Squidy: A Zoomable Design Environment for Natural User Interfaces

Werner A. König
Human-Computer Interaction Group
University of Konstanz
Universitätsstrasse 10, Box D73
78457 Konstanz, Germany
Werner.Koenig@uni-konstanz.de

Roman Rädle
Human-Computer Interaction Group
University of Konstanz
Universitätsstrasse 10, Box D73
78457 Konstanz, Germany
Roman.Raedle@uni-konstanz.de

Harald Reiterer
Human-Computer Interaction Group
University of Konstanz
Universitätsstrasse 10, Box D73
78457 Konstanz, Germany
Harald.Reiterer@uni-konstanz.de

CHI 2009, April 4 – 9, 2009, Boston, MA, USA
ACM 978-1-60558-246-7/09/04.

Abstract
We introduce the interaction library Squidy, which eases the design of natural user interfaces by unifying relevant frameworks and toolkits in a common library. Squidy provides a central design environment based on high-level visual data flow programming combined with zoomable user interface concepts. The user interface offers a simple visual language and a collection of ready-to-use devices, filters and interaction techniques. The concept of semantic zooming enables nevertheless access to more advanced functionality on demand. Thus, users are able to adjust the complexity of the user interface to their current need and knowledge.

Keywords
Natural user interface, design environment, zoomable user interface, multimodal interaction, Squidy

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces – Graphical user interfaces, Input devices and strategies, Interaction styles, Prototyping; D.2.2 [Software Engineering]: Design Tools and Techniques – User interfaces.

Introduction

One of the emerging research fields in Human-Computer Interaction is concerned with the design and development of so-called post-WIMP user interfaces [8], such as tangible interaction, surface computing, gaze-based interaction as well as gestural and voice input (see Fig. 1-3). These technologies provide a richer set of interaction modalities than traditional mouse and keyboard input and break up with standard WIMP-based user interface concepts (Window, Icon, Menu, Pointing device). The recently coined term "Natural User Interfaces" (NUI) highlights the intention behind these novel interaction styles: they build upon users' pre-existing knowledge of the everyday, non-digital world [4] and hence lead to a more natural and reality-based interaction.

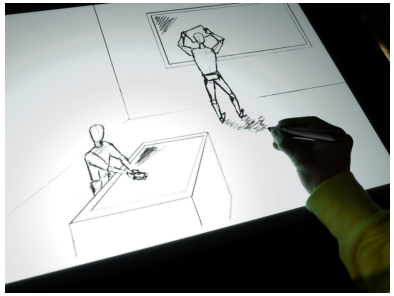

Figure 1: Digital pens benefit from users' pre-existing knowledge and thus offer a very natural mode of interaction e.g. for digital sketching and prototyping.

However, the design and development of NUIs is not only conceptually but also practically a very challenging task. In contrast to the design of traditional graphical user interfaces, it involves both software and hardware components [2]. Yet, conventional development environments (e.g. MS Visual Studio/.Net, Adobe Flash, Java) fall short of supporting uncommon input devices and appropriate data processing (e.g. computer vision) as well as the handling of multipoint and multi-user applications (e.g. for multi-touch interaction). To address this issue, a broad variety of heterogeneous and very specialized toolkits and frameworks have evolved over the last few years (e.g. Apple iPhone SDK, Microsoft Surface SDK, NUIGroup Touchlib, GlovePIE). Researchers and interaction designers therefore have to choose between these different toolkits, depending on the compatibility with the specific hardware being in use. This situation makes the design and development of NUIs a lot more demanding. Particularly, researchers and interaction designers have to face the following challenges:

- They need practical knowledge on different layers, ranging from hardware prototyping, drivers, protocols and signal processing, to application programming interfaces and the final application.
- They have to deal with different more or less monolithic tools as well as the according programming language and development environment.
- The resulting complexity restricts rapid prototyping and fast design iteration processes. This may also reduce comparability of the realized interaction techniques and concepts.

