific work on information visualization and advanced graphical effects, input has been little studied [11]. Some contributions, however, have made a step towards input adaptability, although none of them have addressed all of its aspects. Most significant work include advanced graphical toolkits and interactive behaviour editors.

Post-WIMP toolkits. Graphical toolkits such as Garnet/Amulet [33] and Artkit/Subarctic [41] describe standard event handling in a cleaner and more extensible way than traditional toolkits and support techniques such as gesture recognition. However, they are aware of a limited set of input devices and require important modifications to handle any new interaction paradigm. Post-WIMP toolkits are specialized in specific interaction paradigms such as gesture recognition [20], ambiguity solving through mediation [28], multi-pointer interaction [8], [21], context awareness [38] or augmented virtuality [18]. They allow a better controllability using standard or non-standard devices, but still support limited sets of input devices and use them in *ad-hoc* ways.

3D Toolkits. 3D toolkits such as World-Toolkit [39] or Avid/Softimage [2] support alternative input devices, mainly Space-Mice and 3D trackers. Their channel-based models allow free assignation of device dimensions to attributes of 3D objects, allowing for rich and configurable interaction. However, more elaborate input techniques like speech or gesture control are not supported and toolkit components can not be controlled with these devices.

Behaviour editors. Constraint-based editors such as Thinglab [10] or Fabrik [23] allow building some parts of interactive applications graphically, mainly for describing geometrical layout behaviours. Control-flow approaches such as ICO/PetShop [4] use Petri Nets or State-Transition Diagrams to describe control-intensive, highly modal parts of interactive applications. Data-flow-based editors have been widely used but their application to interaction specification has been rarely exploited outside the area of 3D authoring and animation. Virtools Dev [42], for example, uses a graphical data-flow editor for specifying 3D input. Jacob's VRED system [24] uses an hybrid control/data-flow editor to describe discrete and continuous aspects of 3D interaction. The

data-flow approach has proved to be quite promising for describing interactions with multiple devices. But as far as we know, the only attempt to use it for describing 2D interaction has been made by Whizz'Ed [15]. This notation has been successfully used to specify animation and bimanual interaction, though other techniques and input devices have not been investigated.

3. THE ICON INPUT TOOLKIT

ICON (Input Configurator) is a novel system for designing input-reconfigurable interactive applications, based on a reactive data-flow architecture that describes input techniques using interconnected modules. In this section, we give an overview of the concepts behind ICON and then describe the tool itself. We finally show how, by relying on ICON, applications can reach a high level of input adaptability.

3.1 Overview of ICON Components

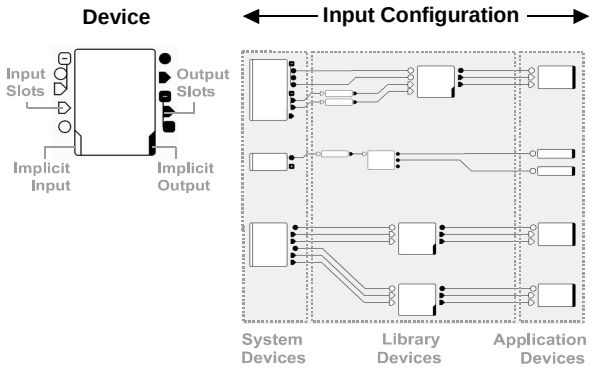

Figure 2: ICON components.

Devices and slots. ICON's model is essentially based on *devices*, which are a broad generalization of input devices: ICON's devices can produce output values, but can also receive input values. Figure 2 shows on the left the graphical representation of a device. A device contains typed channels called *input slots* and *output slots*, as well as parameters similar to JavaBeans accessors [14] to configure them. Slot types belong to a small set of basic types and the Object type, and each type has a distinct graphical representation (e.g. circle for Booleans, triangle for integers). Slots can be hierarchically grouped to form structured types (see Figure 2).

