Immersive Mixed-Reality Configuration of Hybrid User Interfaces

Christian Sandor
TU München, Institut für Informatik
sandor@in.tum.de

Alex Olwal
Royal Institute of Technology, NADA
alx@kth.se

Blaine Bell, Steven Feiner
Columbia University, Department of Computer Science
{blaine,feiner}@cs.columbia.edu

Abstract

Information in hybrid user interfaces can be spread over a variety of different, but complementary, displays, with which users interact through a potentially equally varied range of interaction devices. Since the exact configuration of these displays and devices may not be known in advance, it is desirable for users to be able to reconfigure at runtime the data flow between interaction devices and objects on the displays. To make this possible, we present the design and implementation of a prototype mixed reality system that allows users to immersively reconfigure a running hybrid user interface.

1 Introduction

In *hybrid user interfaces* [2, 3, 10], information can be distributed over a variety of different, but complementary, displays. For example, these can include stationary, opaque displays and see-through, head-worn displays. Users can also interact through a wide range of interaction devices. In the unplanned, everyday interactions that we would like to support, we would not know in advance the exact displays and devices to be used, or even the users who would be involved. All of these might even change during the course of interaction. Therefore, a flexible infrastructure for hybrid user interfaces should automatically accommodate a changing set of input devices and the interaction techniques with which they are used. This paper presents the first steps toward building a mixed-reality system that allows users to configure a hybrid user interface.

A key idea underlying our work is to immerse the user within the authoring environment. Immersive authoring has been explored by Lee and colleagues [5], in a system that has a wider range of possible parameters than we currently support. While their system is restricted to a single view and interaction with real objects is limited to AR Toolkit[1] markers, our system supports multiple coordinated views with different visualizations and interaction with a variety of physical controllers.

As a first step towards allowing end users to configure hybrid user interfaces, we have realized a specific scenario, which we support with an augmented reality overlay, presented on a head-tracked, see-through, head-worn display. In our scenario, a user interacts with physical input devices and 3D objects drawn on several desktop displays. The input devices can be configured to perform simple 3D transformations (currently scale, translation, and rotation) on the objects. The user's see-through head-worn display overlays lines that visualize data flows in the system, connecting the input devices and objects, and annotates each line with the iconic representation of its associated transformation. The user wields a tracked wand with which she can reconfigure these relationships, picking in arbitrary order the three elements that comprise each relationship: an input device, a 3D object, and an operation chosen from a desktop menu.

While we have designed our initial scenario to be simple, we would ultimately like to support users in their daily tasks. For example, a typical apartment contains functionality that a user might want to control (e.g., the volume of a living-room audio system or the brightness of the lights in a bedroom). Additionally, the user might own several pocket-sized devices that she might want to use to control the apartment. There are several possible reasons why a user might want to reconfigure the way this functionality is controlled. First, the apartment will be used in different contexts (e.g., the user is home alone, is having guests for dinner, or she is having a cocktail party). Second, the user might buy new input devices or new devices to control. Both cases require the user to reconfigure the mapping of input devices to the controlled functionality.

Thus, the objects used in our scenario are placeholders

[1] `http://www.hitl.washington.edu/artoolkit`

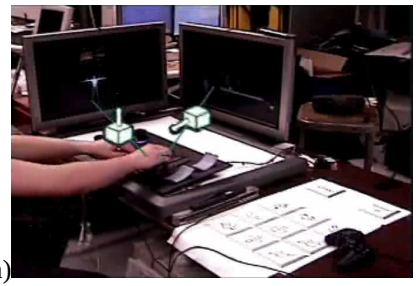
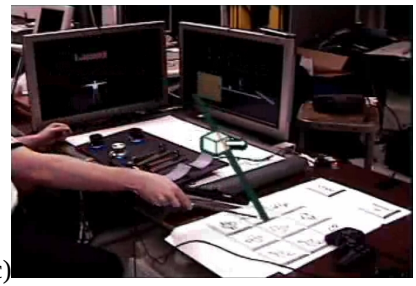
(a)  (b)  (c) 

Figure 1. Videomixed view through another user's tracked, see-through, head-worn display. (a) Lines show the data flow between tracked input devices and virtual objects. (b) Untracked input devices are shown as screen-stabilized models. (c) A tracked wand is used to reconfigure the data flow network.

for aspects of an application a user might want to manipulate, while the 3D transformations are placeholders for more general operations a user could perform on them.

