XDKinect: Development Framework for Cross-Device Interaction using Kinect

Michael Nebeling, Elena Teunissen, Maria Husmann and Moira C. Norrie
Department of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
{nebeling,husmann,norrie}@inf.ethz.ch, telena@student.ethz.ch

ABSTRACT

Interactive systems set in multi-device environments continue to attract increasing attention, prompting researchers to experiment with emerging technologies. This paper presents XDKinect—a lightweight framework that facilitates development of cross-device applications using Kinect to mediate user interactions. The main benefits of XDKinect include its simplicity, adaptability and extensibility based on a flexible client-server architecture. Our framework features a time-based API to handle full-body interactions, a multi-modal API to capture gesture and speech commands, an API to utilise proxemic awareness information, a cross-device communication API, and a settings API to optimise for particular application requirements. A study with developers was conducted to investigate the potential of these features in terms of ease of use, effectiveness and possible use in the future. We show several example applications of XDKinect, as well as discussing advantages and limitations of our framework as revealed by our user study and experiments.

Author Keywords

Kinect; development framework; cross-device applications.

ACM Classification Keywords

H.5.2 User Interfaces: Input devices and strategies

INTRODUCTION

The flexibility and complexity imposed by multi-device environments often requires experimentation with alternative, sometimes radically different, user interface designs. Our goal is to exploit the potential of Microsoft Kinect[1] for experimentation with different forms of multi-modal interaction involving multiple devices and users. Kinect's RGB camera, 3D depth sensors and high-quality audio capture are valuable for augmenting target applications with support for multi-modal input. Its relatively cheap price and increased popularity make Kinect attractive to be explored for multi-device application development, in particular, in settings that previously required complex hardware and software setups [1, 16].

This paper presents XDKinect, a user interface development toolkit that uses Kinect to mediate interactions between multiple devices and users. XDKinect is characterised by three main aspects: (1) lightweight support for cross-platform, multi-device development based on native web technologies with which many developers are already familiar, (2) a flexible client-server architecture enabling a variety of multi-device ecosystems around Kinect, (3) useful programming abstractions from low-level details of Kinect's standard SDK.

First, our goal is to provide a lightweight toolkit for easy and rapid development of multi-device applications using Kinect as an intermediator. Currently, a wide range of skill sets and experience with many different platforms, languages and technologies are required for implementations to achieve compatibility with many types of devices. We promote a web-based approach that seems promising to reach across device boundaries as it, not only enables applications to run on the wide range of web-enabled devices available today, but also allows developers to leverage their knowledge of web standards. This is in contrast to approaches that promote new models and languages to support multi-device development [13]. These approaches often tie in with certain programming paradigms (e.g. model-based [8], object-oriented [4], data-oriented [2]), impose multiple abstraction levels, or require proprietary languages for programming and specification, all of which seems to work against widespread adoption.

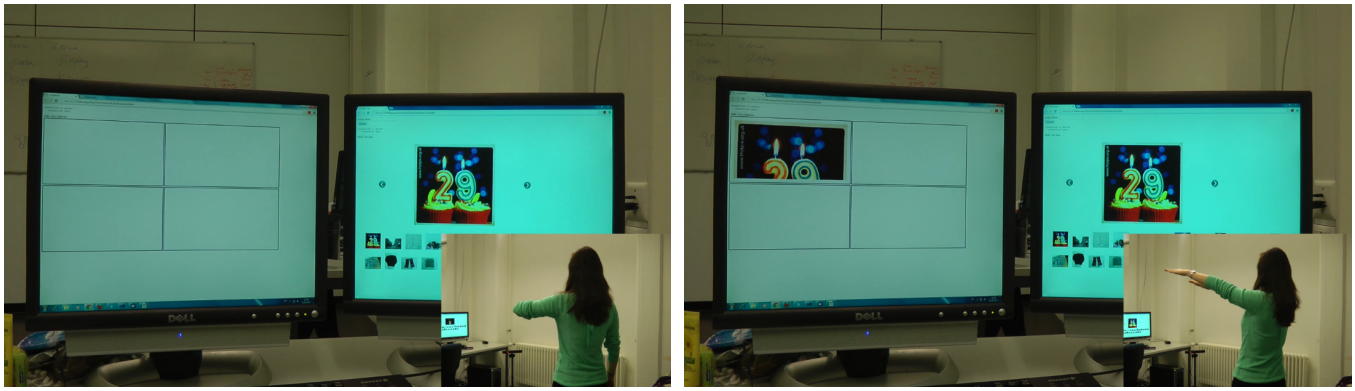
The second important aspect of XDKinect is its flexible client-server architecture. The core logic of XDKinect applications can be shared and distributed between one or multiple clients and a server used for hosting Kinect. This special role of the server enables many different multi-device scenarios around Kinect. Supported operating systems and platforms include Android, iOS and Windows. Given that the Kinect SDK is only available for Windows, the XDKinect server currently requires Windows. However, XDKinect clients can be any web-enabled device with support for modern web browsers. This includes the whole range from mobile devices such as smartphones and tablets (e.g. Android phones, Apple iDevices), over laptops and desktops, to tabletops (e.g. Microsoft PixelSense) and interactive walls.