Implicit I/O. Non-deterministic devices are described as having *implicit input* (see Figure 2), i.e. additional source of information not fully described by its set of input slots. Examples of such devices include devices which are producing data on their own (physical input devices) or asynchronous devices which are temporally non-deterministic. Similarly, devices having *implicit output* (see Figure 2) produce alternative effects in addition to simply putting values on the output slots. Examples are devices manipulating external objects or producing user feedback.

Connections. An input slot of a device can be linked to one or several compatible output slots of other devices by *connections*, which are represented by wires (see Figure 2).

Device Folders. There are three main categories of devices: *System devices* describe system resources such as input peripherals; *Library devices* are system-independent utility devices such as processing devices and adapters; *Application devices* are devices that control an application. An instance of each device is available in a container which is hierarchically organized into *device folders*. Devices are copied from device folders in order to be used, just like prototype-based languages.

Input configurations. An *input configuration* is defined by a set of system and application devices, and a set of library devices and connections which map the system devices to the application devices (see Figure 2). ICON is modular, and subparts of an input configuration can be encapsulated into compound devices. For example, an input device and a feedback device can be connected then grouped to form a compound device having both external input and external output.

Execution. Whereas the contract of a device is simply to update its output slots every time it is asked to, ICON's execution model describes which devices must be triggered and when, and how values are propagated to other devices. The propagation mechanisms used are very effective and use simple data structures. ICON's execution model builds upon the semantics of reactive synchronous languages such as Esterel [7] or Lustre [19], which are used to describe systems that continuously react to the environment.

3.2 Interaction Techniques as Configurations

From our point of view, implementations of interaction techniques are essentially *interpretation* and *feedback* mechanisms. User's actions (or input device data) are *interpreted* to determine how they will modify the state of the application. In some cases the system must provide *additional feedback* on the way it interprets data.

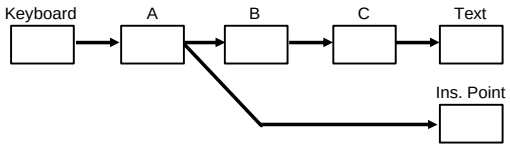

Figure 3: Interpreting keyboard actions.

Figure 3 shows the interpretation of keyboard actions in a text editing context (slots are not represented). A "key press" action is transformed into a raw key code through device A, then into a localized symbol trough device B. Device C generates strings from combinations of symbols. Those strings are finally used by the text editor. Besides encapsulating a processing function, each device provides its own abstraction of the keyboard. For example, a text component expects to receive strings but the insertion point is best controlled with a lower-level keyboard.

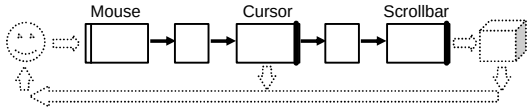

Figure 4: Scrolling through a document.

ICON describes feedback with implicit input and output. Figure 4 gives an example of scrolling through a document, and shows the feedback loop through implicit I/O. The Mouse device receives implicit input from the user, the Cursor device produces immediate feedback towards this user and the Scrollbar tells the application to update its document view.

3.3 The Toolkit

The ICON toolkit contains an extensible set of system devices and library devices for building input configurations, and provides a reactive machine for executing them. Input configurations run quite fast, with a negligible propagation time compared to useful processing code. ICON is written in Java, and uses native libraries for managing input devices.

3.3.1 ICON Devices

System devices. ICON toolkit's system devices provide low-level access to standard and alternative input devices. Under Microsoft Windows operating systems, it currently supports multiple mice through the CPNMouse library [43], multiple graphical tablets through Wintab API [26], multiple gaming devices and 6DOF controllers through DirectInput [30], speech recognizers through IBM JavaSpeech [40] and MIDI musical instruments and controllers. Under X Windows platforms, it provides access to XInput Extension [16] devices.

