HIPerFace: a Multi-channel Architecture to Explore Multimodal Interactions with Ultra-Scale Wall Displays

Nadir Weibel[1,3]
weibel@ucsd.edu

Arvind Satyanarayan[1,4]
arvind@ucsd.edu

Amanda Lazar[1,5]
alazar@ucsd.edu

Reid Oda[1,3]
roda@ucsd.edu

So Yamaoka[2,4]
syamaoka@ucsd.edu

Kai-Uwe Doerr[2]
kdoerr@ucsd.edu

Falko Kuester[2,4]
fkuester@ucsd.edu

William G. Griswold[4]
wgg@ucsd.edu

James D. Hollan[1,3,4]
hollan@ucsd.edu

[1]Distributed Cognition and Human Computer Interaction Laboratory
[2]Gravity Laboratory, California Institute for Telecommunications and Information Technology
[3]Department of Cognitive Science
[4]Department of Computer Science and Engineering
[5]Department of Electrical Engineering
University of California San Diego, La Jolla CA, 92093, USA

ABSTRACT

Interacting with large high-resolution wall displays is an emerging challenge. While new interface and device technologies encourage multimodal multiperson interaction, this strains the capabilities of current GUI architectures. In this paper we present HIPerFace, a multi-channel architecture supporting the integration of multiple tangible devices and enabling experimentation with multi-device, multimodal interaction. To provide this capability, HIPerFace implements a three-layer architecture employing a multi-stage hybrid adapter/observer design pattern. We describe this architecture, how we addressed unique challenges by adapting state-of-practice techniques, and discuss lessons learned.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Patterns, Domain-specific architectures*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Input devices and strategies, Prototyping*

General Terms

Design, Experimentation

Keywords

Software Engineering practices, Design patterns, Human Computer Interaction, Multimodal interactions, Multi-channel input

1. INTRODUCTION

The decreasing cost of displays and growing power of graphics processing units (GPUs) are enabling visualization and analysis of increasingly large multidimensional data sets on tiled wall displays. The scale of these data sets prevents them from being readily viewed or manipulated on desktop or simple projection displays. High-resolution wall displays promise to be particularly useful for visualization of such data sets, allowing detection of patterns that would otherwise be impossible to see on current desktop displays. However, there are still several systemic challenges to be addressed. One pervasive challenge is how to help users interact with content in distant regions of the display [14]. Another is how to support interactions for multiple users working in collaboration [40]

The CalIT2 Highly Interactive Parallelized Display Space (HIPerSpace[1], – Fig. 1) is a 31.8 foot wide by 7.5 foot tall wall display. Seventy tiled 30" Dell LCD displays provide a resolution of 35,840 x 8,000 pixels (286,720,000 pixels total). The wall is powered by 18 Dell XPS 710/720 computers with Intel quad-core central processing units and dual nVIDIA FX5600 GPUs. A head node and six streaming nodes complete the hardware configuration for a total of 100 processor cores and 38 GPUs.

[1]http://vis.ucsd.edu/projects/hiperspace

Figure 1: The HIPerSpace wall display

ICSE 2011 Waikiki, Honolulu, Hawaii, USA

The system is powered by CGLX [12] (Cross-Platform Cluster Graphic Library), a flexible, high-performance OpenGL Graphics framework. Two 24" displays, positioned on the opposite side of the room, provide control for the HIPerSpace wall. Until recently, users interacted with the wall display and digital content through a standard keyboard and a single gyroscopic mouse. Recent development of the CGLX interface introduced support for interaction with single tangible interface components such as Nintentdo Wii and PS3 Controllers, Apple iPad, iPhone and Android smartphones, 3D tracking systems and multitouch tables.

Such individual input devices have been investigated as mechanisms for interacting with large displays, and research in Human Computer Interaction (HCI) has demonstrated that they can be beneficial for specific tasks [3, 37, 33, 14, 6]. However, neither a single technique nor a single device is likely to be appropriate for all activities and contexts. In addition, large wall displays are especially appropriate for multiperson interaction. It is important to explore and evaluate new ways of interacting with settings like the HIPerSpace wall to simplify interaction and make it more natural. Creating a flexible and extensible architecture to support exploration of a variety of input devices and interaction techniques is the primary objective of our current research. We are particularly interested in enabling *simultaneous* use of multiple input devices to allow natural multiperson interaction.

In order to achieve this goal we implemented HIPerFace, a Java-based software infrastructure[2] that integrates multiple input and output devices and supports exploration of *composite* multi-device interactions. We exploit a range of well-known software engineering practices, design patterns, and principles. Employing a layered interface to abstract device specific events, we use a hybrid *adapter/observer* design pattern to abstract native device events into corresponding device independent *atomic* events. Using the same pattern, atomic events are further transformed into *composite* events to represent more "complex" interactions. However, to be displayed on output devices, composite events must be transformed back into appropriate native device events. This series of transformations is serviced by three components – *producers*, *interpreters* and *consumers* – in a sequence of hybrid *adapter/observer* objects, crossing three layers of abstraction. The resulting HIPerFace infrastructure connects the HIPerSpace wall display with multiple tangible devices such as multitouch tables and displays, digital pens and smartphones.

In this paper we introduce, describe, and discuss the HIPerFace architecture and its use with the HIPerSpace wall display. Section 2 introduces related architectures, systems, and frameworks supporting the handling of multi-channel input data, tangible interactions, and multimodal interactions. Section 3 describes the HIPerFace architecture in terms of its layers and components and how it supports combining multiple input channels. Section 4 discusses the challenges we experienced while building specific components of HIPerFace and, particularly, several tradeoff decisions that had to be made to support interactivity. Section 5 outlines the specific setting in which HIPerFace has been introduced, presenting the input devices that have been connected to it and highlighting a specific case study of their integration. Section 6 summarizes the software engineering contribution.

