

# The Input Configurator Toolkit: Towards High Input Adaptability in Interactive Applications

**Pierre Dragicevic**  
Ecole des Mines de Nantes  
4, rue Alfred Kastler  
La Chantrerie  
F44307 Nantes Cedex, France  
[Pierre.Dragicevic@emn.fr](mailto:Pierre.Dragicevic@emn.fr)

**Jean-Daniel Fekete**  
INRIA Futurs/LRI  
Bat. 490,  
Université Paris-Sud  
F91405 Orsay Cedex, France  
[Jean-Daniel.Fekete@inria.fr](mailto:Jean-Daniel.Fekete@inria.fr)

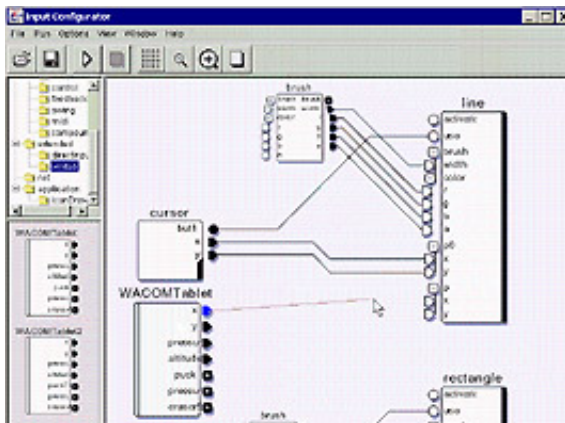


Figure 1: Configuring a Drawing Application for Bimanual Interaction with ICON

## ABSTRACT

This article describes ICON (Input Configurator), an input management system that enables interactive applications to achieve a high level of input adaptability. We define *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 describe several examples of interaction techniques implemented using ICON with little or no support from applications that are hard or impossible to implement using regular GUI toolkits.

## Categories and Subject Descriptors

D.1.7 [Visual Programming], H.5.1 [Multimedia Information Systems], H.5.2 [User Interfaces]

## Keywords

Interaction techniques, Input devices, Visual programming, Toolkits.

## 1. INTRODUCTION

For years, using a computer was synonymous with 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 heterogenous platforms such as palmtops, laptops, wearables, tablet PCs and smart phones. Computer users 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 richer and more complex: mice, keyboards and joysticks are continuously being augmented with new controls, whereas alternative devices such as web cams, voice recognition microphones, tablets, and dedicated gaming devices are becoming affordable and popular. The population of desktop computer users has gained in experience, and their demands evolved from simplicity, standardization and ease of use to efficiency, speed and usability for all.

Unfortunately, available applications and tools currently put a brake on the evolution of input interaction: they are still designed for a fixed 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.

This article explains how, with our system ICON (Input Configurator), we can create fully input-configurable interactive applications and 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

For a better accessibility and portability, it is critical for interactive applications to be able to exploit modified sets of input devices. For optimized 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. We therefore define *input adaptability* as the combination of control-

lability, accessibility and configurability:

**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.

**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.

**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 customization to specification of rich interaction techniques. Ease of specification also plays a crucial role in input configurability.

Buxton not only demonstrated the importance of matching input devices with tasks, but also gave a good idea of the subtleness of the notion of device/task compatibility [1]. 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. Task-dedicated input devices are widely used in geometric modeling and video games, and can even be part of the application design [2].

However, most real-world applications are designed for a fixed set of input devices and interaction techniques and offer almost no input adaptability.

### 3. RELATED RESEARCH WORK

There has been much less work on input than on graphics in the HCI community. Graphical toolkits such as Garnet/Amulet [3] and Subarctic [4] describe standard event handling in a cleaner, more general and 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. Most so-called “Post-WIMP” toolkits are specialized towards one specific interaction paradigms, such as gesture recognition [5], ambiguity solving through mediation [6], multi-pointer interaction [7] [8], context awareness [9], or tangible interfaces [10]. They allow a better controllability using standard or non-standard devices, but still support a limited set of input devices and use them in *ad-hoc* ways. One exception is the Phidgets / WidgetTaps library [10], which binds physical devices to widgets at the model level. Phidgets allow highly parallel control with physical devices but interaction techniques are limited to direct widget control with one-dimensional devices.

