HAL
open science

Input Device Selection and Interaction Configuration with ICON

Pierre Dragicevic, Jean-Daniel Fekete

To cite this version:

Pierre Dragicevic, Jean-Daniel Fekete. Input Device Selection and Interaction Configuration with ICON. Proceedings of the International Conference IHM-HCI 2001, Sep 2001, Lille, France. pp.543-448. hal-00877336

HAL Id: hal-00877336

https://inria.hal.science/hal-00877336v1

Submitted on 28 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Input Device Selection and Interaction Configuration with ICON

Pierre Dragicevic
Ecole des Mines de Nantes
4, rue Alfred Kastler, La Chantrerie
44307 Nantes Cedex, France

Pierre.Dragicevic@emn.fr

Jean-Daniel Fekete
Ecole des Mines de Nantes
4, rue Alfred Kastler, La Chantrerie
44307 Nantes Cedex, France
+33 2 51858208
Jean-Daniel.Fekete@emn.fr

ABSTRACT
This paper describes ICON, a novel editor designed to configure a set of input devices and connect them to actions into a graphical interactive application. ICON allows physically challenged users to connect alternative input devices and/or configure their interaction techniques according to their needs. It allows skilled users – graphic designers or musicians for example – to configure any ICON aware application to use their favorite input devices and interaction techniques (bimanual, voice enabled, etc.).

ICON works with Java Swing and requires applications to describe their interaction styles in terms of ICON modules. By using ICON, users can adapt more deeply than before their applications and programmers can easily provide extensibility to their applications.

Keywords
Multiple inputs, input devices, interaction techniques, toolkits, assistive technologies

INTRODUCTION
Today, interactive desktop applications manage a very limited set of input devices, typically one mouse and one keyboard. However, the population of users requiring or simply possessing alternative input devices is growing, as well as the number of new devices available.

Given the proliferation of input devices and the importance of tasks performed by users on a computer, being able to adapt an existing application to one or several input devices is an important issue for physically challenged users, musicians, video game players, graphic artists etc. These users find their environment more usable or simply improved when they use their favorite input devices with existing applications.

However, the complexity of supporting alternative input devices is currently very high: each application has to explicitly implement some code to manage each device and all the desired interaction techniques using these devices. At best, specialized applications support a limited set of devices suited to their task.

In this paper, we describe ICON (Input Configurator), a system for selecting alternative input devices and configuring their interaction techniques interactively.

Using ICON, users can connect additional input devices – such as tablets, voice recognition software, assistive devices or electronic musical instruments – to an ICON aware application and/or assign specific interactive behavior to connected devices. Figure 1 shows a typical configuration.