[2]Using Java allowed us to exploit existing SDKs recently made available to interact with different tangible devices

2. TANGIBLE INTERACTIONS

As physical artifacts gain new computational components, they become programmable, customizable and interoperable. A variety of approaches have been introduced to control and provide more flexible interactions with such tangible devices. Inspired by the early work of Ishii and Ullmer [22, 38] and others [32, 16], techniques for managing information across multiple physical and digital devices have been extensively investigated. For example, the iServer infrastructure enables management of pervasive cross-media applications based on physical hypermedia [35], iStuff [2] allows multiple users to interact with the same information space with multiple devices, and Papier-Mâché [26] supports development of interfaces that mix tangible interaction with other modalities, such as speech or gesture based interaction. Recently, several frameworks and toolkits have been developed for interacting with physical and tangible devices such as Shared Phidgets [29] or the VodooIO platform [39].

Tangible touch-based interfaces that allow the tracking of fingers and objects placed on interactive displays are of increasing importance. Innovative technologies for simultaneously sensing multiple touch points in a two-dimensional space, e.g., ThinSight [21], Frustrated Total Internal Reflection (FTIR) approaches [19], and DiamondTouch [11], or in a three-dimensional space such as LightSpace [43], are being combined with techniques to detect physical objects, such as in the reacTable [25] or SLAP widgets [42], to enable design of a wide variety of novel applications. The first commercial touch-based products such as the Apple iPhone, iPod Touch or iPad, Microsoft Surface[3], the Multi-Touch Wall[4] and other multitouch LCD displays are now available.

Paper continues to be an important physical interface. By using an almost invisible pattern printed on paper to record digital pen movements, Anoto technology[5] allows interactions with paper documents to be captured and tracked. Several frameworks have been developed to link paper and digital worlds: PADD [17], iPaper [30], and PaperToolkit [44]. To better support pen and paper interaction these frameworks are being augmented with gesture-based facilities like those of PapierCraft [27] and iGesture [34]. Others [18, 8, 7, 5] have begun to combine digital pen and paper with touch-based interactive tables. The HephaisTK system and the SMUIML language[13] have investigated multimodal interaction with multiple devices including input devices such as RFID readers and DiamondTouch tables as well as modalities such as speech and keyboard input.

Existing frameworks and systems for digital-physical management of information increasingly involve novel tangible interfaces. However they do not currently support flexible integration of multimodal multi-channel *composite* interactions. In the remainder of this paper we outline how the

[3]http://www.microsoft.com/surface
[4]http://www.perceptivepixel.com
[5]http://www.anoto.com

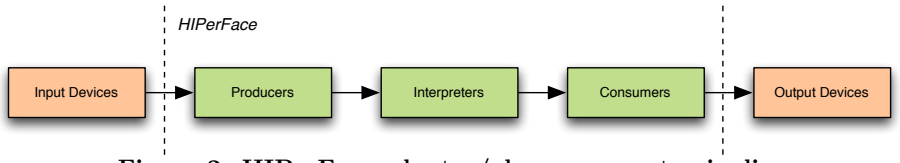


Figure 2: HIPerFace adapter/observer events pipeline

HIPerFace infrastructure approaches this issue through design and implementation of a three-layer architecture and use of a combined multi-stage adapter and observer design patterns.

3. THE HIPERFACE ARCHITECTURE

HIPerFace is based on a flexible architecture that supports the simultaneous use of multiple interface devices and was specifically designed to facilitate rapid prototyping and experimentation with different device collections. We exploit existing SDKs and device drivers to enable communication with the core framework from a range of hardware devices such as touch-based displays, multitouch tables, and digital pens. Events generated by a device are transmitted to the HIPerFace infrastructure and then combined and composed. HIPerFace is inserted between the devices and the applications, intercepting device events and delivering them directly to the applications via pre-defined plugins or APIs. In cases where the application lacks an event-based API for receiving events, we route the events back to the window system, which then routes the events to the application in the standard way. Following a variant of a model-view-controller paradigm, the results of interactions are events sent to output devices.

HIPerFace employs a three-tier layered event architecture consisting of *native*, *atomic*, and *composite* events that is coupled with a three-component multi-threaded architecture of *producers*, *interpreters* and *consumers*. As illustrated by Fig. 2, it follows a chained adapter/observer design pattern [15] forwarding events received from the native input devices to the target output devices. Figure 3 outlines the overall structure of HIPerFace and highlights the interplay between the different components across the layers. The three layers and the three-component architecture enable extensibility and allow experimentation with how multiple devices might flexibly work together. Each stage of the architecture deals with a unique issue, as defined by the layers, raising the level of abstraction on the "way up", and lowering the level of abstraction on the "way down".

At the native layer, different input devices generate a stream of *native device events* (NDEs). Producers are then responsible for making the first transition by providing two main

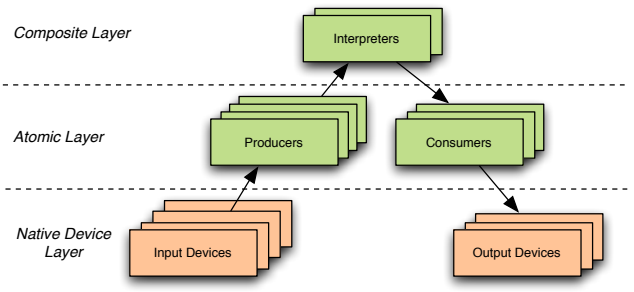


Figure 3: HIPerFace layered architecture

functions. First, they provide a connector interface to allow NDEs to be captured by HIPerFace, and second, via an adapter interface, they abstract NDEs into one of three *atomic HIPerFace events* (AHEs): `DownEvent`, `UpEvent` or `MoveEvent`. AHEs can be thought of as "basic" interactions and were introduced to establish a common foundation for later layers, regardless of which input device the NDE originates. Once a producer has successfully adapted an NDE into an AHE, it notifies registered interpreters, which perform the next transformation from atomic to *composite HIPerFace events* (CHEs) – high-level events that define more complex interactions such as *scaling* or *rotating*. Interpreters *fire* CHEs to registered consumers that ultimately translate them back to NDEs compatible with selected output devices, functioning essentially as an inverse of the producers' function.

As later subsections will describe, the components in HIPerFace have been implemented by combining multi-stage adapter and observer design patterns [15]. The first phase consists of interpreters individually registering with producers (observer) and transforming the received AHEs into CHEs. The second phase consists of consumers registering with interpreters (observer) and wrapping CHEs into NDEs.