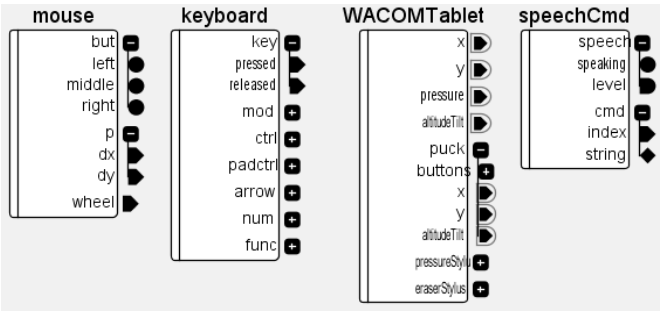

Figure 5: Mouse, keyboard, tablet and speech input devices.

Figure 5 shows some input devices as they appear in ICON: the low-level mouse sends screen-independent delta values; The keyboard sends key codes as well as state changes for individual special keys such as arrow keys or modifiers ; The graphical tablet sends positional, pressure and tilt values as well as specific information for each individual pointer (i.e. stylus tip, stylus eraser and puck) ; The speech command sends recognized strings or their index according to the vocabulary specified in the device's property box.

System output devices are also available, such as Midi devices for playing music on soundcards, and speech synthesis devices using IBM text-to-speech engines [40].

Library devices. The ICON toolkit has a set of built-in utility devices including mathematical and Boolean operators, signal processing devices, type adapters, and devices for conditional control and dispatch. It also provides a set of graphical feedback devices such as cursors and semi-transparent components, which support overlay animation on top of Swing frames.

Toolkit devices. The Swing device folder contains *widget devices* for controlling existing Swing applications that have no knowledge of ICON. The *JComponent* device provides generic control of Swing widgets. It can send events to widgets and modify their focused, enabled and visible states. Specific widget devices allow moving scrollbars programmatically or sending strings and caret commands to text components. All widget devices contain an input slot which specifies the component they operate on. Picking and Focus strategies are externalised and supported by two devices. ICON also provides experimental support for the Jazz toolkit [6] using a similar interaction model.

3.3.2 Implementing New Devices

Adding utility devices. ICON's library can be enriched with additional devices to suit the programmer's needs. Writing a new processing device is quite straightforward. It mainly consists in declaring device's slots and implementing an *update* method. This method basically checks which input slots have received a signal, retrieve their values and puts new values on output slots.

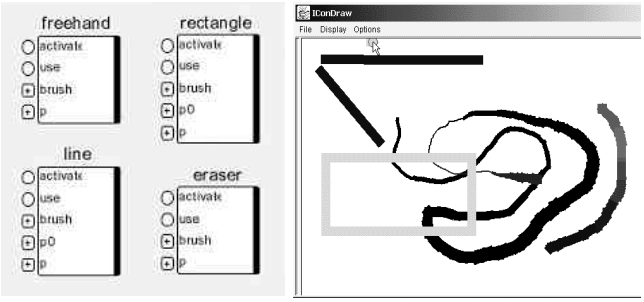

Figure 6: ICONDraw and its *application devices*.

Declaring application devices. Java developers can enhance ICON controllability of their application by implementing specific application devices. Figure 6 shows a sample application called ICONDraw that has been written to be entirely configurable with ICON. Instead of listening to system events, this application simply describes the way it expects to be controlled by externalising a set of drawing tools. Each tool basically receives activation signals, brush attributes, and one or two positions. The code for an application device is similar to that of a utility device, except that the update method makes application calls instead of setting output values.

3.4 Building and Editing Input Configurations

ICON configurations can be built and modified using a graphical editor. An early prototype of this editor has been described in [44]. In this section, we give an overview of a subset of interaction techniques that can be currently built using ICON.

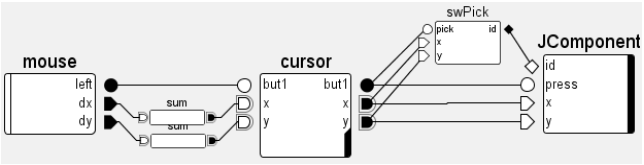

Figure 7: Standard positional control of swing components.