There is a long history of interactive control of animation with desktop devices, which Laszlo and colleagues review [4]. There are also many examples of configuring interaction devices at runtime to support a wide range of tasks, from database queries [12] to musical performance [9]. Our work differs from these in that we apply an immersive augmented reality user interface to both interactively specify and visually document the system data flow.

2 Interaction Design

We address two general issues in designing a reconfigurable user interface: presenting appropriate feedback and supporting interactive reconfiguration.

2.1 Visual Feedback

We developed a simple graphical language to visualize the relationships between objects, the input devices that control them, and the associated operation. This provides the user with an intuitive overview of the active mappings in the environment.

A real object (or its virtual representation) is visually connected to a controlling input device through a line that can be seen in the head-worn display. An iconic representation of the currently assigned operation is attached to the line. Figure 1(a) shows a view of the interaction and its overlay as seen from the vantage point of another user. To avoid clutter, we show a relationship's visualization only while the user manipulates its associated input device or reconfigures its mapping, as described below.

Some of our input devices are not tracked, and, therefore, their locations are unknown. We represent an untracked input device by a screen-stabilized image (fixed to the coordinate space of the head-worn display), and draw a line from the appropriate part of that device to the virtual object it controls, as shown in Figure 1(b).

2.2 Interactive Reconfiguration

The user can interactively create or modify existing relationships with a tracked wand, as shown in Figure 1(c). To establish a relationship between a physical device and a virtual object, three attributes need to be chosen (in any order): an operation, a physical device, and a target virtual object. The physical device and target virtual object are selected by moving the wand within the proximity of a physical object or a virtual object's projection on a physical display, triggering highlighting and auditory feedback. The operation is selected by moving the tip of the wand to one of the operations displayed on a printed 3×3 grid at a known location on the desk. Our menu, inspired by the printed wall-mounted menu of [13], allows the specification of translation, rotation, and scale along the *x*, *y*, or *z* axis.

Table 1 represents all possible states that can occur when a relationship with a tracked device is being configured: the input (target object, input device, and operation) is mapped to how the connection is displayed (the beginning and end points of the line, and whether the icon is shown). For example, when both the target object (highlighted on the screen) and the operation (displayed as an icon) are selected, as shown in Figure 1(c), the user still needs to select the input device.

Table 1. Visual feedback provided during reconfiguration of tracked devices. $wand_{base}$ is the wand's base, $wand_{tip}$ is the wand's tip, $device$ is the physical input device and $target$ is the selected virtual object.

	input			line feedback		
	target	device	operation	begins	ends	icon
1			×	$wand_{base}$	$wand_{tip}$	×
2		×		$wand_{tip}$	device	
3		×	×	$wand_{tip}$	device	×
4	×			$wand_{tip}$	target	
5	×		×	$wand_{tip}$	target	×
6	×	×		device	target	
7	×	×	×	device	target	×

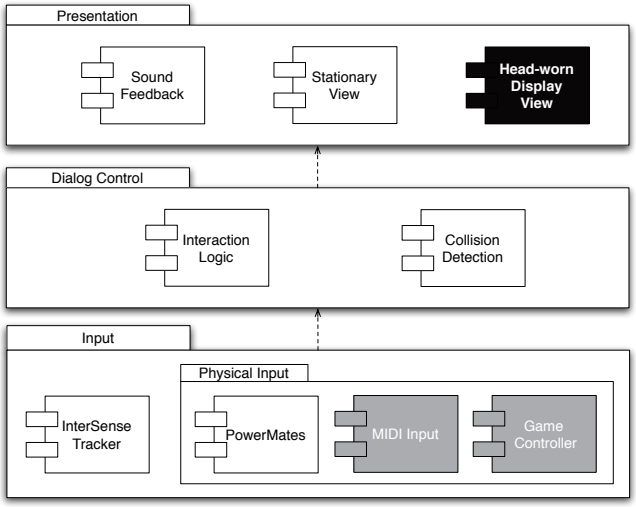


Figure 2. Mapping of components to subsystems. White components are implemented in DWARF, light grey components in Unit, and dark grey components in DP.