3D toolkits such as World-Toolkit [11] or Avid/Softimage [12] support alternative input devices, mainly 6 degrees of freedom (6DOF) trackers. A channel-based model allows free assignation of device channels to dimensions 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.

Control-flow approaches such as ICO/PetShop [13] use state-transition diagrams or Petri nets to describe control-intensive, highly modal parts of interactive applications. Data-flow-based editors have been used in various domains, but their application to

interaction specification has been rarely exploited outside the area of 3D authoring and animation. Vrttools Dev [14], for example, uses a data-flow editor for specifying 3D input techniques interactively. Jacob’s VRED system [15] uses both a control-flow (state transition diagrams) and a 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 [16]. This editor has been successfully used to specify animation and bimanual interaction, though other techniques and input devices have not been investigated.

### 4. INPUT CONFIGURATOR

ICON is a novel system for building input-reconfigurable interactive applications, based on a reactive data-flow architecture that describes input techniques using interconnected modules.

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 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).

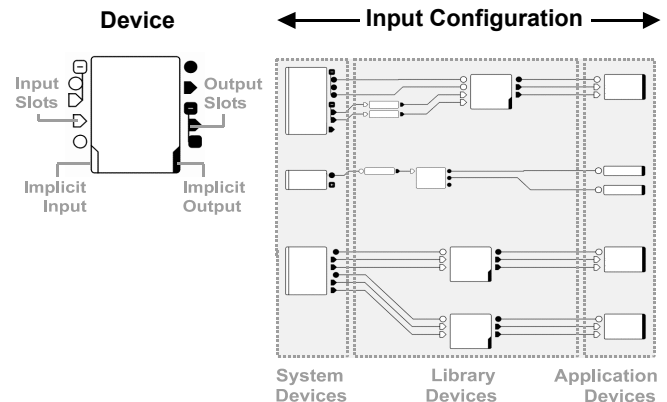


Figure 2: ICon components

Whereas the basic behavior of a device is processing input values into output values, devices can have *implicit input*, i.e. additional source of information not fully described by its set of input slots, e.g. input devices. Similarly, devices having *implicit output* produce alternative effects in addition to simply putting values on the output slots. Examples are devices that manipulate objects of an application, or devices producing graphical or sound feedback.

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).

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 are controlling 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.

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.

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.

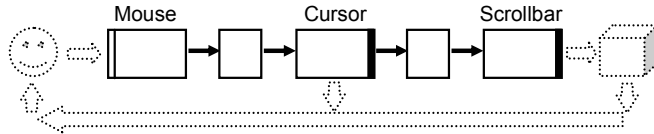


Figure 3: Scrolling through a document.

ICON describes feedback with implicit input and output. Figure 3 also shows the feedback loop through implicit I/O: mouse has implicit input, Cursor has implicit feedback and the Scrollbar updates its document view.

## 5. THE ICON 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. ICON is written in Java, and uses native libraries for managing input devices. On the Microsoft Windows operating systems, ICON currently supports multiple graphical tablets through Wintab API[17], multiple gaming devices and 6DOF controllers through DirectInput, speech recognizers through IBM JavaSpeech and Midi instruments. Under X Windows platforms, it provides access to XInput Extension [18] devices. Figure 4 shows some input devices as they appear in ICON: the low-level mouse sends screen-independent delta values.

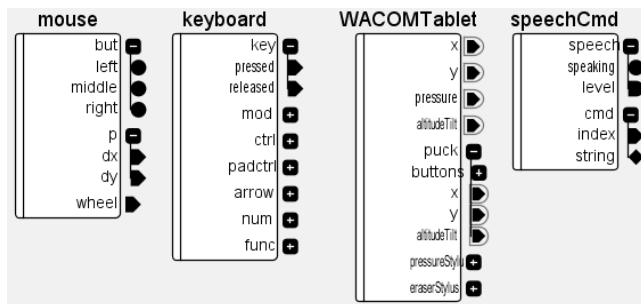


Figure 4: Mouse, keyboard, tablet and speech input devices.