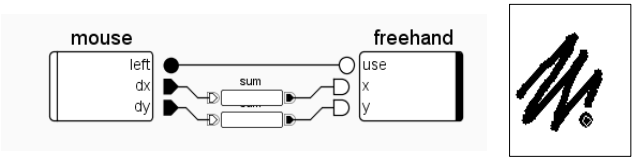

Figure 8: Mouse control of the Freehand device

3.4.1 Describing Standard Input Techniques

Standard input techniques can be built and shipped with applications as default input configurations. Figure 11 shows a configuration which reproduces basic positional control of Swing components (only used slots are displayed). Relative positional values of the low-level mouse are transformed into absolute values which move a cursor, and sends mouse events to the Swing component that has been picked. Similar mouse and keyboard-based control

can be built for specific application devices: Figure 8 shows basic control of the freehand tool with a mouse. To obtain a decent default configuration, this construction can be extended to handle additional cursor feedback, brush attributes control, and multi-tool control through a mode.

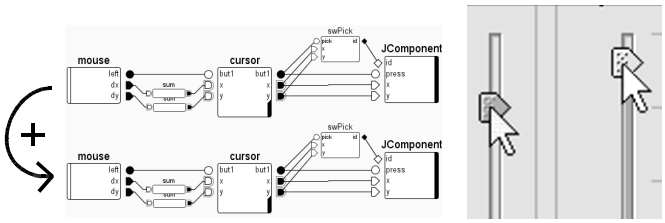

Figure 9: multi-pointer Interaction with Swing.

3.4.2 Adding Pointing Devices

Because each cursor device manages its own feedback, techniques using multiple pointers are easy to describe with ICON. As shown on Figure 9, Swing applications can be controlled this way by simply cloning the configuration of Figure 11. This technique is however limited due to the fact that Swing has been designed for a single pointer (e.g., clicking somewhere with a pointer closes a menu opened by the other pointer). In contrast, application devices can be designed to support concurrency. ICONDraw's drawing tools are first-class objects that can be instantiated without limit. This makes it possible for example to dedicate each physical device to a drawing tool. Figure 1 also shows how an additional positional device can be used to draw lines or rectangles using bimanual interaction.

3.4.3 Making Use of Additional Dimensions

With ICON, it is also easy to exploit any additional dimension of an input device. Figure 10 shows how the pressure channel of a tablet can be assigned to the brush size inside ICONDraw.

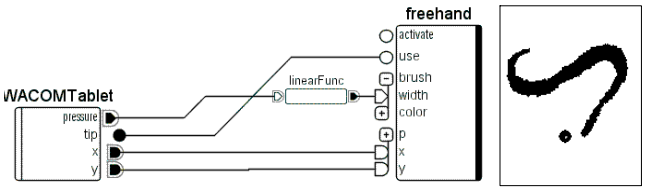

Figure 10: Pressure control of the freehand device.

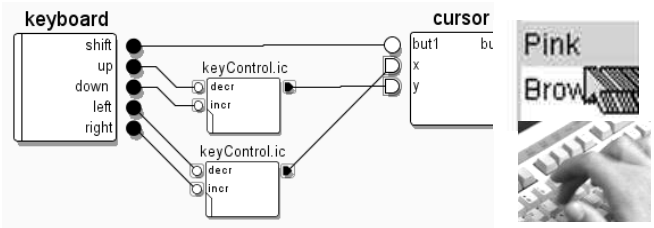

Figure 11: Key control adapters.

3.4.4 Switching Input Devices

Switching compatible devices such as the mouse and the tablet is just a matter of substituting one to another in the configuration. If devices are of different nature, emulation can be performed at low-level by using an adapter device, or by building an emulation technique by hand if no appropriate adapter is present. Figure 11 shows how an adapter can be inserted between a keyboard and a cursor: the KeyControl adapter is a compound device that has been built with ICON, and allows smooth control of continuous dimensions using two-state devices. Of course, more device-specific techniques can be built using ICON. For example, keyboard keys can be directly assigned to buttons in a window.