Few development environments are available which address these issues by supporting some novel input devices (e.g. physical turntables, mixing desks, multi-touch surfaces and simple vision tracking). Two examples are MAX/MSP [5] and vvvv [9]. Both are graphical development environments for music and video synthesis and are widely used by artists to realize interactive installations. Using the concept of visual dataflow programming both toolkits provide a simple way of defining and organizing the desired functionalities. However, the visual representation of each primitive variable, parameter, connection, and low-level instruction (e.g. matrix multiplication) lead to complex and scattered user interfaces, even for small projects. MAX/MSP and vvvv offer the possibility to encapsulate consecutive instructions in so-called "abstractions" or "subpatches". This approach helps to reduce the size of the visual dataflow graph, but introduces additional complexity by the hierarchical organization. Moreover, the multiplicity of the provided tools and add-ons as well as the tight coupling with visual interface

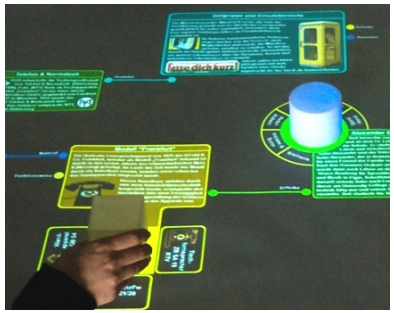

Figure 2: Multi-touch surface augmented with physical tokens reduces the gap between real-world and digital-world interaction.

Figure 3: Well-known devices such as an omnipresent laser pointer enhanced with button module, LEDs and vibration motor provide flexible input from any distance. Users benefit from a more natural and convenient pointing experience.

components and rendering further increase the complexity that users have to deal with. ICON Input Configurator [1], its successor MaggLite [3] and the OpenInterface Framework [7] are further development environments for post-WIMP user interfaces. They also employ the concept of visual dataflow programming and share the complexity issues with the frameworks described above. They partially reduce the visual complexity by encapsulating advanced functionalities in pre-defined modules which are textually developed with a conventional programming language. Thus, the user can apply these "black-box" modules in the visual user interface. However, one has to leave the environment in order to develop or change a module in source code. The user is therefore forced to switch between multiple development environments in favor of a decreased visual complexity.

To sum up, all of these development environments ease the implementation of NUIs by supporting some hardware devices and by providing a visual user interface to design and realize the desired interaction techniques. However, the complexity caused by crowded user interfaces (low-level visual dataflow programming) or by the need of additional tools, programming languages and compilers is still a major research issue. Furthermore, the demands on the designers' expertise are still very high, since they have to understand and route each primitive variable/data even when using "black-box" modules.

Squidy – Zoomable Design Environment

We address these issues with our interaction library "Squidy" which unifies various device toolkits and NUI frameworks in a common library and provides a central user interface for visual dataflow management as well as device and data filter configuration. Squidy thereby hides the complexity of the technical implementation from the user by providing a simple visual language and a collection of ready-to-use devices, filters and interaction techniques. This facilitates rapid prototyping and fast iterations. However, if more functionality and profound customizations are required, the visual user interface provides these on demand by using the concept of semantic zooming. Thus, users are able to adjust the complexity of the user interface to their current need and knowledge (ease of learning).

User Interface Concept

The basic concept which enables the visual definition of the dataflow between the input and output is based on the pipe-and-filter concept (Fig. 6). This offers a very simple, yet powerful visual language to design the interaction logic. The user thereby selects the input device or hardware prototype of choice as "source", e.g. a laser pointer, connects it successively with filter nodes for data processing such as compensation of hand tremor or gesture recognition and routes the refined data to the "sink". The filter nodes may transmit, change, delete data objects, or generate additional ones (e.g. if a gesture is recognized). The "sink" can be any output modality or device such as a vibrating motor for tactile stimulation or LEDs for visual feedback. Squidy also provides a mouse emulator as an output node to offer the possibility of controlling standard WIMP-applications with unconventional input devices. Multipoint applications (e.g. for multi-touch surfaces or multi-user environments) and remote connections are supported by an output node which transmits the interaction data either as TUIO messages or as basic OSC messages over the network. TUIO is a widely used protocol for multipoint interaction based on

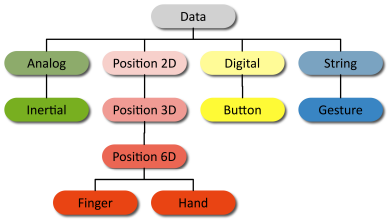

Figure 4: Squidy data type hierarchy based on primitive virtual devices [10].