3.1 Producers

A producer serves two functions. First, it connects an input device with HIPerFace either through interfaces provided by a device's native SDK or, if this is not available, an open socket with the device transmitting text representations of its events to it. Second, a producer abstracts received NDEs – via an adapter – into AHEs. The class hierarchy (Fig. 4) depicts how the `NativeConnector` and `ProducerAdapter` interfaces expose the methods required to collect native device events and abstract them into atomic HIPerFace events. We implement them using the abstract `HIPerFaceProducer` class in order to establish a common framework for all producers. If a producer needs to open a socket because a `NativeConnector` is not available for the device, an implementation of an instance of `ServerConnector` for that device is required. As we will outline in Sect. 5, HIPerFace is coupled with a range of tangible devices such as iPhones, multitouch displays, touch-based interactive tables and digital pens. For each of the supported devices we implemented specific producers, as shown in the bottom part of Fig. 4.

Abstracting NDEs to AHEs serves two purposes: it allows the system to be readily extended to support new input devices, and it allows composite events to be constructed with a common base language. We quickly noticed that our initial abstraction mechanism resulted in loss of crucial metadata associated with a device's native events. As a result, we augmented HIPerFace events to be parameterized generic containers. Each has a common list of data about the event – for example, the coordinates of the event – but also can represent device-specific metadata – for example, the pressure

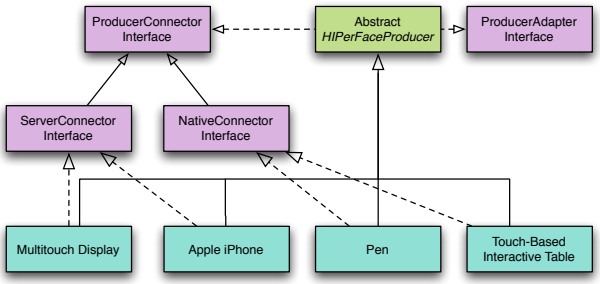


Figure 4: HIPerFace producers class hierarchy

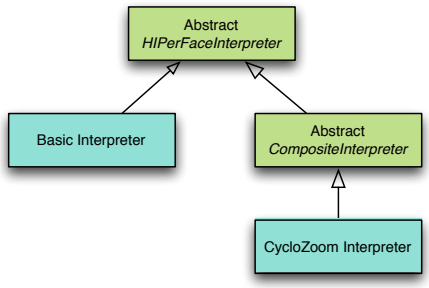


Figure 5: HIPerFace interpreter class hierarchy

with which a pen nib was pressed onto a surface. This allows capture of device-specific event metadata without sacrificing extensibility, retains the identity of the device from which a HIPerFace event originated, and, depending on the originating device, permits treatment of specific events in different ways when building composite events in an interpreter.

A producer can adapt NDEs into the following ADEs:

- `DownEvent` – when a device goes into the "down" state, for example, when a mouse or keyboard button is pressed down, when the pen nib makes contact with paper or when a finger touches a multitouch surface.
- `UpEvent` – when a device releases the down state, for example when a mouse or keyboard button is released or when the pen nib or finger is lifted off.
- `MoveEvent` – when a device causes a cursor to move. Within this event, we also track the "up or down" context of the event to be able to differentiate between simple movement versus dragging. For example, a mouse is capable of producing both move and drag events (thus the context is set depending on whether a button is depressed or not) but a touch-based input device lacks natural vocabulary to express non-dragging movement (when a finger makes contact with the touch surface and moves, a user expects "dragging", thus the `MoveEvent`'s context is always set to down).

3.2 Interpreters

Interpreters are the second class of components within HIPerFace, performing the first phase of the multi-stage observer pattern. They determine whether the HIPerFace events they receive should be combined into composite HIPerFace events (CHEs). Currently HIPerFace provides two CHEs:

- `ScaleEvent` – scales the "selected" object up or down by a specified scale factor.
- `RotateEvent` – rotates the "selected" object by a specified angle around the x, y or z axes.

Figure 5 shows the class hierarchy for interpreters. An abstract `HIPerFaceInterpreters` class establishes the observer framework for all interpreters – maintaining a list of consumers to notify once a composite interaction has been built – and the following two concrete classes extend it:

- `BasicInterpreter` – this is a special interpreter that to some degree "short-circuits" the interpreter infrastructure by elevating all AHEs it receives to CHEs without any transformation. This is useful in several cases. The three AHEs a producer creates (`DownEvent`, `UpEvent` or `MoveEvent`) may, at times, be full-fidelity interactions that we would like to display on our output devices. Additionally, a device that a producer is connected to may also be capable of generating events that can be translated into CHEs within a producer directly, without requiring any combination of additional AHEs by interpreters.
- `CompositeInterpreter` – an abstract class to be inherited by interpreters that do need to adapt streams of AHEs, translating them to CHEs.

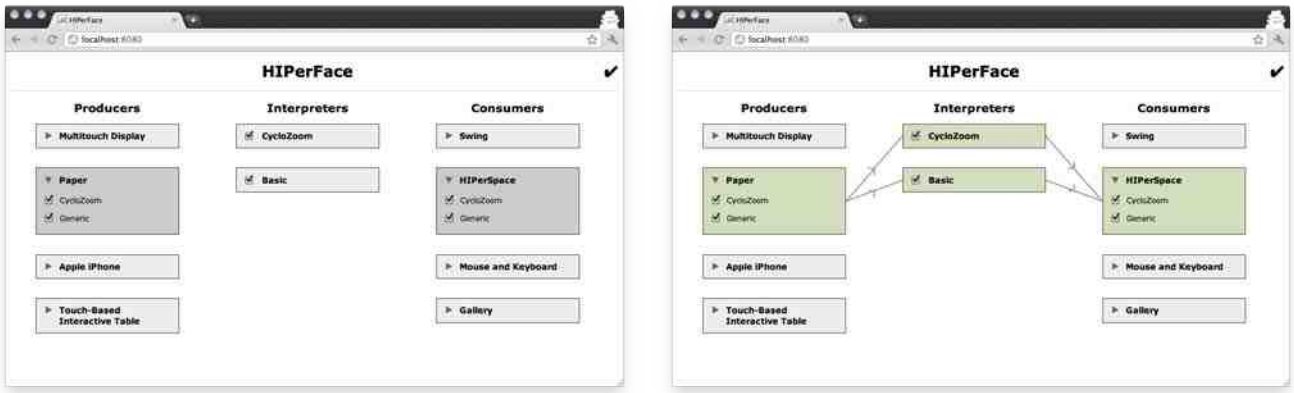
The `CycloZoomInterpreter` is one instance of a composite interpreter, exploiting the CycloStar approach [28]. This interpreter tracks `MoveEvent`s it receives to determine whether a circle is being traced by the input device. If it is, the interpreter discards the `MoveEvent`s and instead issues a `ScaleEvent` to observing consumers.