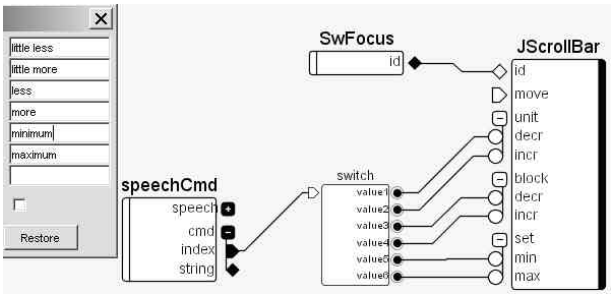

Figure 12: Speech control of a scrollbar.

3.4.5 Using Speech

Although it does not support grammars, the speech device can be used for simple command control. Figure 12 shows a configuration for controlling Swing scrollbars at model-level with commands such as "less", "little more" or "minimum"; The figure also shows the device property windows in which the vocabulary is specified. Examples of application-specific uses in ICONDraw include control of the brush colour or size with commands like "blue", "red", "bigger", "smaller" or direct control of the brush size with speech level. A "speech cursor" device technique has also been implemented for generic speech control of applications.

3.4.6 Using Advanced Input Devices

ICON is suitable for describing highly parallel interaction techniques with advanced input devices and direct control techniques. Figure 13 shows an example of using a Magellan device for simultaneously controlling zoom, pan and rotation on a zoomable document. This example, which uses our minimal ICON support for the Jazz toolkit, also allows making annotation and moving objects at the same time.

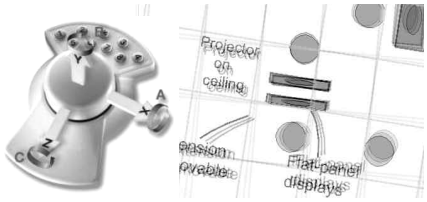

Figure 13: Using a pan/zoom/rotate technique with a 6DOF device.

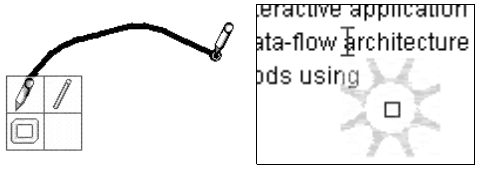

Figure 14: The Toolglass and the Floating QuikWriting interaction techniques.

3.4.7 Using Advanced Interaction Techniques

Although simple interaction techniques can be built with ICON editor, more complete techniques are better implemented as specialized

devices. Figure 14 gives two examples of transparency-based interaction techniques that are currently available in ICON. The *Toolglass* device implements the Toolglass interaction technique [13], with controllable transparency (Figure 14). This device sends activation or deactivation signals to connected tools according to positional values it receives, and renders itself on top of the application. The "Floating QuikWriting" device, based on Ken Perlin's text input method for PDAs [35], allows text input with a positional device. Because it only needs a single gesture to move the caret and insert or delete characters, it is well-adapted to proofreading tasks. To work, it just has to be inserted between a positional device and the Swing text device. ICON also has a device that supports more general and conventional gesture techniques by using the Satin library [22].

3.4.8 Controlling Existing Applications

Compatibility with Swing applications is built-in, but deeper control can be achieved by externalising application-specific entry points as application devices. If full externalisation is not possible for a big application which has not been designed for concurrent control, partial ICON support can be achieved.

ICON can also be used for controlling non-Swing applications: except for the small set of Swing-specific devices, ICON is independent from graphical toolkits. Control of external applications is limited by the absence of support for graphical feedback and event deactivation. However, ICON can be used to enhance standard event handling. Using the Java Native Interface, we implemented devices making simple queries to the active Microsoft Word document, and we could control commands such as cut and paste by voice, or manipulate the document zoom with analogue physical controls.