Third, XDKinect affords useful abstractions from Kinect's standard SDK. Specifically, it provides a settings API to op-

[1] http://www.kinectforwindows.org

EICS 2014, June 17–20, 2014, Rome, Italy.

http://dx.doi.org/10.1145/2607023.2607024



(a) Browsing a collection of images using hand gestures (b) Selecting an image and moving it to another device

Figure 1. Scrapbook application using XDKinect for whole-body gesture-based cross-device interaction

timise for particular application requirements, a time-based API to query and constrain Kinect streaming data, a multimodal API for gesture and speech recognition, and APIs for inter-client communication to implement cross-device sessions involving multiple, distributed clients [11] as well as for proxemic interaction [1].

To demonstrate the potential of XDKinect, we present two applications created using our framework. A first example application is Scrapbook shown in Figure 1. Here, a user can browse through a collection of images on one device using hand gestures and pull selected ones over to another device for additional tasks. In a second application, we show how XDKinect can also take cues from the user's position and adapt the displayed content to fit either into personal or ambient interaction modality [16]. Moreover, collaborative interaction is feasible since input from more than one user and device can be distinguished using XDKinect.

This paper is organised as follows. We begin in the next section with a discussion of related work. The key features of XDKinect and its architecture are then presented together with our implementation. This is followed by a description of the two sample applications developed using XDKinect. We then present our study conducted with 12 developers to assess its ease of use and effectiveness for building cross-device applications. Finally, we compare XDKinect with other solutions and discuss limitations as well as future work.

BACKGROUND

Our work relates to three streams of research: proxemic and ambient interaction, multi-device user interfaces and in-browser Kinect applications.

Proxemic and Ambient Interaction

Research in proxemic interaction investigates spatial relationships between users and objects, taking cues from distance, orientation, movement and identity to study interaction behaviour of users and devices in their immediate environment. Ballendat et al. [1] extended Hall's proxemic theory by adding notions of fine-grained sensing of nearby people and devices, mediating between implicit and explicit interaction, and distinguishing between four discrete proxemic zones. These principles were incorporated into a framework, which they illustrated using the scenario of a Proxemic Media Player. Later, they developed the Proximity Toolkit [6] that gathers data from various tracking devices, making it available through an event-driven, object-oriented API. Relationships between entities can be observed and closely investigated using a visual monitoring tool. Finally, GroupTogether [7] augments proxemic principles with theories of F-formation and micro-mobility. The former investigates physical arrangements of a small group of people engaged in a focused conversation, while the latter considers the impact of re-orienting and re-positioning physical devices on information sharing techniques. Combining these two aspects, new techniques of cross-device interaction with emphasis on fluid and smooth communication can be designed. For instance, tilting a device by a small angle may trigger an information sharing process with other devices within proximity.

Similar ideas for proxemic interaction were also proposed in [16], conceptualising design principles for developing a shareable, public and private Interactive Ambient Display. The principles include visualising the data in a calm, aesthetically pleasing manner, naturally revealing meaning and functionality, supporting short-duration fluid interaction, and promoting shared use while combining public and personal information. Based on these principles, an interaction framework was developed with support for four continuous phases with fluid inter-phase transitions: ambient display, implicit interaction, subtle interaction and personal interaction.

XDKinect builds on these works, implementing the core ideas in a web-based toolkit with the goal of providing lightweight technical tools for rapidly developing and experimenting with new forms of cross-device interaction in multi-device environments. While previous works focused on the types of applications and interaction techniques that could be supported if there were such technologies, our main goal is to enable developers with varying programming experience to implement such applications given technologies such as Kinect.

Multi-Device User Interfaces

A logical framework to alleviate development of multi-device applications with distributed user interfaces was presented in [13]. Their work identifies several design dimensions to help

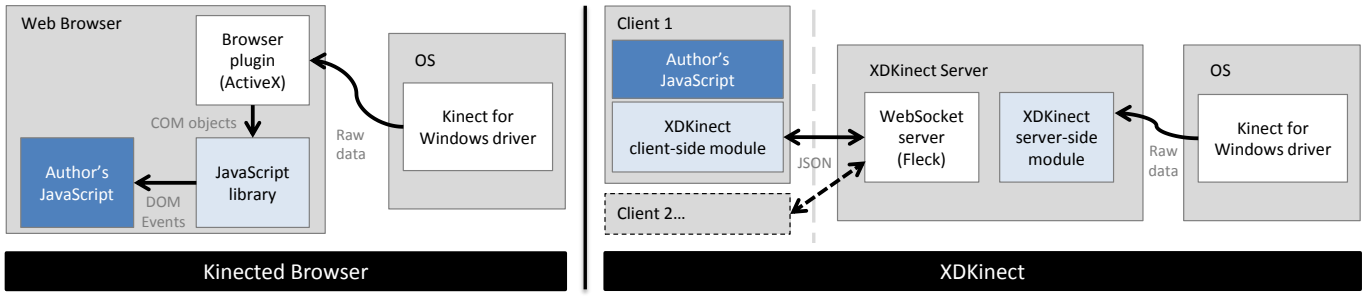


Figure 2. XDKinect's client-server architecture compared to the typical way of using a web browser plugin to access Kinect data directly

researchers and developers better analyse and evaluate current solutions. It provides an extensive overview of a wide range of various strategies in cross-platform development including abstract user interface representations, distribution and migration between multiple devices and users. As such, it offers a systematic classification of an assortment of design decisions and their shortcomings, serving as a valuable starting point into research about multi-device environments.