Interpreters have proven to be an extremely powerful approach. They aid extensibility by making it easy to integrate existing logic. More importantly they allow us to combine input received from multiple devices to produce interactions that are not possible with a single device. This has important consequences for supporting interaction with the HIPerSpace Wall since due to its scale it is inherently a multi-user and multi-device setting. In addition, even a single individual can benefit from use of multiple modalities such as using a digital pen with their dominant hand while interacting with a multitouch surface or other device with their non-dominant hand. We can easily integrate Pen+Touch interactions [20] by extending the `CompositeInterpreter` or one of the implemented scaling interpreters.

The combination of the "short-circuiting" `BasicInterpreter` and `CompositeInterpreter`s, however, introduced several challenges – most significantly with interactivity. We will discuss challenges and possible solutions in Sect. 4.

3.3 Consumers

Consumers are the second phase of our multi-stage observer pattern. Each consumer observes a set of interpreters and is notified when CHEs are created and need to be dispatched to output devices. As Fig. 7 shows, all consumers are subclasses of `HIPerFaceConsumer`, which establishes routines to adapt CHEs into NDEs.



(a) Initial state of the GUI

(b) Highlighting currently connected components

Figure 6: HIPerFace GUI: from left to right *Producers*, *Interpreters* and *Consumers*

Currently HIPerSpace provides the following consumers:

- `GalleryConsumer` – provides access to a specific application developed in Flash, enabling interaction with a set of photos mimicking the behavior of the HIPerSpace wall. It was used as a demonstration tool.
- `MouseKeyboardConsumer` – allows connected input devices to control the mouse cursor and keyboard on the machine currently running HIPerFace.
- `SpaceConsumer` – enables applications running on the HIPerSpace wall to be controlled by external events.
- `SwingConsumer` – uses Java Swing framework to emulate the HIPerSpace Wall. A Swing window is created and shapes are drawn to represent any cursors. It has proved to be a useful debugging tool.

As shown in Fig. 7, like producers, consumers also implement two interfaces enabling adaptation of the received events to NDEs and support communication with the output device or application. While consumers can exploit different ways of communicating with local or remote devices and applications, we identify two patterns of interaction with the output windowing system. Specific applications do not present a clear interface with which an HIPerFace consumer can effectively communicate, but being based on graphical user interface interactions, they usually support X-window events such as the ones generated by a mouse or keyboard. In this case the HIPerFace consumer exploits X-window events directly and conveys the target interaction by emulating mouse or keyboard events. This is, for example, used by the `GalleryConsumer` that sends mouse events to the external Flash application to perform object dragging. However, if an application or a device provides ready access to an internal interaction model – through a dedicated API – then the HIPerFace consumers can exploit this channel and directly send NDEs to the final application. This is how the `HIPerFaceConsumer` works in combination with the HIPerSpace wall applications.

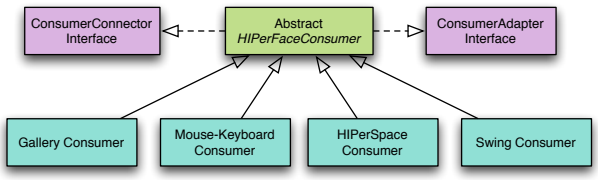


Figure 7: HIPerFace consumer class hierarchy

3.4 Configuring HIPerFace

The final piece of HIPerFace is a Web-based user interface that can be used to register interpreters with devices and consumers with interpreters. When HIPerFace is run, it starts a servlet (powered by the Jetty HTTP server[6]) that builds a list of all available components – producers, interpreters and consumers – by scanning their package paths, and building an associative array of ID, name, and type for every retrieved component. The servlet then produces an HTML 5 page in which the above data is represented as a JSON[7] object. Through the use of Javascript, the page lays out the available components across three columns. Initially none of the producers or consumers will have been started nor will any of the components be linked together.

To begin, the user must first enable a set of interpreters by checking the boxes beside the selected interpreters' names. Any of the producers or consumer boxes can be expanded to reveal interpreters with which they can be paired (see Fig. 6(a)). An AJAX call is made to register selections – thus starting a producer or consumer and associating it with the appropriate interpreter. If a checkbox is unchecked, the Web interface fires another AJAX call that stops the producer or consumer and detaches it from the corresponding interpreter. On every configuration change, HIPerFace updates the configuration file to ensure that the setup persists across running instances.

The primary reason we chose HTML 5 to build the front-end to HIPerFace was to employ the new `canvas` element and allow dynamic relationships between components to be shown. When a user hovers over a particular component, its box changes color and a number of arrows appear to highlight the flow of HIPerFace events through the system (see Fig. 6(b)).

4. INTERPRETING INTERACTIONS

We have described in the previous section how employing a multi-stage observer design pattern that combines producers, interpreters, and consumers in a layered architecture enables translating native events and building composite events. The implemented HIPerFace infrastructure allows us to easily create producer and consumer wrappers

[6]http://jetty.codehaus.org
[7]http://www.json.org

for both input and output devices and associate basic interpreters to process events and even permits short-circuiting for specific interactions (such as scaling). However, while implementing more complex interpreters, such as the `CycloZoomInterpreter`, we realized that the composite layer and the operations required to process a composite events could negatively impact interactivity as well as how users perceive the interaction. If the interpreter combines a sequence of atomic events without immediately signaling component events, this allows consumers to fire based only on the component events, and interactivity can be impacted. Atomic events have to be processed in a way such that users can experience direct and continuous feedback. Once a composite interaction is eventually recognized, the interpreter can then generate a composite event and its resulting consequences.