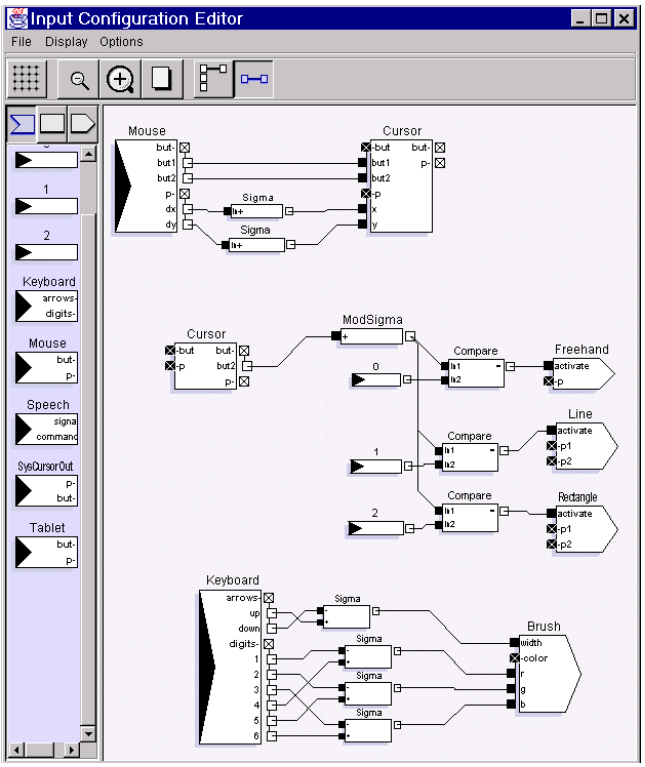

Figure 1: Screenshot of ICON showing part of the configuration of a drawing editor. The right mouse changes the selected tool and keypad keys change the color and line width attributes. Mouse relative positions are added and sent to a "cursor" module that abstracts a 2D locator device.

Adding new devices and configuring them using ICON not only opens applications to the use of alternative, better suited devices but also tremendously simplifies the integration of new interaction techniques published each year on conferences like CHI (for example [9, 12, 23]) but very seldom implemented on commercial products.

ICON is implemented using Java Swing and requires applications to externalize their interaction techniques to be effective. The effort required to do so is very modest compared to the benefits.

Typical ICON users need not be computer scientists; a good understanding of computer systems is sufficient. Users with disabilities or special requirements may need the help of such "power" users to configure their application.

This paper first describes a typical scenario a user would follow to edit the input configuration of a simple application. After comparing ICON to related work, we describe the ICON editor in details; we give then more implementation details and discuss practical issues for the use of ICON in applications.

SCENARIO

In this section, we show how John, a typical ICON user, may adapt a sample application called "IconDraw" that can draw lines, rectangles and freehand curves. By invoking the "configure" menu of IconDraw, John starts ICON that displays the current configuration as in Figure 1. This dataflow diagram shows several connected blocks called modules. Each module has input or output slots where links can be connected. Only connected slots are displayed in the figures. The two input devices of the default configuration – the mouse and the keyboard – are on the left, connected to some processing modules and finally to IconDraw's interaction tools appearing as input modules.

In IconDraw's standard configuration, the mouse is connected to a cursor module which displays feedback and is used as a virtual device. The right mouse button is used to cycle through the drawing tools. Keyboard keys are used to change the color and size of the lines. John can then change the configuration, save it to a file or load it from the configuration files available for IconDraw.

Stabilizing the Mouse Position

John remembers his friend Jane wants a drawing program but couldn't use a regular one because she suffers from Parkinson's disease that provokes uncontrollable hand shaking. With ICON, he can stabilize the pointer position by inserting a low-pass filter – averaging pointer positions to remove quick moves – between the mouse device and the cursor as shown in Figure 2.

To insert a LowPass module, John drags it from the left pane – where all existing modules are shown – and drops it in the editor pane. Clicking on one slot and dragging into another creates a connection. The configuration is effective immediately in IconDraw. When finished, John saves the configuration and sends it by email to Jane.

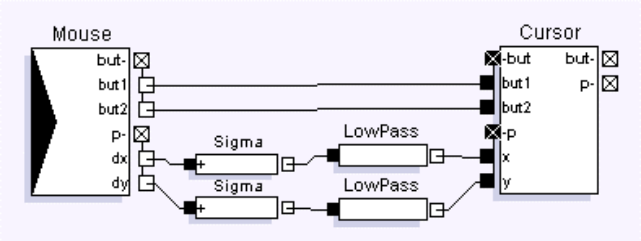

Figure 2: LowPass filters are inserted between the mouse and the cursor to stabilize the position.

Adding a Pressure Sensitive Stylus

John is a graphic designer and has a tablet with a pressure sensitive stylus. To use it inside IconDraw, he needs to disconnect the mouse, drag the tablet module in the ICON pane and connect it to the cursor through scale modules.