Many different approaches have been explored with object-oriented frameworks such as ZOIL [4], data-oriented frameworks such as Shared Substance [2] and model-based frameworks such as [8]. In a recent paper [11], we presented our ongoing work on a platform facilitating design and evaluation of cross-device applications both at the data model and user interface level. The paper introduces the notion of cross-device sessions that link the concepts of user, device and information. Developers can specify different scenarios of information sharing, where devices and sessions can be paired flexibly, allowing completely independent interactions, shared interactions, or arbitrary mixtures thereof. XDKinect implements a simplified version of this session concept for cross-device communication and keeping track of participating devices and users.

In-Browser Kinect Applications

Several attempts have been made to make Kinect available from within web browsers. Common to libraries such as Kinected Browser [5], KinectJS[2] and DepthJS[3] is that they extend the familiar JavaScript DOM event model with custom events to support processing of Kinect's skeletal tracking and raw data streams (audio, colour, depth and infrared) directly in the browser. While this is one of the initial challenges we also had to address in developing XDKinect, these solutions are limited to single-device scenarios and only certain browsers. Also, tight integration with the browser requires that the Kinect is connected to the same computer on which the client is running. These factors taken together considerably limit the types of applications that can be created and the settings in which they can be deployed and tested.

XDKINECT

The aim of XDKinect is to facilitate development of cross-device interfaces and interaction techniques that can cater for a wide range of applications and use scenarios. A crucial step in designing a general and useful framework was to experiment with different concepts and architectural designs to support the development process. To do so, we created several applications to assess Kinect's capabilities and recognise the most common requirements. We present some examples in the following sections.

XDKinect's architecture was developed in several iterations. Initially, the main idea was to provide a shorthand method to access Kinect data from within the browser. We also wanted to base the project on existing frameworks, KinectJS and DepthJS. Sadly, however, these are no longer maintained and incompatible with current browser versions. As we started to experiment with our own Kinect framework in several related projects, the number of architectural concerns grew with the need to support different requirements. For example, some applications such as a simple gestural controller for PowerPoint presentations only required Kinect's skeleton tracking, while another one with the goal of supporting web browsing using Kinect for gesture and speech input required additional Kinect data streams. In March 2013, Microsoft released a new version of the Kinect SDK[4], which was after we had started working on the XDKinect project. A novel set of features was provided with KinectInteraction, an extension that incorporates gesture-based tracking of a user's primary interactive hand allowing grip and press gestures to be detected. KinectInteractions rely on another tracking method that requires an additional Kinect data stream. Evolutions like this drove us to pay particular attention to the design of XDKinect's architecture so that any component could be adapted and extended to include new features with little effort.

Figure 2 shows XDKinect's final architecture and how it compares to previous solutions. Although based on a client-server model, XDKinect breaks from the traditional roles in that the server is not primarily used to host and manage the content, but to host and control the Kinect. Our architecture is different from existing solutions as it enables scenarios in which the Kinect is not directly connected to the client computer. This includes cross-device interaction involving multiple distributed clients, which is possible based on a single Kinect server. The cost of this indirection is kept minimal as XDKinect is highly configurable and implements several mechanisms to reduce client or server-side processing as required and manage with the bandwidth while controlling what kinds of data are processed and transferred.

[2] http://kinect.childnodes.com
[3] http://depthjs.media.mit.edu
[4] http://www.microsoft.com/en-us/kinectforwindows/develop/new.aspx

Feature	Examples	Description
settings	XDKinect.settings({ maxUsers: 2, joints: ['leftHand', 'rightHand'] });	Set tracked skeleton limit and selected joints to be tracked
skeleton	XDKinect.on('skeleton', function(skeleton) { console.log("distance: "+skeleton.joints.spine.z); });	Register callback for new skeleton frames
gesture	XDKinect.on('gesture', { 'leftGrip', 'rightGrip' }, function(skeleton, gesture) { console.log("gesture recognized: "+gesture); });	Register callback for gesture events of interactive hand
speech	XDKinect.on('speech', { 'Kinect on', 'Kinect off' }, function(term) { console.log(term); });	Register callback for speech commands
distance	XDKinect.on('distanceFar', function() { console.log("ambient zone"); });	Register callback for 4 or more metres distance from Kinect
users	XDKinect.on('userJoined', function(skeleton) { console.log("Welcome, user "+skeleton.trackingId); });	Register callback for new recognised user
message	XDKinect.message('ambient', 'hello', JSON);	Send hello message with JSON data object to ambient display
	XDKinect.on('hello', function(client, data) { console.log("received "+data+" from client "+client); });	Register callback for hello message from main display

Figure 3. XDKinect's features with code examples

The client and server-side components both employ an event-driven design. Based on this architecture, XDKinect supports two different ways of accessing Kinect data. First, similar to Kinected Browser [5], a low-level approach is supported by providing direct access to Kinect's data streams. For example, XDKinect provides an event to capture skeleton coordinates from the server (Figure 3). The event is fired whenever a new Kinect skeleton frame is available. To receive skeleton data, developers only need to subscribe to the event by declaring a callback function. At the same time, however, we also support high-level access similar to KinectJS and DepthJS, with which it is possible to fire custom events for more complex actions composed of multiple low-level events that may be generated on either the client or the server side.

In the following, we discuss the key concepts and how they are encapsulated in XDKinect's APIs (cf. Figure 3).