The problem is that in the process of recognizing a composite interaction, atomic events out of which it is composed might have changed the status of the application currently running on the output device (consumer), the displayed GUI, or in fact any property of the application. Ideally the system should be able to "retract" the atomic events that have been unnecessarily transmitted to the consumer, but this is often difficult and sometimes impossible. Since application state *undos* are extremely complex, especially in a multi-user, multi-device environment, our aim is to avoid the need to undo the results of atomic events that have been transmitted during the recognition of a composite interaction. The overall goal is that atomic/composite event sequences should not create unintended or invisible application states that might confuse the user. If successful then no undo is needed, the user sees what is happening and correctly understands the evolving state of the application.

In order to support this we employed a set of *design rules* for creating and handling composite events.

1) Atomic events that might be used as part of composite interactions should never change the stored state of the application (e.g. modify files, delete objects, etc.).

2a) Any visible changes resulting from atomic events that might be used in composite events should not confuse the user, or

2b) Composite events should be visually related in some way to the atomic events composing them (e.g., both should result in a similar pointer movement on the display).

This set of rules results in only visible application side-effects and ones that are unambiguous for users, avoiding invisible interaction caused by atomic events that cannot be undone.

5. CASE STUDY: HIPERSPACE WALL

The main goal of developing the HIPerFace architecture has been to enable exploration and rapid prototyping of multi-channel multimodal interactions based on composite events. In this section we detail the interface components that have been integrated into HIPerFace and the rationale behind the selections. We also describe how device interfaces have been wrapped in producers and consumers and we finally outline how our system supports extensibility by describing a series of use cases.

Observations of user interaction with the HIPerSpace wall reveal three key challenges: (1) interacting with the wall almost always involves multiple people working in coordination, (2) while applications running on the wall are often controlled through custom commands issued using a tethered mouse and keyboard or a gyroscopic mouse typically controlled by one experienced individual, there is a great potential to make interactions available to all participants, and (3) the sheer size and physical layout of the HIPerSpace wall challenge normal patterns of interaction. Although the workspace surrounding the wall was purposefully designed to be reconfigurable, with easy-to-move chairs and writing tables, it remains unclear how to best support groups and enable flexible interaction.

5.1 Alternative Input Devices

It is difficult to know in advance which device will perform best for a specific task, application, or situation. Our approach has been to build the infrastructure necessary to explore the introduction of a variety of tangible interactive devices and allow investigation of which specific devices or combinations of them best suit particular applications and settings. In the remainder of this section we will present an overview of the HIPerFace information flow with different input devices and how they have been integrated for interacting with the wall display.

Information Architecture

The HIPerFace architecture enables flexible definition of producers and consumers that exploit native input and output of a range of tangible devices. Figure 9 outlines the main information flow from the input devices through the HIPerFace infrastructure to the final output devices.

While the HIPerFace framework has been designed to be extensible and support multiple input and output devices, our initial research focus is on integration of three types of input devices – touch enabled smartphones, digital pens and paper, and multitouch surfaces – and one output device – the HIPerSpace wall. Figure 9 shows the integration of the respective HIPerDevices – *HIPerPhone*, *HIPerPaper* and *HIPerTouch* – and the SDKs that have been used to communicate with them.

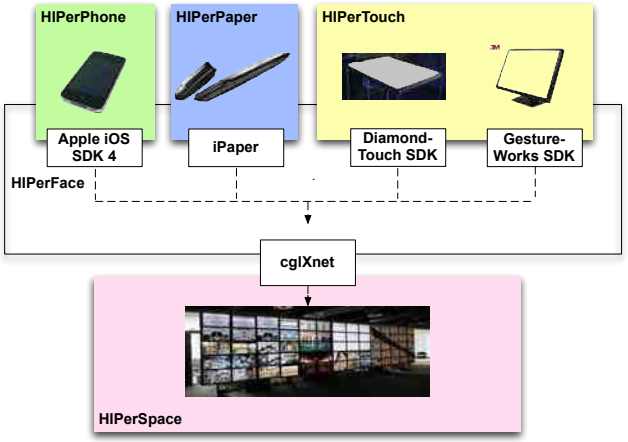


Figure 9: HIPerFace information flow

Figure 8: Tour guide (left, white shirt) narrates a wall demonstration. Her assistant (right, at the computer terminal) operates the HIPerSpace display with a keyboard and mouse while looking over his shoulder.

Multitouch surfaces

Touch-based interaction promises to be particularly well-suited for interacting with multiple objects in parallel [9]. One promising device is the 3M M2256PW multi-touch display[8]. This 22" display allows users to interact with information displayed on the screen with up to 20 touch points. Instead of using the multi-touch display to interact with information shown locally on the screen itself, we use the system as an input device only and, by coupling it with the HIPerFace infrastructure, we exploit its multiple touch points to interact with information displayed on the HIPerSpace wall. In order to do that, we integrated a special Flash-based bridge within the *HIPerTouch* component, enabling the transmission of touch-points to the HIPerFace system. The flash bridge exploits the GestureWorks SDK[9] to record touch-based interactions and forward the related native events to the HIPerFace system. In order to integrate the touch-based interface as part of the HIPerFace architecture, we only had to define how the HIPerTouch *producer* wraps native touch-based events in AHEs.

Although multi-touch displays such as the 3M M2256PW support multiple touch-points, they can not distinguish multiple users interacting simultaneously with the display. Since large wall displays like the HIPerSpace wall display are inherently multi-person, we decided to investigate a device that supports multi-person touch-based interactions. The DiamondTouch system[10] distinguishes between the touches of multiple people. To integrate the DiamondTouch table as a HIPerTouch component, we followed a similar approach to incorporating the 3M display; again only the touch-sensitive surface is used (nothing is projected on it) and touch-inputs are transferred to the HIPerFace system by wrapping them within HIPerTouch into AHEs. We exploit the DiamondTouch SDK to track touch-based information and allow native events to be transmitted to the HIPerFace system.