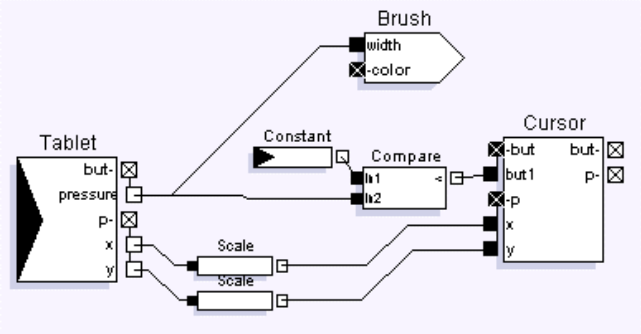

Figure 3: Adding a pressure sensitive stylus to IconDraw where the pressure changes the line width of the drawing.

To further use the pressure for changing the line width when drawing, he needs to connect the pressure slot of the stylus to the size slot of the brush used by the drawing tools, as shown in Figure 3. Brushes abstract graphic attributes just like cursors abstract positional devices. John can now draw with the stylus and have varying width strokes when using the freehand drawing tool.

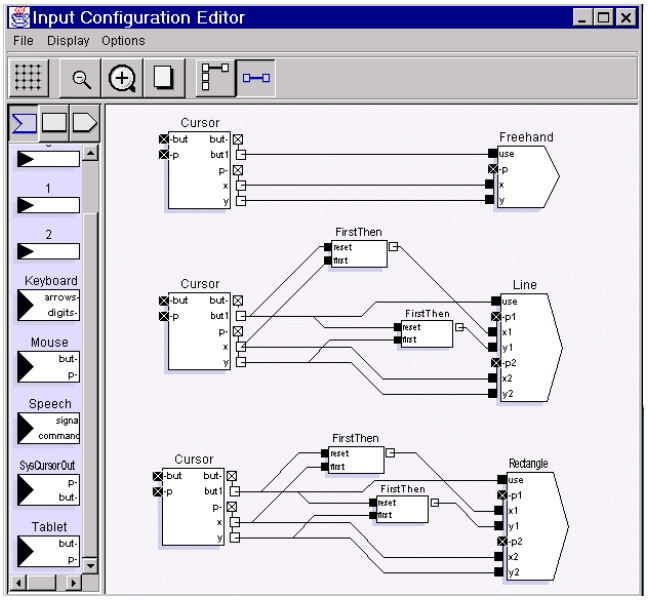

Figure 4: creation tools of IconDraw represented as input modules.

Configuring for Bimanual Interaction

Figure 4 shows part of the configuration controlling IconDraw's interaction tools. John now wants to use both his mouse and the stylus. A bimanual configuration requires a second pointer that can be dropped from the left pane into the ICON pane. Figure 5 shows the configuration required to create a line using bimanual interaction: one should be connected to the "p1" slot and the second to the "p2" slot. A combination of boolean modules determine when the creation mode is triggered and when it is finished.

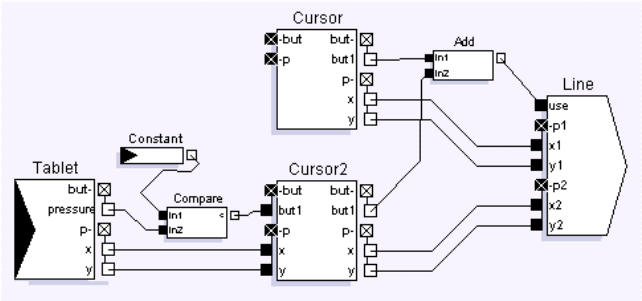

Figure 5: Configuration for creating a line with bimanual interaction.

Other Configurations

Current input modules also include voice and gesture recognition. John could use it to control the selected tool or to change the color if he wishes, effectively adapting the program to his skills and environmental particularities such as limited desk space or noisy environment.

RELATED WORK

There have been several attempts at simplifying the connection of alternative input devices or specifying the configuration of interactive applications.