Time-based API

To support low-level access, XDKinect generalises the central concept of a touch history used in jQMultiTouch [10] to different interaction modalities in that XDKinect maintains internally a history for each Kinect data stream. For example, XDKinect collects joints coordinates for each tracked user from the skeleton stream. New Kinect skeleton events are generated at a rate of 30 frames per second. Every incoming Kinect skeleton object is pushed into a buffer, while the oldest entries are deleted. Here, the default storage duration amounts to 4.5 seconds (135 frames per tracked user). This is sufficient for many applications relying on skeletal tracking, but we will later describe how it can be configured both client and server-side. Possible applications include hand-as-cursor tracking, custom gesture detection, and various statistics to deduce user behaviour. A sample joints coordinates history object is shown in Listing 1.

```
[{trackingId: 148, timestamp: 1374225394869, rightHand:
    {x : 0, y: 0.1, z: 0.2}},
{trackingId: 148, timestamp: 1374225395678, rightHand: {x
    : 0.3, y: -0.15, z: 0.25}}]
```

Listing 1. Sample history object; only right hand is tracked

Similar to jQMultiTouch, developers are able to segment, query and constrain the history easily, e.g. to seek a certain frame, extract a portion of the history, filter by skeleton, look for selected joints, and evaluate it based on thresholds. Listing 2 shows a simple example for flick-hand gestures.

```
XDKinect.on("skeleton", function(skeleton) {
  var hand = skeleton.joints[hand], handRef =
      XDKinect.skeletonHistory({ skeleton: skeleton,
      time: '0..300', joints: ['leftHand', 'rightHand']
      }).last().joints[hand], deltaX = hand.x-handRef.x,
      deltaY = hand.y-handRef.y;
  if (Math.abs(deltaX) >= 0.4 && Math.abs(deltaY) <=
      0.075) {
    if (deltaX < 0) {
      console.log(hand+" flick-left");
    } else {
      console.log(hand+" flick-right")
    }
  } });
```

Listing 2. History usage for left/right-hand flick gestures

Additionally, the colour stream can be buffered either at 30 frames per second with 640x480 RGB colour bitmaps or at 12 frames with 1280x960 pixels. The depth and infrared streams are represented in specific image formats at 30 frames with 640x480 pixels, while audio is available in 16-bit PCM format, sampled at 16 kHz. Similar to the skeleton stream, these raw data streams can be segmented, constrained and, using additional libraries, further processed, e.g. looking for certain speech commands in the audio stream.

Multi-modal API

As mentioned above, XDKinect also offers a high-level API to register for, and react on, recognised gesture and speech commands from the server. XDKinect deduces high-level events from the skeleton, interaction and audio streams. While Kinect's skeletal tracking does not include hand tracking, KinectInteraction can detect hand gestures such as grip, grip release and press, and can discern between the right and the left hand of one or two users. The audio stream can be processed using speech recognition software. Kinect's default speech recognition service, Microsoft.Speech[5], was specifically optimised for the Kinect hardware. It can detect speech commands specified in W3C speech recognition grammar[6]. XDKinect application developers do not have to worry about the specifics and differences of these Kinect-internal APIs. Rather, clients only need to subscribe for the desired gesture and speech commands and will be notified automatically as these are recognised (cf. Figure 3).

Proxemic API

Based on Kinect's skeleton stream, another feature of XDKinect is distinguishing different proxemic parameters such as near and far distances between users and Kinect or between users themselves [1]. XDKinect allows developers to divide Kinect's field of view into multiple interaction zones and trigger events based on distance thresholds. Applications may use this information to support transition from one mode into another similar to [16]. XDKinect also supports a simple form of multiple user tracking. Whenever a user enters or leaves Kinect's field of view, XDKinect triggers a corresponding event. User identification is performed on the basis of the Kinect skeleton IDs (cf. Figure 3).

[5] http://msdn.microsoft.com/en-us/library/jj127857

[6] http://www.w3.org/TR/speech-grammar

Cross-Device Communication API

While the features discussed so far rely on Kinect, XDKinect adds an inter-client communication mechanism based on instant messaging to enable cross-device interaction in real-time. Information can be exchanged and shared between any clients registered in the system. Each new client only needs to connect to the server to be able to communicate with other clients. Clients can pass information along and share state through XDKinect's internal messaging service that was inspired by [14], but was adapted to the web environment by using web sockets and JSON for data exchange. We experimented with different JSON formats to find a good compromise between readability and traceability, which are important for debugging, and processing time required for encoding/decoding.

XDKinect is scalable to support a large number of clients. First, each client runs independently from the others and may only subscribe for information from the server. By default, interactions performed with one client have no impact on others. In order to be able to accept messages from other clients, a listener for this type of event is required (Figure 3). An important principle is the notion of active and passive clients. While active clients continuously receive Kinect data, passive clients do not. For example, while the main display is alert and awaits interaction, a secondary, passive display could be in sleeping mode since it first requires shared content from the main display. All clients can be explicitly set to active/-passive mode by the developer. This contributes to a boost in performance, decreasing the overhead by cutting information flow. Passive clients can be configured to automatically "wake up" upon a message receipt from another client.

Note that it is also possible to run one or more XDKinect clients and the XDKinect server locally so that it is sufficient to use a multi-monitor setup on a single device.

XDKinect was specifically designed for cross-device application development. While the cross-device support is therefore an essential part of the framework, it is implemented as a module following good principles of software design. As a result, developers may choose to extend or even replace this component without having to touch other parts of the framework. It is also the case that not all XDKinect applications must make use of the cross-device communication API and, as mentioned above, clients can work independently.