Therefore, a line is drawn from the tip of the wand to the target object, as specified in the fifth row of Table 1.

For untracked controllers, touching a "Learn" button causes the next device manipulated by the user to be selected for assignment. (In contrast to our fixed printed menu, we are experimenting with projecting the Learn button on the desk such that it automatically avoids being occluded from the user's viewpoint by the tracked board [1].)

3 Implementation

Our prototype is built using a set of existing frameworks. The overall architecture and most of the components are taken from the DWARF user interface framework [11], the input device handling is inspired by Unit [8], and the material presented on the head-worn display uses the DP (data programming) framework [1], as shown in Figure 2. We chose this mapping based on the strengths of each framework. The head-worn display view relies on the ability to easily specify rules in DP, while the many input devices supported by Unit made it a natural match for the input device drivers. The remaining components are based on existing DWARF components (with only the PowerMates component implemented from scratch).

The data flow between the components relies on two different mechanisms. Within each framework, we use the framework's native communication protocol: CORBA (Common Object Request Broker Architecture) events for DWARF, and UDP (User Datagram Protocol) for Unit and DP. Across framework boundaries, we also use UDP, which has proven to be a simple, yet viable solution. We run Unit and DP frameworks on Windows XP Professional, and DWARF on SuSE Linux Professional 9.1.

Figure 3. Input devices used in our prototype: (a) Dance mat. (b) Game controller. (c) Wand with attached InterSense 6DOF tracker. (d) Tracked board with PowerMate sensors (left) and MIDI sensors (right): sliders and bend sensors attached to playing cards.

Our components include:

Stationary Views. Our two stationary views reuse DWARF's Open Inventor-based Viewer component [11]. We extended this component to send 2D screen-space bounding box information for objects to the collision-detection component, to allow collisions between the wand and objects on the screens to be detected. This information is also sent to the head-worn display view, enabling it to visualize relationships (through overlaid lines) between objects on the screens and input devices.

Head-worn Display View. Each head-worn display view uses the DP framework, which can efficiently implement tabular mappings, such as those of Figure 1.

Interaction Logic. The interaction logic is modeled within DWARF's User Interface Controller component [11]. Petri nets are used to specify and execute the interaction logic, in the spirit of earlier work on User Interface Management Systems [7].

Collision Detection. This DWARF component is based on the Euclidean distance of the center points of tracked physical objects and the centers of the 2D screen-space bounding boxes of virtual 3D objects.

InterSense Tracker. 3D tracking is implemented using InterSense IS-900 and IS-600 trackers. This component is a straightforward wrapper for the InterSense native library.

PowerMates. Our tracked board includes several Griffin PowerMate[2] rotary sensors.

MIDI Input Devices. The availability of many different MIDI (Musical Instrument Digital Interface) input devices motivated us to support them in our system. We use the Unit framework to encapsulate MIDI functionality into a separate Java-based library. An A/D converter[3] converts analog sensor data to 7-bit MIDI data, making it easy to support bend sensors and sliders.

Game Controllers. To accommodate game controllers, we

[2] `http://www.griffintechnology.com/products/`
[3] `http://infusionsystems.com/catalog/index.php`

use a platform-specific library for low-level interfaces to peripheral devices through the RAWINPUT functionality in Windows XP. We support any number of Windows-compatible game pads or joysticks, including the Microsoft Sidewinder FreeStyle Pro gamepad, which has two accelerometers for sensing pitch and roll. By using two types of USB adapters that support up to four PlayStation controllers, we exploit the wide range of controllers available for the PlayStation platform, including a generic 1m×1m dance pad with 14 buttons and an analog controller with buttons and two analog joysticks.

4 Conclusions and Future Work

Our hybrid user interface allows the user to manipulate objects through whole-body interaction (on the dance pad), manual input devices (knobs, sliders, and bend sensors), and game controllers. We support interactive end-user reconfiguration of the mapping between devices, objects, and operations. Using a head-tracked, see-through display, we provide overlaid visual documentation of the system's current configuration and overlaid visual and auditory feedback as the system is reconfigured.