Assistive technologies

Assistive technologies include hardware devices and software adapters. They are designed to allow disabled users to work on a desktop computer. Software adapters can be external applications that take their input from various special hardware devices and translate them as if they were actions on the mouse and keyboard. NeatTool [6] is such a system and configurations are designed using a rich graphical dataflow system. However, systems like NeatTool are limited to using existing interaction techniques of an application. More generally, external software adapters have problems when they need to maintain the internal states of an application like the current selected drawing tool.

In contrast, internal software adapters are becoming more common. Microsoft Active Accessibility® [20] and Java Swing [1] have provisions for accessibility and programmers can modify existing applications to adapt them to various input and output configurations. However, accessibility functions are not well designed for continuous interaction such as drag and drop or line drawing, and no graphical configuration tools exist yet, requiring a programmer's skill and sometimes source access to use them.

Games

Most current games offer some configuration options and all 3D video games offer a large choice of supported input devices. However, most of the games have a set of standard configurations and adapt alternative devices by compatibility. A regular action game will provide one configuration using the keyboard and another using positional devices (mouse, joystick or any compatible device). Sometimes, very specific devices are also managed like a force feedback joystick for flight simulators or a driving wheel for car racing. However, no general mechanism could allow a pianist to use a midi keyboard on a car racing program for example. The configuration is usually done through a simple form based interface or a simple script-like language which only allows direct bindings of device channels [13].

Furthermore, alternative input devices can only be used to play the game but other controls such as menus or dialogs can only be controlled by the regular mouse and keyboard.

3D Toolkits

3D toolkits and animation environments are usually aware of alternative input devices. The AVID/SoftImage system implements a "channel" library to connect any valuator device as a set of channels [2]. These channels can in turn be connected to internal values inside 3D models or trigger the internal constraint solver to perform sophisticated direct animations. However, channels are limited to these direct animations and cannot be used to enhance the interaction of the 3D modeling tool itself for instance. The World toolkit [25] can use any kind of device if it is described as an array of relative positional values. Again, this view of input device is meant for animation or direct control in VR environments but not for the interactive creation of objects or control of menus. Furthermore, the configuration of these input devices has to be programmed.

Recently, Jacob proposed a new architectural model to manage the interaction [15] using VRED, a dataflow system similar to ICON. Due to its complexity, VRED is meant to be used by expert programmers since it interacts deeply with the internals of animation programs.

2D Toolkits

There has been some attempts at simplifying the integration of alternative input devices in applications, such as the X Input Extension [8]. However, very few 2D programs use it and, when they do, their level of configuration is very limited. The X Toolkit [24] specifies a textual format to configure application bindings but its syntax is complex and requires the application to be restarted when it changes.

Myers described an interactive system for visual programming of interactions using interactors [22] in the Garnet environment. [21]. Garnet is still oriented towards programmers and interaction techniques cannot be changed

dynamically during the execution of an application. Furthermore, Garnet only manages one mouse and one keyboard.

Other systems for managing multiple devices exist such as MMM, Chatty's two-handed aware toolkit and Hourcade's MID library [5, 7, 14] but they all offer tools to programmers instead of users.

Classification

Table 1 summarizes this section, classifying existing systems in term of support for configurability and multiple input devices (MID). Configurability is classified in 6 categories: (1) none, (2) direct binding from device and events to program actions (i.e. without higher level control such as conditional binding), (3) environments configurable by users, (4) environments configurable by a programmer using a specialized language, (5) environments requiring a regular programming language for configuration, and (6) for completeness, adaptive environments that could automatically adapt interaction techniques and input devices to user's needs. No existing systems belong to this category yet. MID are classified in 4 categories: aware of only a fixed set of devices, aware of accessibility services and aware of many (a fixed set of classes) or any alternative devices.