Mobile touch-based interfaces

Multitouch surfaces are typically not easily transportable nor movable. However, interacting with a large scale visualization on the HIPerSpace wall requires mobility and flexibility. In order to explore the use of mobile devices we integrated Apple iPhone and iPod-touch devices. Another advantage of these devices is that they exploit users' existing practices. These devices communicate over the network (WiFi or 3G) to the HIPerFace infrastructure and can generates specific interactions such as *scale* or (x,y) movements like *pan* or *drag*. We implemented a special iPhone/iPod-touch (and potentially also iPad) application on top of the Apple iOS SDK 4[11], as part of the *HIPerPhone* component. This application tracks finger touches, interprets gestures, and send them to HIPerFace through the `ServerConnectorInterface` outlined in the previous section. The HIPerPhone producer defines how to wrap events generated by the touch-based mobile devices and make them available as AHEs to the other components of HIPerFace.

Digital pen and paper

The interaction paradigm with the input devices presented until now intentionally neglects their display capabilities in order to encourage users to attend to the high-resolution wall rather than the low-resolution device. However, this does not permit the deployment of any interactive capabilities that go beyond the (in)direct manipulation of the displayed objects. While this could easily be added to touch displays and mobile devices, we have not done this and still think it has strong potential to distract the user's attention.

We decided to explore a novel digital pen and paper interface based on Anoto technology. Here we do make use of a display of information printed on the paper. *HIPerPaper* is a paper-based interface that allows printed documents to be used for interaction with the wall display [41]. As shown in Fig. 10, a printed representation of the wall enables users to navigate the display space, select and drag objects on it by positioning the pen in the input area of a sheet of paper. By moving the pen with a non-marking tip users can control the position of a cursor and, through different pressure levels applied with the pen on paper, select and drag objects. The flexibility supported by interactive paper in terms of *printed* interfaces, introduces further interaction capabilities in the form of detachable paper-based widgets that can be configured in different ways depending on the specific user needs. Printed widgets can be combined with the main interaction interface (as illustrated by the scaling widget in Fig. 10, which enables scaling of the selected object through circular gestures [41]) or can be deployed on removable stickers and attached to any physical object in the environment, such as the existing writing tablets.

[8]http://www.3m.com
[9]http://www.gestureworks.com
[10]http://www.circletwelve.com
[11]http://developer.apple.com/devcenter/ios

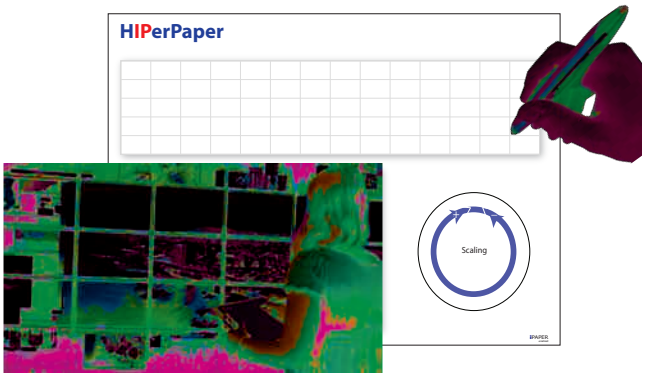


Figure 10: The HIPerPaper interface: paper-wall (top) and scaling widget (bottom-right)

To integrate digital paper interactions with HIPerFace, we exploited Anoto DP-201 digital pens and the iPaper framework [30]. Pen-based interactions are first collected by the framework, associated to the specific interactive area (paper-based navigation area or interactive widgets) and finally interpreted by specific Active Components [36] that generate native events to be transmitted to the HIPerFace infrastructure. The HIPerPaper consumer is then responsible for wrapping the generated events into AHEs and informing interpreters about them. For example, we implemented specific interpreters that react to HIPerPaper events and transform fired *move* events into *scale* events if a circular motion is detected by the specific `CycloZoomInterpreter`.

Additional tangible devices

While we feel that the three types of tangible devices we have already implemented enable an interesting variety of multi-device composite interactions, other instances of devices can be easily integrated and coupled with HIPerFace. The flexible architecture of HIPerFace enables other tangible device to be integrated with and communicate to the core system. A simple definition of a HIPerFace producer for the new tangible device, that knows how to communicate with it and how to wrap any received event into an AHE, is all that is required. We envision extending the range of supported devices, for example by integrating tablet interfaces such as Wacom[12] or any Tablet-PCs, Microsoft Surface systems, Nintendo Wii controllers[13], systems for tracking the physical movements of a person such as body or hands [10, 24, 4], and infrastructure for feet-based interaction with an interactive floor [1].

5.2 Communicating with the Wall Display

The HIPerFace infrastructure also takes care of handling the output channels. As we have seen earlier, NDEs generated by the different input devices are combined into CHEs which are translated into output NDEs and dispatched to output devices. In the same way as HIPerFace supports multiple input devices, it also supports multiple output devices. The infrastructure is general but since the main focus of our research is to study multimodal composite interactions with large wall displays, we have primarily focused on the HIPerSpace wall as the only output channel, exploiting different CGLX applications such as TiffViewer[14], GigaStack [31] as well as a newly defined CGLX test application. However, similarly as the simple definition of producers for the input devices, adding a new output device requires only implementing the corresponding HIPerFace *consumer* to communicate with the output device by transforming HIPerFace events into output NDEs.

In order to communicate with the HIPerSpace wall we defined a dedicated *HIPerSpace consumer*. It exploits CGLX, a flexible framework that enables access to the HIPerSpace wall and supports the transmission of events to CGLX applications running on the wall. CGLX provides flexible ways to interact with applications, depending on the type of device used. The standard interface is based on X-events which are usually generated by mouse and keyboard on a desktop machine. Recently, the CGLX framework has been extended with new services supporting the handling of multitouch events, 3D mouse[15] events, and custom-defined events. Even though CGLX supports multiple devices, bound to multiple users, the current infrastructure is expecting specific CGLX events that will be then interpreted by specific applications.

Once NDEs have been generated, we exploit the *cglXnet* library as part of the *HIPerSpace consumer* to connect to a remote CGLX controller and eventually send the generated events to the applications running on the wall display. The HIPerSpace consumer wraps the received HIPerFace events into the different kinds of events supported by CGLX and exploits a special *application server* to communicate with the wall display. The application server currently accepts X-events, multi-touch events and, most importantly, custom events. This way we are able to generate complex HIPerSpace events from composite events and send them through the CGLX framework, encoded as custom events, to the final applications. While at the current stage, single applications have to support and understand received events, our long-term goal is to define a set of more complex interactions (such as scale, rotate, pan, translate, etc.) that every application can natively exploit.