Currently, our system supports only a fixed number of operations. We are exploring how we can extend it to allow users to specify new operations at runtime, such as model deformation. While we anticipate using programming by demonstration [6] to address a carefully planned universe of possibilities, supporting arbitrary operations through demonstration and generalization is an open problem. A more pragmatic approach to increase coverage would be to use the Python services in DWARF [11], since Python is well suited for rapid development by end-users who can program. We are also interested in extending our mappings to permit their parameters (e.g., scale factors) to be modified interactively, employing techniques similar to those used in Unit [8]. Additional enhancements that we plan include support for grouping and selecting multiple devices, operations, and objects, along with the ability to load and save configurations. For example, we would like to make it easy for a user to select a single device and specify that it controls multiple operations (e.g., scaling in x, y, and z) on a group of objects.

As we extend our user interface, we will be designing a formal user study to validate our approach, benefiting from the informal user feedback that we gathered when demonstrating earlier versions of the system. After receiving a brief explanation of how to use the system, these early users found it easy to reconfigure the system themselves, and told us that they appreciated the overlaid visual and audio feedback provided during reconfiguration. Based on user feedback, we replaced textual annotations on the lines (visible in the first segment of the accompanying video) with the iconic representations of operations that we currently use. However, an overview visualization that showed all of the relationships simultaneously, proved too confusing to be useful because of the visual clutter caused by overlapping lines and icons. Therefore, we removed it from the current version of the system. To address this problem, we are interested in applying view-management techniques [1], information filtering, and dynamic graph layout to improve the way in which these relationships are displayed.

Acknowledgements

This research was funded in part by Office of Naval Research Contract ONR N00014-04-1-0005. Christian Sandor's stay at Columbia University was partly sponsored by a scholarship from the Deutscher Akademischer Auslandsdienst. We also wish to thank Surabhan "Nick" Temiyabutr for assembling the sensor board and Sinem Güven for recording the voice feedback prompts.

References

[1] B. Bell. *View Management for Distributed User Interfaces*. PhD thesis, Department of Computer Science, Columbia University, New York, NY, 2005.

[2] A. Butz, T. Höllerer, S. Feiner, B. MacIntyre, and C. Beshers. Enveloping users and computers in a collaborative 3D augmented reality. In *Proc. IEEE and ACM IWAR 1999*, pages 35–44, San Francisco, CA, October 20–21, 1999.

[3] S. Feiner and A. Shamash. Hybrid user interfaces: Breeding virtually bigger interfaces for physically smaller computers. In *Proc. ACM UIST '91*, pages 9–17, Hilton Head, SC, November 11–13, 1991.

[4] J. Laszlo, M. van de Panne, and E. Fiume. Interactive control for physically-based animation. In *Proc. SIGGRAPH 2002*, pages 201–208, New Orleans, LA, 2002.

[5] G. A. Lee, C. Nelles, M. Billinghurst, and G. J. Kim. Immersive authoring of tangible augmented reality applications. In *Proc. IEEE and ACM ISMAR '04*, pages 172–181, Arlington, VA, November 2–5, 2004. IEEE Computer Society.

[6] H. Lieberman, editor. *Your Wish Is My Command*. Morgan Kaufmann Publishers, 2001.

[7] D. Olsen. *User Interface Management Systems: Models and Algorithms*. Morgan Kaufmann Publishers, 1992.

[8] A. Olwal and S. Feiner. Unit: Modular development of distributed interaction techniques for highly interactive user interfaces. In *Proc. GRAPHITE '04*, pages 131–138, Singapore, June 15–18, 2004. ACM Press.

[9] I. Poupyrev. Augmented groove: Collaborative jamming in augmented reality. In *SIGGRAPH 2000 Conference Abstracts and Applications*, page 77. ACM Press, 2000.

[10] J. Rekimoto and M. Saitoh. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *Proc. CHI 1999*, 1999.

[11] C. Sandor and G. Klinker. A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality. In *Journal for Personal and Ubiquitous Computing*. Springer Verlag, 2004.

[12] B. Ullmer, H. Ishii, and R. Jacob. Tangible query interfaces: Physically constrained tokens for manipulating database queries. In *Proc. INTERACT '03*, Zurich, Switzerland, September 1–5, 2003.

[13] D. Vickers. *Sorcerer's Apprentice: Head-Mounted Display and Wand*. PhD thesis, Department of EE, University of Utah, Salt Lake City, UT, 1972.