Configure / MID	**Fixed set**	**Accessi-bility**	**Many**	**Any**
None	Most applications			
Direct binding	Video games		Midi config.	
User oriented	Hardware accessibility NeatTool	Software accessibil ity	Softimage Channels	**ICON**
Limited programmer oriented	Garnet interactors			VRED
Programming	Most toolkits	Java SWING MFC	World Tk	MMM Chatty MID
Adaptive				

Table 1: classification of current systems in term of support for configuration and multiple input devices.

THE ICON SYSTEM

The ICON Editor allows users to view and edit the mappings between all the available input devices and an application. It is based on a dataflow model, with the underlying semantics of reactive synchronous languages such as Esterel [4] or Lustre [11]. In this dataflow model, modules are input devices, processing devices, and application objects. Links are connections between input and output slots of the modules. A configuration is built by dragging modules into the ICON workspace and connecting them to perform high level actions, expressed as input modules. This section first describes how to use ICON, then how to design an ICON Aware application.

Using ICON

Modules that can be used to build an input configuration are available in three repositories : an output module repository, a processing module repository, and an input module repository (see Figure 1 on the left). Each type of module has its own graphical representation. New compound modules can also be created at will.

Output module repository: When the editor is launched, it asks the input APIs (WinTab [18] for the tablets, JavaSpeech [26] for the voice recognition engine, Direct Input [16] and USB [3] for yet other devices) for the set of connected input devices, and their capabilities. Device's capabilities are interpreted as a set of typed channels. For each detected device, a module is created and filled with output slots (corresponding to the device's channels), and is added to the output module repository. Timer modules also belong to this repository. With timer modules, users can detect timeouts, idle cursors, perform auto repeat or other time-oriented constructs.

Processing module repository: The editor contains a set of processing modules loaded from a library. A processing module has both input and output slots. There are three categories of processing modules: control modules, primitive modules and utilities. Control modules are useful to implement tests and control switches. Primitive modules include arithmetic, Boolean operations, comparisons and memorization of previous values. Utilities include debug modules and modules that could be built by composing primitives but are faster and smaller as primitives themselves.

Input module repository: This repository contains application-specific modules that are loaded when the user chooses an application to edit. These output modules show what the application needs in terms of input. It also contains global output devices such as the standard text output, the system cursor, or a Text-To-Speech engine.

Applications choose the level of granularity and how they describe their interactions in term of input modules. For example, all the atomic commands usually attached to menu items or buttons can be exposed as single input modules or can be grouped and exposed as one larger module. Exposing control objects as input modules is supported at the toolkit level. For application specific interactions, programmers have to provide suitable input modules according to the level of configurability they want to provide. More details are given in the next section.

Compound modules: Part of an input configuration can be used to create a user-defined module, by simply selecting a group of modules and issuing the "create compound" command. Selected modules and all the connections

between them are moved into a new compound module. External connections are also preserved: slots are automatically created on the compound device to enable external connections to internal modules.

Module properties: In addition to slots, some modules have properties that can be edited in a separate window. All modules have the *name*, *help,* and *enabled* properties. Other properties are mostly numerical parameters in mathematical processors.

Properties can also exist in input modules depending on the input API. As an example, recognizer devices issued from the JavaSpeech API have an array of string describing their vocabulary (empty by default). The properties describing the way an input module interprets user's actions to generate data is sometimes called the *device context*. Several instances of the same output module can live separately, each having its own device context.

Module cloning and shortcuts: Modules and groups of modules can be cloned in the workspace by dragging them while holding the control key. During this operation, all property values are copied. For example, cloning a processing module is useful when we have to perform the same operations elsewhere in the configuration. But an input module such as a mouse has a unique data source, and its clone will produce exactly the same data. However, cloning an input module can be useful to describe different device contexts (e.g. different working vocabularies for the same speech recognizer). The semantics of cloning an output module depends on how the application manages several instances of this module.

Another useful feature is module shortcuts. They allow the user to display the same device in different places of the workspace, so that an input configuration looks much more clearer(Figure 1-5 use shortcuts). A shortcut is made by dragging a device while holding both the shift and control keys.