While the integration with the HIPerSpace wall was our primary objective, achieving the integration meant defining a dedicated consumer that was able to communicate over the network and comply with an existing framework. This necessitated coordinating many different components. In order to better understand the flexibility of our approach, we illustrate it with another example: the `GalleryConsumer` that was presented earlier. It was integrated in the system to mimic the functionality of the wall for demonstration purposes using a portable device such as a laptop and exploiting an external Flash-based picture viewer application to allow manipulation of displayed pictures. The `GalleryConsumer` communicates directly with the external flash application that reacts to drag events (dragging pictures on the display), navigate events (enabling exploration of photos on the display through a mouse cursor, without selecting them) and scale events (scaling selected photos up or down). In order to integrate the Flash application with HIPerFace we only

[12]http://www.wacom.com
[13]http://www.nintendo.com/wii/console/controllers
[14]http://vis.ucsd.edu/mediawiki/index.php/Research_Projects
[15]http://www.3dconnexion.com

needed to extend the abstract `HIPerFaceConsumer` class with a dedicated `GalleryConsumer` able to map the move events to either navigate or drag events, depending on the state of the HIPerFaceEvent (down or up), or to scale events sent directly to the corresponding scaling function in Flash.

5.3 Composing events

In this section we have seen how input and output devices can be added to the HIPerFace infrastructure by defining dedicated HIPerFace producers and consumers. We exploit this flexibility also at the level of the composite events by introducing dedicated *Interpreters* to support the composition of events generated by a range of input devices. In order to provide interactions that combine different devices or modalities, we just have to create new interpreters and bind them to the corresponding input and output devices, represented by HIPerFace producers and consumers. Besides complying with the layered architecture and the hybrid adapter/observer infrastructure, interpreters can define any logic necessary for AHEs to be combined and composed. As outlined in the previous section, it is crucial that interactivity design rules are respected.

An example of an interpreter exploiting multiple devices and combining different kind of tangible interactions is the envisioned `PenAndTouchInterpreter` we outlined earlier. Similar to the `CycloZoomInterpreter`, we can exploit characteristics of the interactions to define different states or modes. For example, by having the non-dominant hand control a touch-based menu in combination with a circular gesture with a pen in the dominant hand, we can support multiple circular interaction modes, such as scaling, rotating in 2D, rotating in 3D, etc. Integrating this interpreter, would only require: (1) implementing the mode switching functionality as part of the interpreter logic to respond to the different touch-based commands with different modes, (2) binding the new interpreter to the HIPerPaper and HIPerTouch producers and (3) linking the interpreter to HIPerSpace or Gallery consumers.

6. CONCLUSIONS

In this paper we introduced HIPerFace, a multi-channel architecture to enable exploration of multi-device multimodal interaction with a large display wall using tangible interface components such as multitouch surfaces and digital pens and paper. We described how we applied software engineering practices and design patterns to implement a novel three-tiered multi-channel architecture supporting multimodal interactions and enabling the generation of composite interaction events by combining native events from multiple tangible devices. We documented how we exploited a multi-stage hybrid adapter/observer design pattern to support effective communication in HIPerFace's three-layered architecture and how interactivity problems were avoided by introducing design rules to ensure that atomic/composite pipelined events streams present consistent visual feedback and are handled consistently throughout the system.

We provided case studies of how we employed HIPerFace to connect applications running on the ultra-scale HIPerSpace wall with multitouch displays, multi-user touch-based interactive tables, touch-based mobile phones, and digital pens and paper. The goal of the case studies was to describe how HIPerFace enables exploration and rapid prototyping of multi-channel multimodal interactions based on composite events. We also detailed the layered architecture of HIPerFace and described its generality by documenting how new interface components are integrated.

While the HIPerFace architecture enables flexible composition of events, it is not yet clear how the system scales in a situation where a large number of users is concurrently utilizing a range of different devices. These issues have been investigated in the past, for instance in the Event Heap system [23], and are one of the topics of our future work.

7. REFERENCES

[1] T. Augsten, K. Kaefer, R. Meusel, et al. Multitoe: High-Precision Interaction with Back-Projected Floors Based on High-Resolution Multi-Touch Input. In *Proc. UIST 2010*, pages 209–218, 2010.

[2] R. Ballagas, M. Ringel, M. Stone, and J. Borchers. istuff: a physical user interface toolkit for ubiquitous computing environments. In *Proc. CHI '03*, pages 537–544, 2003.

[3] R. Ballagas, M. Rohs, and J. G. Sheridan. Sweep and point and shoot: phonecam-based interactions for large public displays. In *Proc. CHI '05*, pages 1200–1203, 2005.

[4] H. Benko and A. Wilson. DepthTouch: Using depth-sensing camera to enable freehand interactions on and above the interactive surface. In *Proc. IEEE Workshop on Tabletops and Interactive Surfaces*, 2008.

[5] M. Bernstein, A. Robinson-Mosher, R. B. Yeh, and S. R. Klemmer. Diamond's Edge: From Notebook to Table and Back Again. In *Extended Abstracts (Posters) of Ubicomp '06*, 2006.

[6] X. Bi and R. Balakrishnan. Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. In *Proc. CHI '09*, pages 1005–1014, 2009.

[7] F. Block, M. Haller, H. Gellersen, C. Gutwin, and M. Billinghurst. VoodooSketch: Extending Interactive Surfaces with Adaptable Interface Palettes. In *Proc. TEI '08*, pages 55–58, 2008.

[8] P. Brandl, M. Haller, J. Oberngruber, and C. Schafleitner. Bridging the Gap between Real Printouts and Digital Whiteboard. In *Proc. AVI '08*, pages 31–38, 2008.

[9] W. Buxton, R. Hill, and P. Rowley. Issues and techniques in touch-sensitive tablet input. *SIGGRAPH Comput. Graph.*, 19(3):215–224, 1985.

[10] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *Proc. SIGGRAPH '93*, pages 135–142, 1993.

[11] P. H. Dietz and D. L. Leigh. DiamondTouch: A Multi-User Touch Technology. In *Proc. UIST '01*, pages 219–226, 2001.