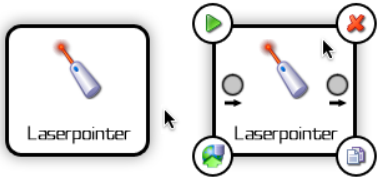

Figure 5: In order to reduce visual complexity the node-specific functions (active/inactive, delete, duplicate, publish to knowledge base) and the unconnected in and out ports are only shown if the cursor is inside the node.

the more general OpenSound Control protocol (OSC). The internal dataflow between the nodes in Squidy consists of a stream of single or multiple grouped data objects of well-defined data types (Fig. 4) based on the primitive virtual devices introduced by Wallace [10]. In contrast to the low-level approaches used in related work, such abstracting and routing of higher-level objects has the advantage that not every single variable has to be routed and completely understood by the user.

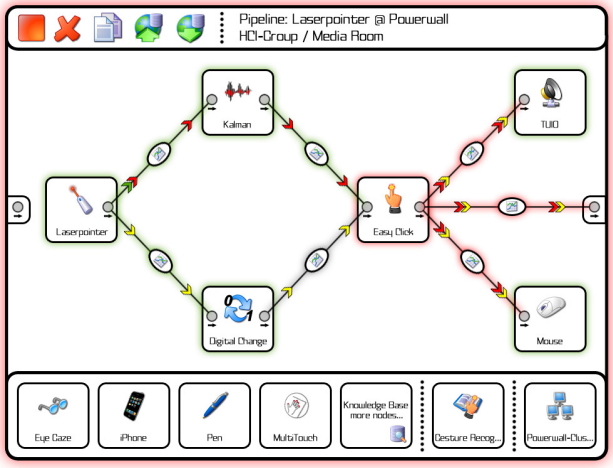

Figure 6: View of a zoomed pipeline in the Squidy Design Environment. The pipeline receives position, button and inertial data from a laser pointer, applies a Kalman filter, a filter for change recognition and a filter for selection improvement and finally emulates a standard mouse to interact with conventional WIMP-applications. The data is alternatively sent via TUIO to listening applications. The pipeline-specific functions and breadcrumb navigation are positioned on top. The zoomable knowledge base with a selection of recommended input devices, filters, and output devices are located at the bottom.

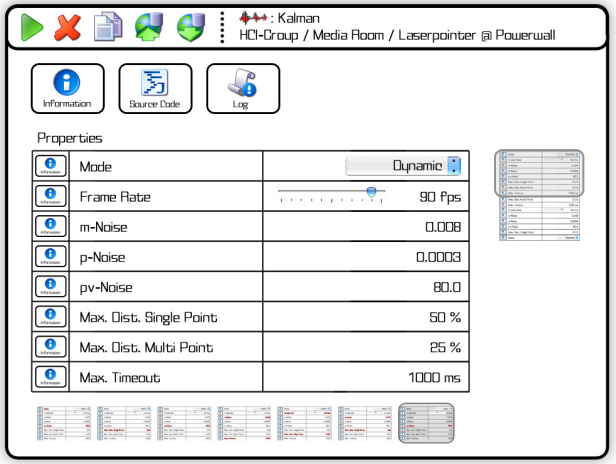

Figure 7: View of a zoomed Kalman filter node with table of parameters. Changes of parameters are immediately applied. Spatial scrolling with overview window (right) and temporal scrolling of last changes (bottom) is visually provided. The user can access further information (Fig. 8), the filter source code (Fig. 11) and node-specific logging by automatic zooming.