Java application developers can enhance ICON controllability of their application by implementing specific application devices. Instead of listening to specific Swing events, these applications simply describe the way they expect to be controlled in term of ICON devices. They then become configurable interactively or through saved configurations.

ICON configurations can be built or modified using a visual editor described in [19]. We now describe some novel interaction techniques we have built using ICON without any specific application support.

## 5.1 Adding pointing devices

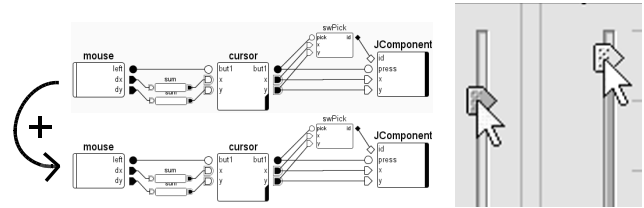


Figure 5: multi-pointer interaction with Swing.

Because each cursor device manages its own feedback, multi-pointer control is easy to achieve using ICON. Widget-level multi-user or bimanual control of Swing applications, as shown on figure Figure 5, can be achieved by simply cloning the standard mouse configuration.

## 5.2 Making use of additional dimensions

With ICON, it is easy to exploit any additional dimension of an input device. Figure 6 shows how the pressure channel of a tablet can be assigned to the brush size inside an ICONDraw configuration.

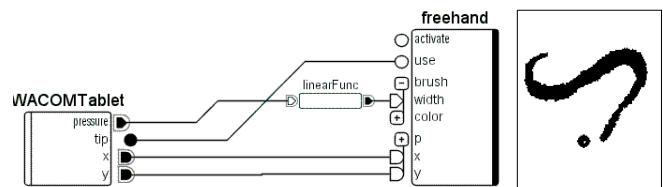


Figure 6: Pressure control of the freehand device.

## 5.3 Using Speech

The speech device can be used for simple command control. Figure 7 shows a configuration for controlling Swing scrollbars at model-level with commands such as “less”, “little more” or “minimum”. A “speech cursor” device technique has also been implemented for generic speech control of applications.

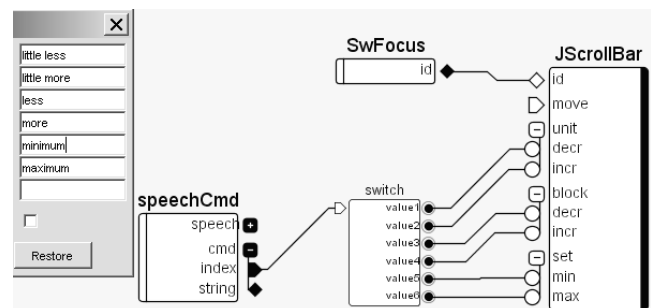


Figure 7: Speech control of a scrollbar.

## 5.4 Using advanced input devices

ICON is well-adapted for describing highly parallel interaction techniques with advanced input devices and direct control techniques. Figure 8 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 [20], also allows making annotation or moving objects at the same time.

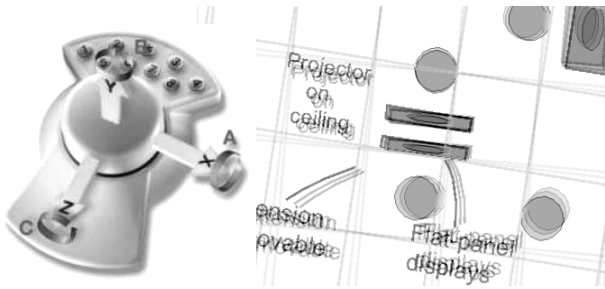


Figure 8: Using a pan/zoom/rotate technique with a 6DOF device.

## 5.5 Using advanced interaction techniques

Figure 9 gives two examples of transparency-based interaction techniques that are currently available in ICON. The *Toolglass* device implements the toolglass interaction technique [21], with controllable transparency (Figure 9). 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 [22], 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. The QuikWriting device is inserted between a positional device and the Swing text device. ICON also provides a device that supports more general gesture techniques relying on the Satin library [5].

## 6. CONCLUSION AND FUTURE WORK

In this article, we introduced the concept of *input adaptability* and showed how we achieved it using ICON, available as a free Java package at the following URL:

<http://www.emn.fr/dragicevic/ICON>. We hope other research projects will experiment with it and send us feedback to improve it.