Settings API

One of XDKinect's key features is the facility to configure the server so that only data relevant for subscribed events is processed and transferred to clients. Furthermore, many parameters serve the goal of increasing the system's overall performance. For example, developers may switch Kinect's tracking (default/near), choose a skeleton selection strategy (track the closest/most active skeletons) and specify the maximum number of users to be tracked[7].

In order to receive information, clients must connect and first instruct the server by providing an individual configuration, such as active/passive status, joints to be tracked, gestures to be recognised, speech grammar, and any additional XDKinect events. Based on the information desired by clients, the server configures the Kinect and filters the event histories to only send back relevant data. While clients also have the possibility to change Kinect settings dynamically at runtime, some changes may cause a delay or may even require Kinect to be reset at run-time. For example, switching Kinect tracking mode and reloading grammar for speech recognition exhibits a seconds-long delay. XDKinect tries to avoid such interruptions as much as possible by aggregating individual client configurations on the server side.

IMPLEMENTATION

XDKinect is implemented in C# on the server and in JavaScript/jQuery[8] on the client side. The main purpose of the server application is to pre-process Kinect data and transfer it to each client as specified by that client's individual configuration. The server logic can be separated into four main components responsible for 1) configuring Kinect, 2) the handling of Kinect streams, 3) history keeping and filtering, and 4) inter-client messaging. The client-side module parses and dispatches messages from the server and other clients. It exposes the Kinect data to the developer through XDKinect's client-side APIs. Communication between the server and clients, and between clients, is based on the WebSocket protocol. Note that web sockets offer full duplex bidirectional communication with much less overhead than traditional HTTP-based methods. XDKinect relies on Fleck[9], an open-source web socket server in C# to transmit data to, and between, subscribed clients. All data, server-side notifications as well as inter-client messages, are exchanged via a common JSON format.

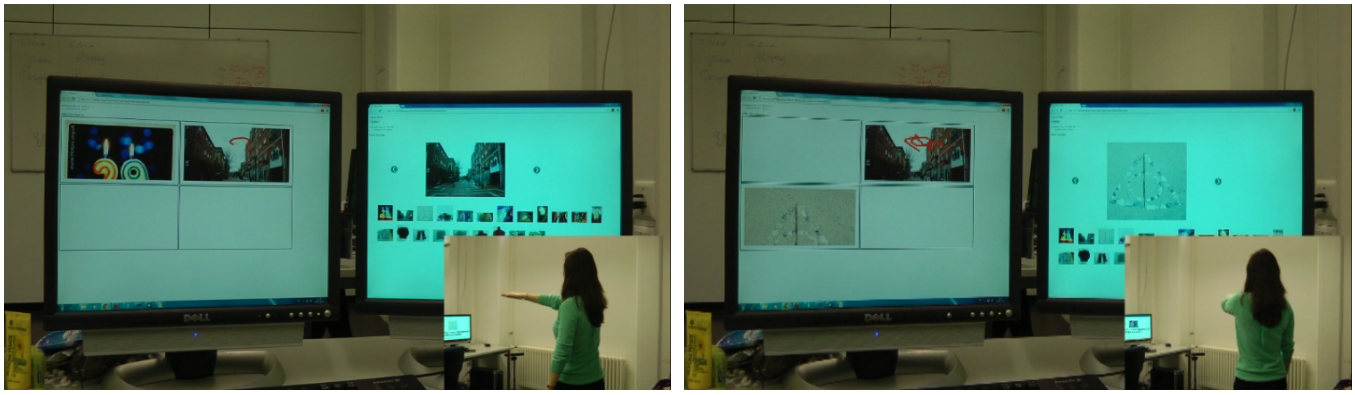
Kinect's skeleton stream is the primary source of information, as it offers not only joint coordinates, but distance metrics and multiple user tracking is also deduced from it. Essentially, the server maintains a list of tracked skeletons and checks it for users that joined or left by comparing the Kinect skeleton IDs each time a new frame arrives. If there are any changes, all subscribed clients are notified. Analogously, any client subscribed for distance change notification receives a message when a user's `z` skeleton coordinate is smaller or larger than specified thresholds. Accordingly, the interaction stream and audio stream are watched for gesture and speech commands. Whenever a command is recognised, the server compares the recognition results with the criteria set by any of the clients. Should a match occur, the corresponding client is notified. Cross-device communication is implemented using the same mechanism. The server listens for message events from clients. Once such an event is received, the server parses the message and forwards it to the target client.

XDKinect can be extended on both the client and the server side with little programming effort. For example, when the new KinectInteractions came out, we linked the library to the server, registered the interaction stream in XDKinect, and specified a new event handler. Likewise, the client side was

[7]Note that current Kinect hardware can recognise up to six users and fully track 20 joints of up to two skeletons.

[8]http://jquery.com

[9]https://github.com/statianzo/Fleck



(a) Annotating second image using hand-as-brush

(b) Removing first image using gesture or speech

Figure 4. Scrapbook application using XDKinect for whole-body gesture-based cross-device interaction (cont.)

extended with a new event listener. It is also easily possible to integrate client-side gesture libraries such as $1 recogniser [17], e.g. to support stroke-based gestures such as circle-hand. This requires a mapping from 3D space to 2D, e.g. by only considering `x` and `y` coordinates.

APPLICATIONS

In this section, we present two sample applications based on XDKinect. The first application, the Scrapbook mentioned in the introduction, was produced with an early version of XDKinect. The idea of Scrapbook is to browse through and collect images from a large photo collection, e.g. photos of places visited on a holiday, to put together an album. The second application, Fotobook, is similar in idea, but makes more advanced use of XDKinect's features as it reacts to multiple users and distinguishes different interaction zones.