Connections : Connections are created by dragging from one slot to another. Inconsistent connections – i.e. connections between input or output slots, type-incompatible slots, or connections that generate a cycle – are forbidden. Only authorized slots are highlighted during the dragging operation. ICON provides facilities for modifying connections, such as group deleting or fast reconnecting (changing one side of an existing connection).

Hierarchical slots: The configuration editor has a hierarchical representation of slots (Figure.6), which facilitates browsing of numerous slots. This also allows the structure of input devices to be preserved. Furthermore, hierarchical slots can be used to manipulate complex types.

Extended/Minimal Display : There are two display modes for modules. Extended mode shows all slots. Minimal mode shows only used slots, which reduces the visual complexity of a configuration. Entering a module with mouse cursor automatically displays it in extended mode.

Panning/Zooming : The user can pan and zoom on the workspace, and easily work on large configurations. It is also possible to enlarge and shrink individual modules.

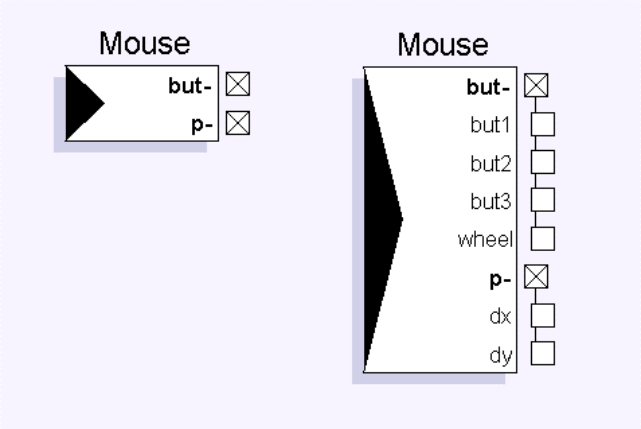

Figure 6: Hierarchical slots of the mouse device

Designing ICON Aware Applications

ICON changes the programming paradigm used by interactive systems. Current systems rely on an event based model, an event dispatching strategy and a state management programmed in a general purpose language such as Java or C. ICON uses another paradigm where values are propagated through a network of operations directly into actions. This paradigm is used extensively and successfully in automatic control systems, a domain we consider close to input configuration management.

Configurations in ICON are very similar to controllers in the traditional Smalltalk Model View Controller (MVC) triplet[17]. Externalizing the controller requires to externalize a protocol to communicate between a Model and a View. This protocol is modeled as input and output modules.

```
public class ToolModule extends Module {
// INPUT SLOTS
protected final IntSlot tool = new IntSlot("tool");
protected ToolBox toolbox;

public ToolModule(String name) {
  super(name);
  addInSlot(tool);
  toolbox = (Toolbox)getComponentNamed("ToolBox");
}

public void changed(Change change) {
  if (change.hasChanged(tool)) {
    toolbox.setTool(tool.getIntValue());
  }
}
}
```

Figure 7: Implementation of an input module displaying the currently selected tool in IconDraw.

Concretely, new modules have to be programmed in three circumstances: for a new application that needs to export some new Views and Models, when an unimplemented input device is available and when a new processing is required such as a gesture classifier. Programming a new

module involves modest efforts as shown in Figure 7. Existing modules have an average size of 50 lines, the largest being the speech recognition module, 512 lines long.

IMPLEMENTATION ISSUES

ICON is currently implemented in Java 1.2 and relies on Java Swing [10] with some specific extensions. Connections to the low level input APIs are usually implemented in C++ using the Java Native Interface [19]. The implementation is divided in three parts: the reactive language interpreter, the native modules API and the graphical editor ICON.