Knowledge Base

Squidy provides a wide range of ready-to-use device and filter nodes in an online knowledge base. An assortment of them is directly offered at the bottom of the pipeline view (Fig. 6). The selection and arrangement of the nodes depend on the statistics of previous usage and thus suggest suitable partners to the currently focused device or filter. This dynamic suggestion may lead to a higher efficiency but also helps novice users to limit the number of available nodes to a relevant subset. The user can directly drag a desired node from the selection (bottom) to the design space of the pipeline (center). If the desired node is not part of the suggested collection, the user has the

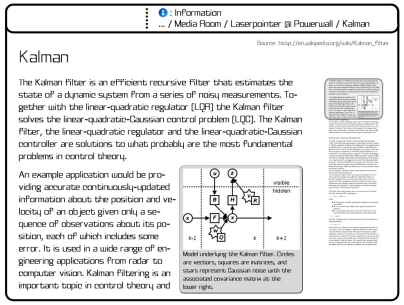

Figure 8: Zoomed information view of the Kalman filter node. The user gets illustrated (also images and videos) descriptions to the general functionalities. There are similar information views for each filter parameter located at the first table row (Fig. 7).

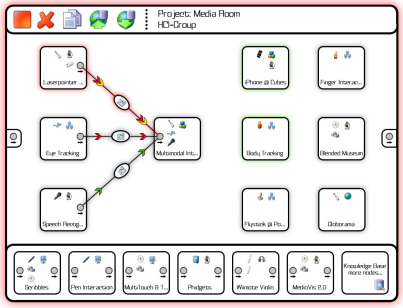

Figure 9: Project perspective overviews all pipelines within the project and offers vast possibilities in combining fragmented pipelines to sophisticated concepts (e.g. combination of eye-gaze, laser pointer and speech recognition to multimodal interaction).

possibility to access all nodes of the knowledge base by zooming into the corresponding view which is also located at the bottom.

Semantic Zooming

According to the assumption that navigation in information spaces is best supported by tapping into our natural spatial and geographic ways of thinking [6] we use a zoomable user interface concept to navigate inside the Squidy Design Environment. When zooming into a node, additional information and corresponding functionalities appear, depending on the real estate available (semantic zooming). Thus, the user is able to gradually define the level of detail (complexity) according to the current need for information. In contrast to the related work the user does not have to leave the visual interface and to switch to additional programming environments in order to generate, change or just access the source code of device drivers and filters. In Squidy, zooming into a node reveals all parameters and enables the user to interactively adjust the values at run-time (Fig. 7). This is highly beneficial for empirically finding suitable parameters for the current environment setting (e.g. Kalman filter: noise levels). Furthermore, the user can zoom into the information view which provides illustrated information about the node functionality itself and its parameters (Fig. 8). The user may even access the source code (Fig. 11) of the node by semantic zooming. Thus, code changes can be made in the visual user interface. If the user zooms out, the code will be compiled and integrated on the fly. As it is feasible to zoom into the source code a user may add new input and output devices or filters by adding an empty node and augmenting it with applicable code. In order to share the new node with the community the user can publish it into the knowledge base.

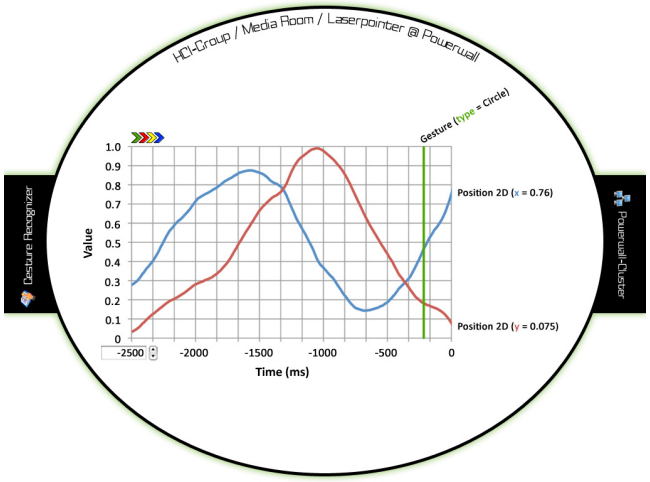

Figure 10: The user is able to visualize the current dataflow of a pipe by zooming into the ellipse located at it.