3.5 Discussion

3.5.1 Applicability of ICON to Post-WIMP Paradigms

ICON's graphical editor allowed us to specify a wide range of non-conventional interactions, although we could not experiment with all known interaction techniques. We however think that our approach is suitable for most Post-WIMP interaction paradigms.

First of all, ICON's data-flow model naturally supports concurrency and allows to easily describe highly parallel, non-modal interaction techniques that make use of multiple and heterogeneous input devices. Concurrency does not add complexity to input configurations. Conversely, control-intensive behaviours are better described using approaches such as State Transition Diagrams or Petri Nets [4]. Control (mode handling, conditional dispatch) can still be described in ICON, the price to pray being a visual complexity that can be reduced by module encapsulation. We also think that control-intensive behaviours are most found in WIMP techniques that make use of modes and multiplexing mechanisms.

We currently support Post-WIMP interaction styles that are mostly reactive and deterministic, and we do not address issues such as ambiguity solving, or smart interpretation and fusion of natural modalities. We still believe, however, that interaction models such as [28] can highly benefit from using ICON as a low-level layer.

Our support for overlay graphical feedback fits most Post-WIMP look-and-feels, including gesture, Toolglasses and Pie Menus. ICON is currently limited by the set of input APIs supported, but because it uses very low-level abstractions there are no conceptual limit to input device support. Most input devices are well-described using low-level channels, and we think that full support of a standard such as USB HID would be sufficient to take into account nearly all existing and future devices. Emerging interaction paradigms such as ubiquitous computing, context awareness and tangible interfaces make use of sensors that can also be described as ICON devices.

3.5.2 Support For Input Adaptability

Controllability. Programmers can make their applications more controllable by implementing application-specific devices and providing input configurations that exploit advanced input devices and/or interaction techniques. ICON makes easy to build device and task-dedicated interactions, and it encourages innovation.

Accessibility. Building configurations with minimal accessibility support is quite straightforward using adapter devices. New adapter devices can be implemented in java. ICON also allows building complex behaviours and encapsulating them into compound devices. Finally, using ICON's visual editor, the application programmer can test and implement a wide range of dedicated accessibility techniques according to specific input contexts (users and working environments) and tasks.

Configurability. ICON's input editor allows application users to customize interaction techniques to suit their needs. We don't expect the end user to build full input configurations. Instead, application programmers or device vendors can provide a set of working configurations, that end users can select, parameterise or customize, depending on their level of expertise.

4. CONCLUSION AND FUTURE WORK

In this article, we presented ICON, an input management system that allows interactive applications to achieve a high level of input adaptability. We introduced the term of *input adaptability* and explained why it was important. We described the concepts behind ICON and showed how the paradigm of cascading devices could be used as an architectural basis to describe interaction techniques and achieve a high level of input adaptability. We strongly believe that the input management supported by current GUI toolkits is not rich enough for current and future needs and that the concept of input configurations is well suited to replace it.

We are currently extending ICON in three directions: designing and experimenting with new interaction techniques, improving the integration of advanced graphics toolkits such as zoomable and 3D toolkits and supporting more input devices. ICON is currently being used in two research projects and is available as a free Java package at the following URL: `[URL omitted for blind review]`. We hope other research projects will experiment with it and send us feedback to improve it.

5. REFERENCES

[1] Adobe Creative Team, Adobe Photoshop 7.0: Classroom in a Book, Adobe Press; ISBN: 0-321-11562-7

[2] Avid Inc. Channel Developer's Kit, Softimage Inc., 2000

[3] Balakrishnan, R. Fitzmaurice, G. Kurtenbach, G. Buxton, W. Digital tape drawing, Proc. of UIST'99, p.161-169, November 07-10, 1999, Asheville, North Carolina, United States.