The Reactive Language Interpreter

The language underlying the execution of ICON is derived from Lustre and Esterel [4, 11]. Instead of defining a new semantics, we have relied on the well established synchronous reactive language semantics for the propagation of values and the flow control. Modules are like digital circuits and connections are like wires. Values are propagated during a clock tick that occurs at regular intervals or when a device requests it and at least one value has changed in the network. We have adapted and improved the value model of these languages by introducing hierarchical compound slots. A slot can either be of atomic type like integer or string, or by a collection of named slots. These compound or hierarchical slots simplify greatly the readability and construction of configurations. The interpreter is 4000 lines of source code long.

```
class Module {
    attribute String name;
    attribute boolean enabled;
    void addInSlot(InSlot s);
    void removeInSlot(InSlot s);
    List getInSlots();
    void addOutSlot(OutSlot v);
    void removeOutSlot(OutSlot v);
    List getOutSlot();
    boolean open(Frame f);
    protected boolean doOpen(Frame f);
    void close();
    protected void doClose();
    abstract void changed(Change c);
}
```

Figure 8: Module API for ICON.

The Module API

A module is implemented as a Java object with a list of input and output slots as shown in Figure 8. The interpreter calls the "changed" method of the object at each internal clock tick when at least one of its input slots has received a new value. The method can then compute new values for its output slots and the signal propagates through connections. New modules are simple to implement with this interface.

The Configuration Editor

The configuration editor modifies the reactive network interactively and relies on the reflection mechanisms of Java to expose module internals to the users. It represents about 4000 lines of source code.

DISCUSSION

The use of ICON on real program raises several issues that we discuss here. Among them, one can wonder about the kind of users who will use ICON, the expressive power of configurations edited with ICON and the practicality of the approach.

ICON Users

We don't expect all users to design large specific configurations of their applications with ICON. Instead, applications should come with a set of sensible configurations and users should usually modify small portions to suit their needs. However, with the possibility of sharing configurations with other users, we expect configurations to improve incrementally.

Some users may need special configurations, either because they have disabilities or because they have particular skills with some set of devices. These users will need the help of expert ICON users or programmers but still, they could adapt the programs to their abilities.

Expressive Power

ICON is a full language, although it doesn't allow run-time created modules or recursion. Considering the experience in the field of reactive synchronous languages, we have chosen to stay away from these run-time modifications. We also believe this is not a serious issue because users are not expected to build large dynamic configurations. As for the readability of dataflow systems, we haven't conducted experiments but they seem quite readable for the configurations we experienced. For larger configurations, the textual form of the ICON language could be better for some users, although compound modules and hierarchical slots enhance the readability by hiding details.

Practicality

ICON is currently in early stages: most of the specification of the interaction of interactive graphical applications can be configured outside the application but we haven't modified all the Swing components yet to be ICON aware. However, we have modified the Java Swing architecture to suit our needs and these modifications are quite small and not intrusive. We believe modern toolkits could benefit from this externalization of the interaction in term of modularity and extensibility. For example, the support of text input methods is currently implemented at a very low level in Java Swing and requires a complicated internal mechanism that could be more simply described using ICON. The same is true for some aspects of accessibility.

From an application programmer's point of view, interaction techniques can be exposed with any level of granularity and abstraction, providing a smooth transition path from no ICON support at all to full ICON support and extensibility.

CONCLUSION AND FUTURE DIRECTIONS

We have shown how ICON enables users to configure applications to available devices as well as to their skills. Such configurations previously required access to the source code of the application and were impossible to improve incrementally.

Configuring an application is important for disabled users, but also to users with special skills or working on a special environment (limited space, noisy, etc.) Furthermore, new devices or interaction modes can be tested and adapted to existing programs. This is even more important for combined devices and their interaction techniques. When a voice recognition software is used in conjunction with a pressure and tilt sensitive stylus, a large number of attributes are produced continuously and the best way to use them together has to be tested by trials and errors. Currently, this kind of testing can only be done by the application's programmers. ICON transfers this effort to any user who is motivated.