In the following list we want to sum up and emphasize some major characteristics of the Squidy interaction library and its zoomable design environment:

- **Multi-threading:** The possibility for multiple in and out connections provides high flexibility and the potential for massive parallel execution by concurrent nodes. Each node generates its own thread and processes its data independently as soon as it arrives. This effectively reduces the processing delay that could have a negative effect on the interaction performance.
- **Reusability & comparability:** Nodes are completely independent components, offer high reuse, are free from side effects, and can be activated separately e.g. for comparative evaluations.

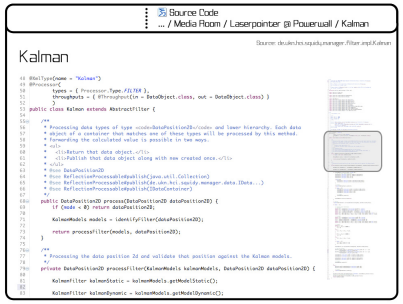

Figure 11: Source Code of the corresponding device or filter node is directly accessible by semantic zooming. Zooming-out leads to runtime compilation of the source code and live integration into the current pipeline.

- **Less demanding:** Semantic zooming enables users to adjust the complexity of the user interface to their current need. Moreover, users may use filters and devices as “black boxes” without any knowledge of the technical details and thus concentrate on the design.
- **Dataflow visualization:** The visual inspection of the current dataflow assists to identify possible issues and facilitates fast error correction at runtime (Fig. 10).
- **Interactive configuration:** Changes in the dataflow and configuration of node parameters results instantly in changes concerning the NUI interaction. This supports fast and interactive design iterations.
- **Visual interaction design:** The pipe-and-filter concept augmented with semantic zooming offers a very simple, but powerful visual language for the design and development of natural user interfaces.

Conclusion and future work

Squidy combines various input and output devices, data filters and interaction techniques in a common interaction library, empowering researchers and interaction designers to visually design novel interaction concepts. Up to now, the Squidy Design Environment does not provide multi-user support. This and the integration of version controlling will be future work. Furthermore, we will conduct qualitative usability tests in order to validate and inform the design of the Squidy user interface concept. We plan to go open-source. Until then, contact authors for a trial version.

Acknowledgements

This work is supported by DFG GK-1042 "Explorative Analysis and Visualization of Large Information Spaces" and the project "Interactive Visualization for Gigapixel Displays" supported by the "Information Technology Baden-Württemberg (BW-FIT)" program.

References

[1] Dragicevic, P., Fekete, J-D. Input Device Selection and Interaction Configuration with ICON. In *Proc. IHM-HCI 2001*, Springer Verlag (2001), 543-558.

[2] Harper, R., Rodden, T., Rogers, Y., Sellen, A., Being Human: Human-Computer Interaction in the Year 2020. Microsoft Research Ltd., Cambridge, 2008.

[3] Huot, S., Dumas, C., Dragicevic, P., Fekete, J., and Hégron, G. The MaggLite post-WIMP toolkit: draw it, connect it and run it. In *Proc. UIST '04*, ACM (2004), 257-266.

[4] Jacob, R. J., Girouard, A., Hirshfield, L. M., Horn, M. S., Shaer, O., Solovey, E. T., and Zigelbaum, J. Reality-based interaction: a framework for post-WIMP interfaces. In *Proc. CHI 2008*, ACM Press (2008), 201-210.

[5] Max/MSP/Jitter, Cycling ’74. http://www.cycling74.com/.

[6] Perlin, K. and Fox, D. Pad: an alternative approach to the computer interface. In *Proc. SIGGRAPH '93*, ACM (1993), 57-64.

[7] Serrano, M., Nigay, L., Lawson, J. L., Ramsay, A., Murray-Smith, R., and Denef, S. The OpenInterface Framework: A tool for multimodal interaction. *In Proc. CHI '08 Extended Abstracts,* ACM (2008), 3501-3506.

[8] Van Dam, A. Post-WIMP user interfaces. *Commun. ACM 40,* 2 (1997), 63-67.

[9] vvvv: a multipurpose toolkit, vvvv group, http://vvvv.org/tiki-index.php.

[10] Wallace, V. L. The semantics of graphic input devices. In *Proc. SIGGRAPH’76*, ACM (1976), 61–65.