[4] Bastide, R., Navarre, D., and Palanque, P. A model-based tool for interactive prototyping of highly interactive applications. In CHI'02 extended abstracts on Human factors in computer systems, pages 516-517. ACM Press, 2002.

[5] Baudel, T., A mark-based interaction paradigm for free-hand drawing, Proc. of the 7th annual ACM symposium on User interface software and technology, p.185-192, November 02-04, 1994, Marina del Rey, California, United States.

[6] Bederson, B., Meyer, J., Good, L., Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java, In ACM UIST 2000 ACM Press, pp. 171-180. 2000.

[7] Berry, G. The Esterel v5 language primer. Tech report, april 1999. http://www-sop.inria.fr/meije/esterel/doc/main-papers.html.

[8] Bier, E.A. and Freeman, S., MMM: A User Interface Architecture for Shared Editors on a Single Screen. in Proc. of UIST 91, ACM Press, 79-86.

[9] Bolt, R. A. "put-that-there": Voice and gesture at the graphics interface. In SIGGRAPH '80 Proc., volume 14, 1980.

[10] Borning A. Thinglab - A Constraint-Oriented Simulation Laboratory. PhD thesis, Stanford University, july 1979. Also available as STAN-CS-79-746 Stanford Computer Science Department technical report.

[11] Buxton, W. There's More to Interaction than Meets the Eye: Some Issues in Manual Input. In Norman, D. A. and Draper, S. W. (Eds.), (1986), *User Centered System Design: New Perspectives on Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 319-337.

[12] Doherty, E. Cockton, G., Bloor, C., Benigno, D. Improving the Performance of the Cyberlink Mental Interface with the "Yes / No Program", CHI 2001 Proc., Seattle, Washington, USA

[13] Bier, E. Stone, M. Fishkin K. Buxton, W. Baudel, T. A taxonomy of seethrough tools. In Proc. of the CHI'94 conference. ACM Press, 1994, p. 358-364.

[14] Englander, R. Developing Java Beans, O'Reilly & Associates, June 1997, ISBN: 1-56592-289-1.

[15] Esteban, O., Chatty, S., Palanque, P. Whizz'ed : a visual environment for building highly interactive software. In Proc. of IFIP INTERACT'95 : Human-Computer Interaction, pages 121-126, 1995.

[16] Ferguson, P. The X11 Input Extension: Reference Pages. The X Resource, 4 (1), 1992 195--270.

[17] Foley, J., Van Dam, A., Feiner, S. and Hughues, J. Computer Graphics Principles and Practice. Addison-Wesley, 2nd edition, 1990.

[18] Greenberg, S. and Fitchett, C. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. Proc. of the UIST 2001 14th Annual ACM Symposium on User Interface Software and Technology, November 11-14, Orlando, Florida, p209-218, ACM Press.

[19] Halbwachs, N., Caspi, P., Raymond, P. and Pilaud, D. The synchronous data-flow programming language LUSTRE. In Proc. of the IEEE, volume 79, September 1991.

[20] Honeywell, S. Quake III Arena: Prima's Official Strategy Guide. Prima Publishing, 1999.

[21] Hourcade, J.P. and Bederson, B.B. Architecture and Implementation of a Java Package for Multiple Input Devices (MID), Human-Computer Interaction Laboratory, University of Maryland, College Park, MD 20742, USA, 1999.

[22] Hong, J. and Landay, J. "SATIN: A Toolkit for Informal Ink-based Applications." In UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters , 2(2), p. 63-72.

[23] Ingalls, D., Wallace, S., Chow, Y., Ludolph, F. and Doyle, K. Fabrik : A Visual Programming Environment. In Norman Meyrowitz editor, OOPSLA'88 : Object-Oriented Programming Systems, Languages and Applications. pp 176-190, 1988.

[24] Jacob, R., Deligiannidis, L. and Morrison, S. A Software Model and Specification Language for Non-WIMP User Interfaces. ACM Transactions on Computer-Human Interaction, 6(1) :1-46, march 1999.