Scrapbook

A web designer, Leia, searches for a collection of pictures for a customer's website. She stands in front of a large display and browses through the image gallery by swiping left or right to go to the next or previous item (cf. Figure 1(a)). One of the images catches her attention. Leia extends her arms and, in one swishing motion, pulls the image to be copied to the Scrapbook display (cf. Figure 1(b)). Satisfied with the result, Leia continues browsing for more pictures. Another one seems to satisfy her demands, and so she drags it over as well. Yet, there is something about that image that disturbs her. Deciding to fix it, Leia makes a graphical note on it using her right hand as a brush (Figure 4(a)). For some time, Leia continues browsing through the gallery and buffering pictures on the Scrapbook. Suddenly she changes her mind about the very first image and decides to remove it from the storage by saying "delete one" (Figure 4(b)).

The early version of XDKinect offered only parts of the time-based and multi-modal APIs. Support for flick hand gestures and the pick-and-drop metaphor [15] was added by tracking the user's arm movements, comparing their actions against a set of templates on the client side, and evaluating against the respective joints' coordinates tracked over time. The beginning of the settings API was already implemented, enabling developers to register only for selected joints. Speech recognition was available, but not yet dynamically configurable as all commands still had to be hard-coded on the server side. The cross-device communication API was close to the final version except for some wrapper functions.

Fotobook

Fotobook is based on the final version of XDKinect and exploits all available APIs. As before, we illustrate its usage by means of a sample scenario. A professional photographer, Luke, receives a last-minute call by a customer to do a web photo album for one of his recent shoots. While he could also look up the project on his main computer, Luke immediately looks at his wall-sized display. Usually in an ambient mode, the display exhibits an image slideshow to demo his work to visiting clients, and no direct interaction is possible (Figure 5(a)). However, as he walks towards it, the display detects his presence and switches into a personal interaction mode [16] (Figure 5(b)). Luke browses through the images by using his hand as a cursor [9]. Hovering over images that look promising, he quickly performs a push gesture in the air [9] to select images for the photo album. As he is still not quite happy with the album page's design, he again marks some of the images and rotates them by performing a mid-air circle gesture. Furthermore, he adds shadow and polaroid effects to the images using additional speech commands. Meanwhile, dinner time approaches, and Luke's friend, Han, comes over just as Luke performs a pick-and-drop interaction [15] to move the content to the main display. When Han enters the interaction zone, the ambient display flickers shortly to announce the presence of another user [1]. Han likes Luke's first design, but suggests adding a background image to the photo album page. He, too, selects an image from the interactive display to "copy" it to the main display. The system notices that Han, not Luke, performs this action, and places the image in a temp area. To finish the design, Luke picks it up and sets it as background.

In this scenario, the use of most XDKinect APIs comes together to realise interaction techniques known from related work [1, 9, 15, 16]. For example, the time-based and multi-modal APIs are used for the gesture and speech commands similar to Scrapbook, but in this case also employing Kinect-

(a) Ambient mode when the user is far away (b) Interactive mode when moving closer (c) Personal mode as the user interacts

Figure 5. Fotobook application using XDKinect with support for hand-as-cursor interaction and different interaction zones

Interaction's built-in press gesture to select images. In addition, the cursor's position is calculated by mapping Kinect skeleton coordinates into browser screen coordinates every time a new Kinect frame arrives. The proxemic API is used to switch between the ambient and interactive modes depending on whether a user is closer or farther than 2 metres from the display. Moreover, the system noticed the arrival of a new user, Han, by receiving information about another tracked skeleton associated with a new ID. As a result, it switches to an edit mode in which changes first have to be confirmed by Luke, the owner of the photo album.

EVALUATION

We conducted a user study to examine XDKinect's development experience. Following Olsen's guidelines [12], the goal of our evaluation is to demonstrate "reduced solution viscosity" and "ease of combination" by making use of XDKinect's various APIs as well as showing the potential to empower new users. The main task assigned to participants was to develop a simple cross-device application based on Fotobook from the previous section. The application consisted of two XDKinect clients—an ambient display showing an image gallery and another display to which the selected images would be copied (Figure 6). Users should be able to move a cursor with their hand and select images by performing a push or a grip gesture. A magnified version of that image would be transferred onto the main display. Users could then use speech to tag the selected image. The recognised text would be shown on an HTML5 canvas on the main display. Moreover, the ambient display should react to proximity of the user, switching from passive mode when users stand further away to interactive mode as they come closer. At the same time, the main display should show a notification when users join and leave the Kinect sensor's field of view.

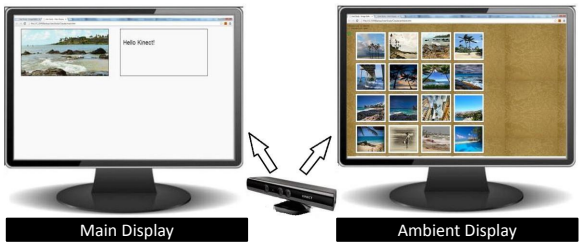


Figure 6. Study setup and target XDKinect application

Method

The study itself comprised 5 programming tasks, leading step-by-step to the fully-fledged XDKinect application just described. To give participants sufficient time to experience development using XDKinect, the study was anticipated to last around 1 hour. Each participant generally worked alone and was free to adjourn at any moment without completing all tasks. The tasks were carefully chosen to encompass every interaction dimension offered by XDKinect. For example, moving the cursor required skeletal tracking and management of the time-based API. The multi-modal API was reflected in the press/grip gestures for image selection as well as in speech recognition results being displayed on the canvas. Inter-client messaging was required to handle and sync interactions across the displays. The settings API was thoroughly exploited as well, as participants had to set and adjust many parameters including joints to be tracked, gestures to be recognised, proximity to the display, and speech grammar.