[12] K. Doerr and F. Kuester. CGLX: A Scalable, High-performance Visualization Framework for Networked Display Environments. *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[13] B. Dumas, D. Lalanne, and R. Ingold. Hephaistk: a toolkit for rapid prototyping of multimodal interfaces.

In *Proc. ICMI-MLMI '09*, pages 231–232, 2009.

[14] C. Forlines, D. Vogel, and R. Balakrishnan. Hybridpointing: fluid switching between absolute and relative pointing with a direct input device. In *Proc. UIST '06*, pages 211–220, 2006.

[15] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley, March 1995.

[16] K. Grønbæk, J. Kristensen, P. Ørbæk, and M. A. Eriksen. Physical Hypermedia: Organizing Collections of Mixed Physical and Digital Material. In *Proc. Hypertext '03*, pages 10–19, 2003.

[17] F. Guimbretière. Paper Augmented Digital Documents. In *Proc. UIST '03*, pages 51–60, 2003.

[18] M. Haller, d. Leithinger, J. Leitner, T. Seifried, P. Brandl, J. Zauner, and M. Billinghurst. The Shared Design Space. In *Proc. SIGGRAPH '06*, 2006.

[19] J. Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proc. UIST '05*, pages 115–118, 2005.

[20] K. Hinckley, K. Yatani, M. Pahud, N. Coddington, J. Rodenhouse, A. Wilson, H. Benko, and B. Buxton. Pen + touch = new tools. In *Proc. UIST '10*, pages 27–36, 2010.

[21] S. Hodges, S. Izadi, A. Butler, A. Rrustemi, and B. Buxton. ThinSight: Versatile Multi-Touch Sensing for Thin Form-factor Displays. In *Proc. UIST '07*, pages 259–268, 2007.

[22] H. Ishii and B. Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *Proc. CHI '97*, pages 234–241, 1997.

[23] B. Johanson and A. Fox. The event heap: A coordination infrastructure for interactive workspaces. In *Proc. WMCSA '02*, page 83, 2002.

[24] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1:67–74, 2002.

[25] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reacTable: Exploring the Synergy between Live Music Performance and Tabletop Tangible Interfaces. In *Proc. TEI '07*, pages 139–146, 2007.

[26] S. R. Klemmer, J. Li, J. Lin, and J. A. Landay. Papier-mâché: toolkit support for tangible input. In *Proc. CHI '04*, pages 399–406, 2004.

[27] C. Liao, F. Guimbretière, K. Hinckley, and J. Hollan. PapierCraft: A Gesture-Based Command System for Interactive Paper. *ACM Transactions on Computer Human Interaction*, 14(4):1–27, January 2008.

[28] S. Malacria, E. Lecolinet, and Y. Guiard. Clutch-free panning and integrated pan-zoom control on touch-sensitive surfaces: the cyclostar approach. In *Proc. CHI '10*, pages 2615–2624, 2010.

[29] N. Marquardt and S. Greenberg. Distributed Physical Interfaces with Shared Phidgets. In *Proc. TEI '07*, pages 13–20, 2007.

[30] M. C. Norrie, B. Signer, and N. Weibel. General Framework for the Rapid Development of Interactive Paper Applications. In *Proc. CoPADD '06*, pages 9–12, 2006.

[31] K. Ponto, K. Doerr, and F. Kuester. Giga-stack: A method for visualizing giga-pixel layered imagery on massively tiled displays. *Future Gener. Comput. Syst.*, 26(5):693–700, 2010.

[32] J. Rekimoto and M. Saitoh. Augmented Surfaces: a Spatially Continuous Work Space for Hybrid Computing Environments. In *Proc. CHI'99*, pages 378–385, 1999.

[33] J. Sanneblad and L. E. Holmquist. Ubiquitous graphics: combining hand-held and wall-size displays to interact with large images. In *Proc. AVI '06*, pages 373–377, 2006.

[34] B. Signer, U. Kurmann, and M. C. Norrie. iGesture: A General Gesture Recognition Framework. In *Proc. ICDAR '07*, pages 954–958, 2007.

[35] B. Signer and M. C. Norrie. A Framework for Developing Pervasive Cross-Media Applications based on Physical Hypermedia and Active Components. In *Proc. ICPCA '08*, pages 564–569, 2008.

[36] B. Signer and M. C. Norrie. Active Components as a Method for Coupling Data and Services - A Database-Driven Application Development Process. In *Proc. ICOODB '09*, 2009.

[37] E. Tse, M. Hancock, and S. Greenberg. Speech-filtered bubble ray: improving target acquisition on display walls. In *Proc. ICMI '07*, pages 307–314, 2007.

[38] B. Ullmer and H. Ishii. mediaBlocks: Tangible Interfaces for Online Media. In *Extended Abstracts of CHI '99*, pages 31–32, 1999.

[39] N. Villar and H. Gellersen. A Malleable Control Structure for Softwired User Interfaces. In *Proc. TEI '07*, pages 49–56, 2007.

[40] G. Wallace, O. J. Anshus, P. Bi, et al. Tools and applications for large-scale display walls. *IEEE Comput. Graph. Appl.*, 25(4):24–33, 2005.

[41] N. Weibel, A. M. Piper, and J. D. Hollan. HIPerPaper: Introducng Pen and Paper Interfaces for Ultra-Scale Wall Displays. In *Proc. UIST '10*, pages 407–408, 2010.

[42] M. Weiss, J. Wagner, Y. Jansen, R. Jennings, R. Khoshabeh, J. D. Hollan, and J. Borchers. Slap widgets: bridging the gap between virtual and physical controls on tabletops. In *Proc. CHI '09*, pages 3229–3234, 2009.

[43] A. D. Wilson and H. Benko. Combining Multiple Depth Cameras and Projectors for Interactions On, Above, and Between Surfaces. In *Proc. UIST '10*, pages 273–282, 2010.

[44] R. B. Yeh, S. R. Klemmer, A. Paepcke, M. Bastéa-Forte, J. Brandt, and J. Boli. Iterative Design of a Paper + Digital Toolkit: Supporting Designing, Developing, and Debugging. Technical Report CSTR 2007-10, Stanford University, Computer Science Department, March 2007.