[25] Knep, B., Hayes, C., Sayre, R., Williams, T. (1995). Dinosaur input device. Proc. of CHI'95: ACM Conference on Human Factors in Computing Systems.

[26] LCS/Telegraphics. The Wintab Developers' Kit, http://www.pointing.com/WINTAB.HTM, 1999.

[27] Mackay, W. E. Which Interaction Technique Works When? Floating Palettes, Marking Menus and Toolglasses Support Different Task Strategies. in Proc. International Conference on Advanced Visual Interfaces (AVI 2002) , ACM Press, pages 203-208, April, 2002.

[28] Mankoff, J., Hudson, S. and Abowd, G.. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In Proc. of CHI'00, ACM Conference on Human Factors in Computing Systems, pages 368-375, N. Y., April 1-6 2000.

[29] McLoone, H., Hinckley, K. & Cutrell, E. Bimanual Interaction on the Microsoft Office Keyboard. In Rauterberg,M., Menozzi, M, & Wesson, J. (Eds.), Human-Computer Interaction INTERACT '03, Zürich, September 2003, pp. 49-56.

[30] Microsoft Corporation, DirectX 9 Graphics Programmers Guide, Microsoft Press International; ISBN: 0735616531, March 2001.

[31] Microsoft Corporation, Microsoft Active Accessibility SDK 1.2, Microsoft Corporation 1999.

[32] Moore, M, Mynatt, E. Kennedy, P. and Mankoff, J. Nudge and Shove: Frequency Thresholding for Navigation in Direct Brain-Computer Interfaces. In Proc. of CHI 2001 Conference Companion, Technical Notes March, 2001. pp. 361-362.

[33] Myers, B. A New Model for Handling Input. ACM Transactions on Information Systems, 8(3) :289-320, july 1990.

[34] Partridge, K. Chatterjee, S. Sazawal, V. Borriello, G. and Want R. TiltType: accelerometer-supported text entry for very small devices, the 15th annual ACM symposium on User interface software and technology, 2002, Paris, France. pp. 201-204.

[35] Perlin, K. Quikwriting: Continuous stylus-based text entry. In Proc. of UIST '98. ACM, November 1998.

[36] Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. & Carey, T. (1994) Human-Computer Interaction. Wokingham, UK: Addison-Wesley.

[37] Rekimoto, J. Tilting Operations for Small Screen Interfaces. Proc. of UIST '96, pp.167-168.

[38] Salber, D., Dey, A., Abowd, G. The Context Toolkit : Aiding the Development of Context-Enabled Applications. In Proc. of CHI'99, ACM Conference on Human Factors in Computing Systems, pages 434-441, New York, may 15-20 1999.

[39] Sense8 Corp. The World Toolkit Manual, Sense8, 1999.

[40] Sun Microsystems Inc. Java Speech API Programmer's Guide, , http://java.sun.com/ 1998.

[41] Tyson, H., Scott, H., and Gary, L. Integrating gesture and snapping into a user interface toolkit. In Proc. of UIST'90, pages 112-122. ACM Press, 1990.

[42] Virtools dev. Virtools SA, 2001. http://www.virtools.com/.

[43] Westergaard M. Supporting Multiple Pointing Devices in Microsoft Windows. Microsoft Summer Workshop for Faculty and PhDs. Cambridge, England, September 2002.

[44] Dragicevic, P. and Fekete, J.-D., Input Device Selection and Interaction Configuration with ICon, Proc. of IHM-HCI 2001,Blandford, A.; Vanderdonckt, J.; Gray, P., (Eds.): People and Computers XV – Interaction without Frontiers, Lille, France, Springer Verlag, pp.543-448.

[45] Thevenin, D., Coutaz, J., Plasticity of User Interfaces: Framework and Research Agenda. In Proc. of Interact'99, September 3, 1999. Chapman & Hall, London, pp. 110-117