The study procedure was as follows. First, participants were given a short motivation about XDKinect and were introduced to the study setup. A background questionnaire then collected demographic information. In particular, they were asked to state their prior knowledge of Kinect—both as a user and developer—as well as rating their general web development, interaction design, and JavaScript/jQuery programming experience. For the rest of the study, participants worked mostly autonomously under our supervision. We supplied participants with a cheatsheet giving an overview of XDKinect's APIs, configuration options, and all supported events. All participants were provided with the same skeleton code in which they had to fill in the gaps. To focus the development process on XDKinect's features, we provided auxiliary functions to avoid implementing XDKinect-unrelated functionalities such as moving the cursor by translating the position using HTML, CSS and JavaScript. Once participants indicated that they were done, they completed a post-study questionnaire, asking them to express agreement with different statements on a 5-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree).

We recruited 12 participants (1 female) with a general background in computer science. Ages ranged from 24 to 39 years with a median of 28. Half of them had prior experience using Kinect and the other half did not. On a 7-point

	Background		Ease of Use							Time	Future Use
	Kinect	Kinect SDK	JS/jQuery	Skeleton	Gesture	Speech	Distance	Multi-User	Cross-Device	mins.	
P1	✓									55	
P3	✓									50	
P5	✓									55	
P7	✓									60	
P11	✓									45	
P12	✓									25	
P2										65	
P4										35	
P6										30	
P8										40	
P9										45	
P10										60	
Mean		1.92	3.42	4.42	4.33	4.82	4.50	4.82	4.67	47.08	4.33
Median		1	4	5	4.5	5	5	5	5	47.5	4

Table 1. Results from our study with 12 developers (first 6 had previous Kinect experience)

scale from 1 = Novice to 7 = Expert, participants reported on average medium web development ($mode = 3$), interaction design ($mode = 3$) and JS/jQuery programming experience ($mode = 5$). While some did have programming experience with the Kinect SDK, the overall rating was low ($mode = 1$).

Table 1 gives an overview of the participants' skills and their ratings. We discuss the results and our observations below.

Results

All participants were able to complete all tasks with almost no help from us. The minimum completion time amounted to 25 minutes, while the maximum completion time was 65 minutes ($mean = 47.5$, $sd = 12.5$).

In the post-study questionnaire, participants rated ease of use and effectiveness of XDKinect in different areas including skeletal tracking, gesture support and speech support, proxemic awareness, multiple users tracking, and cross-device communication. Table 1 shows mean and median ratings for ease of use. The ratings for effectiveness were very similar. All in all, the received feedback was very positive.

We analysed which current concepts and interaction dimensions had the most potential for improvement in terms of both ease of use and effectiveness. For this, we considered the lowest marks for all criteria. Speech recognition and multiple users tracking received the best marks—no one awarded a lesser grade than a 4 out of 5. Skeletal tracking, gesture and distance support also fared fairly well with a neutral 3 being the lowest mark. Cross-device communication was considered the least effective with a 1 by one participant, but was still considered easy to use with the lowest rating of 4. Although based on the principles described in [14], which were argued to be powerful and effective for many applications, some participants felt that our initial support for cross-device sessions [11] using instant messaging might be too simple.

We also compared median ease of use and effectiveness ratings between participants with and without prior knowledge of Kinect. Participants without previous experience awarded very high marks, resulting in the maximum median grade of 5 for all aspects. Participants who had at least used Kinect for gaming or development were slightly less generous, but still provided ratings in the range of 4 to 5.

We wondered to what extent users would want to use XDKinect in the future considering their previous background knowledge and the XDKinect development experience. Most participants were positive about this and, independent of previous Kinect experience, provided mostly high ratings between 4 and 5.

Apart from these ratings, participants were also asked to comment and make suggestions on future versions of XDKinect. Whilst not everyone provided additional feedback, the received comments were helpful for the overall assessment and identifying current drawbacks. The written feedback was analysed, grouped and ordered by frequency as in Table 2.

Comment	Count
Easy to use	5
More built-in gestures	3
Ability to define custom gestures	3
Documentation is too concise	2
Visual notification when Kinect detects the user	1
Server-side GUI to disable certain streams	1

Table 2. Users' comments and feedback

In summary, the overall feedback from the study was very promising. Both ease of use and effectiveness were consistently rated very high. In particular, 5 out of 12 participants praised XDKinect for its ease of use. The initial documentation was sufficient for all participants to complete all tasks. Still, there is room for improvement. The main criticism was the limited gesture support for both built-in gestures and the ability to define custom gestures. Currently, XDKinect only supports KinectInteraction gestures from the official Microsoft Kinect SDK. Given the flexible architecture and support for extensibility of XDKinect, gesture support could be enriched in two ways. First, the server-side could be extended with built-in gestures based on advanced gesture recognition algorithms for static and dynamic template matching and pattern recognition. Second, the client-side could be extended using simpler techniques similar to the $1-family of stroke recognisers [17]. Both support using custom templates.