For future directions, we are currently implementing a C++ version of ICON's library to configure 3D virtual reality applications that require higher performance than Java can provide. We are also enriching our collection of supported input device managers to play with more exotic devices, including force feedback.

REFERENCES

1. Andrews, M. Accessibility and the Swing Set, Sun Microsystems Inc., http://java.sun.com, 1999.
2. Avid Inc. Channel Developer's Kit, Softimage Inc., 2000, www.softimage.com.
3. Axelson, J. *Usb Complete : Everything You Need to Develop Custom Usb Peripherals*. Lakeview Research, 1999.
4. Berry, G. and Cosserat, L., The synchronous programming languages Esterel and its mathematical semantics. in *Seminar on Concurrency*, (1984), Springer Verlag, 389–448.
5. Bier, E.A. and Freeman, S., 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, 79-86.
6. Carbone, M., Ensminger, P., Hawkins, T., Leadbeater, S., Lipson, E., O'Donnell, M. and Rajunas, J. NeatTool Tutorial, 2000, http://www.pulsar.org/neattools/.
7. Chatty, S., Extending a Graphical Toolkit for Two-Handed Interaction. in *Proceedings of the ACM Symposium on User Interface Software and Technology*, (1994), 195-204.
8. Ferguson, P. The X11 Input Extension: Reference Pages. *The X Resource*, *4* (1), 1992 195--270.
9. Frohlich, B. and Plate, J., The Cubic Mouse: A New Device for Three-Dimensional Input. in *Proceedings of ACM CHI 2000 Conference on Human Factors in Computing Systems*, (2000), 526-531.
10. Geary, D.M. *Graphic Java 2, Volume 2, Swing, 3/e*. Prentice Hall PTR, 1999.
11. Halbwachs, N., Caspi, P., Raymond, P. and Pilaud, D., The synchronous dataflow programming language Lustre. in *Proceedings of the IEEE*, (1991), IEEE, 1305-1320.
12. Hinckley, K. and Sinclair, M., Touch-Sensing Input Devices. in *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, (1999), 223-230.
13. Honeywell, S. *Quake III Arena: Prima's Official Strategy Guide*. Prima Publishing, 1999.
14. 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, http://www.cs.umd.edu/hcil/mid.
15. Jacob, R.J.K., Deligiannidis, L. and Morrison, S. A Software Model and Specification Language for Non-WIMP User Interfaces. *ACM Transactions on Computer-Human Interaction*, *6* (1), 1999 1-46.
16. Kovach, P.J. *Inside Direct3d*. Microsoft Press, 2000.
17. Krasner, G.E. and Pope, S.T. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, *1* (3), 1988 26--49.
18. LCS/Telegraphics. The Wintab Developers' Kit, LCS/Telegraphics,, http://www.pointing.com/WINTAB.HTM, 1999.
19. Liang, S. *The Java Native Interface : Programmer's Guide and Specification*. Addison-Wesley Publisher, 1999.
20. Microsoft Corporation. Microsoft Active Accessibility SDK 1.2, Microsoft Corporation, 1999.
21. Myers, B. The Garnet User Interface Development Environment. in *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, 1991, 486.
22. Myers, B.A. A New Model for Handling Input. *ACM Transactions on Information Systems*, *8* (3), 1990 289-320.
23. Myers, B.A., Lie, K.P. and Yang, B.-C., Two-Handed Input Using a PDA and a Mouse. in

Proceedings of ACM CHI 2000 Conference on Human Factors in Computing Systems, (2000), 41-48.

24. Nye, A. and O'Reilly, T. X Toolkit Intrinsics Programming Manual., *4*, 1993 567.

25. Sense8 Corp. The World Toolkit Manual, Sense8, 1999.

26. Sun Microsystems Inc. Java Speech API Programmer's Guide, Sun Microsystems Inc,, http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/index.html, 1998.