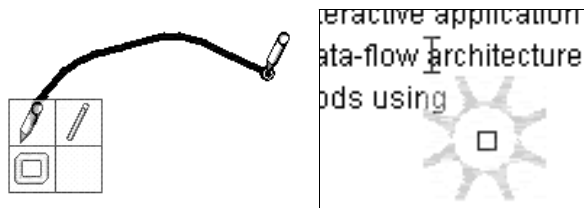


Figure 9: The Toolglass and the Floating QuikWriting interaction techniques.

## REFERENCES

1. Buxton, W., There's More to Interaction than Meets the Eye: Some Issues in Manual Input. 1986: p. 366-375.
2. Knep, B., et al., Dinosaur Input Device, in Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems. 1995. p. 304-309.
3. Myers, B.A., A New Model for Handling Input. ACM Transactions on Information Systems, 1990. 8(3): p. 289-320.
4. Tyson, H., H. Scott, and L. Gary. Integrating gesture and snapping into a user interface toolkit. in In Proceedings of UIST'90, ACM Symposium on User Interface Software and

- Technology. 1990: ACM Press.
5. Hong, J.I. and J.A. Landay, SATIN: A Toolkit for Informal Ink-based Applications. In UIST 2000, ACM Symposium on User Interface Software and Technology, CHI Letters, 2000. 2(2): p. 63-72.
6. Mankoff, J., S. Hudson, and G. Abowd, Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. CHI Letters, 2000: p. 368-375.
7. Bier, E.A. and S. Freeman. MMM: A User Interface Architecture for Shared Editors on a Single Screen. in Proceedings of the ACM Symposium on User Interface Software and Technology. 1991: ACM.
8. Hourcade, J.P. and B.B. Bederson, Architecture and Implementation of a Java Package for Multiple Input Devices (MID). 1999, University of Maryland: College Park, MD 20742, USA.
9. Salber, D., A. Dey, and G. Abowd. The Context Toolkit : Aiding the Development of Context-Enabled Applications. in In Proceedings of CHI'99, ACM Conference on Human Factors in Computing Systems. 1999. New York: ACM Press.
10. Greenberg, S. and C. Fitchett. Phidgets: Easy Development of Physical Interfaces through Physical Widgets. in Proceedings of the UIST 2001 14th Annual ACM Symposium on User Interface Software and Technology. 2001. Orlando, Florida: ACM Press.
11. Sense8 Corp, The World Toolkit Manual. 1999, Sense8.
12. Avid, I., Channel Developer's Kit. 2000, Softimage Inc.
13. Bastide, R., D. Navarre, and P. Palanque. A model-based tool for interactive prototyping of highly interactive applications. in In CHI'02 extended abstracts on Human factors in computer systems. 2002: ACM Press.
14. Virtools S.A., Virtools dev. 2001.
15. Jacob, R.J.K., L. Deligiannidis, and S. Morrison, A Software Model and Specification Language for Non-WIMP User Interfaces. ACM Transactions on Computer-Human Interaction, 1999. 6(1): p. 1-46.
16. Esteban, O., S. Chatty, and P. Palanque. Whizz'ed : a visual environment for building highly interactive software. in In Proceedings of IFIP INTERACT'95 : Human-Computer Interaction. 1995.
17. LCS/Telegraphics, The Wintab Developers' Kit. 1999.
18. X Consortium, The X Input Extension. X Journal, 1992.
19. Dragicevic, P. and J.-D. Fekete. Input Device Selection and Interaction Configuration with ICON. in Proceeding of IHM-HCI 2001. 2001. Lille, France: Springer Verlag.
20. Bederson, B., J. Meyer, and L. Good. Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java. in Proceedings of the 13th annual ACM symposium on User interface software and technology. 2000. San Diego, California: ACM Press.
21. Bier, E.A., et al., Toolglass and Magic Lenses: The See-Through Interface, in Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems. 1994. p. 445-446.
22. Perlin, K. Quikwriting: Continuous stylus-based text entry. in In Proc. of UIST '98. 1998: ACM Press.