CONCLUSION

In this paper, we presented XDKinect, a lightweight toolkit that facilitates the development of cross-device applications using Kinect to track interactions and respond accordingly.

The framework is adaptable to different application requirements and can be extended on both the client and the server side. XDKinect uses an event-driven architecture, where clients subscribe for information required from the server connected to a Kinect. We approached the task of developing XDKinect by re-implementing several example applications described in the literature, and also extending them with features to enable cross-device interaction, and attempting to eliminate the major technical challenges that developers currently encounter while building such applications.

Discussion

Despite the availability of several frameworks and open-source projects such as Kinected Browser [5], KinectJS and DepthJS, the idea of using whole-body gesture and speech commands as primary user input is still rarely implemented in existing applications. We argued that there are several issues and generally believe that there is a need for better experimentation methods and tools. One of the strongest features of XDKinect is its simplicity. As our study demonstrated, users found XDKinect's APIs intuitive and easy to use. Since XDKinect directly builds on web technologies, developers will benefit from existing experience in web languages and tools and be able to rapidly design and develop interactive cross-device applications. As Kinect is at the core of our framework, a number of new input dimensions are available to XDKinect applications that may better describe the use context in terms of users, devices and the environment.

Existing frameworks served as a source of inspiration for XDKinect. For example, [1] exploiting facets of proxemic interaction prompted us to introduce the stand-by and interactive zones for the ambient display scenario, where XDKinect takes cues from the user's distance to Kinect. Moreover, cross-device interaction techniques can be implemented with only little effort due to the flexible architecture and message-exchange protocol. We assessed the role of XDKinect in the context of existing research. Table 3 presents key concepts, available features and supported interaction dimensions of selected projects extracted from the papers and proof-of-concept implementations if available. However, our idea was not to prove that one solution is better than the other. Rather, we wanted to review and compare available frameworks. This means that if a certain feature is not supported by a framework, the lack thereof may not necessarily be considered a shortcoming. Moreover, while they share many principles and concerns, it is difficult to perform a direct comparison between each solution in terms of power and expressiveness and the types of applications that can be constructed. For example, frameworks such as Proxemic Media Player [1] and Interactive Ambient Displays [16] require special motion tracking hardware, whereas Kinected Browser [5] and XDKinect use Kinect as primary interaction medium.

An 'x' stands for a supported feature (partial or complete), while '–' denotes the absence of a feature. All listed frameworks support interaction for more than one user. Device-to-device awareness, i.e. knowledge about identity, position and orientation of other devices in the environment, is best supported by proxemic interaction frameworks. Only a small set

	cross-device comm.	device-device aware.	gesture	speech	distance	orientation	identity	motion	multi-user
XDKinect	x	x	x	x	x	–	x	–	x
Kinected Browser [5]	–	–	x	x	–	–	x	–	x
jQMultiTouch [10]	–	–	x	–	–	–	–	–	x
Shared Substance [2]	x	x	x	–	–	–	x	–	x
GroupTogether [7]	x	x	–	–	x	x	x	–	x
Interact. Ambient Dis. [16]	–	–	x	–	x	x	x	x	x
Proximity Toolkit [6]	x	x	–	–	x	x	x	x	x
Proxemic Media Player [1]	–	x	–	–	x	x	x	x	x

Table 3. Main features of XDKinect and other frameworks

of frameworks support cross-device communication similar to XDKinect. Also, the combination of supporting both gesture and speech input still seems to be a relatively rare feature, as it is incorporated only in XDKinect and Kinected Browser. However, both support only a subset of user-related proxemic interaction principles compared to Proximity Toolkit and GroupTogether. These applications make use of advanced motion tracking systems or multiple Kinects and can thus react on velocity and acceleration of user movement. They also leverage user identity for personalisation and safeguarding as in the Proxemic Media Player. Enhancing XDKinect with a richer proxemic API would in principle be possible and allow for a number of exciting use scenarios, but is currently constrained by the technical limitations of Kinect.

Limitations and Future Work

While XDKinect offers a solid base for a variety of cross-device interactive applications, a number of restrictions apply. Several limitations of the implementation stem from the underlying Kinect hardware and utilised libraries. First, gesture recognition currently manifests in a high rate of false positives. For example, a half-open hand is often interpreted as a grip gesture. Second, XDKinect does not support a dictation mechanism for speech recognition, so the vocabulary to be recognised must be conveyed to the application in advance. As mentioned previously, the multi-modal API relies on the Microsoft.Speech library. Its counterpart, the System.Speech library, permits free text dictation. However, using System.Speech in conjunction with Kinect SDK results in error prone recognition, since only Microsoft.Speech API possesses a language pack specifically calibrated for Kinect hardware characteristics. A further restriction of XDKinect is the ability to track only two people in detail. One possible solution for this shortcoming would be to extend the server-side to receive data from two Kinect sensors, but these then need to be positioned at a specific angle to avoid interference, which may be in conflict with intended interaction zones.

Other limitations concern features that XDKinect's APIs currently lack. As pointed out in the user study, XDKinect offers only a restricted set of built-in gestures and no API for defining custom gestures. To alleviate this shortcoming, the server-side module could be extended to recognise a set of popular gestures, such as swipe, zoom or clap hands, while the client-side component could be extended with an interface for user-defined gestures. In the future, we plan to incorporate

solutions from other Kinect research, e.g. for more efficient multi-modal input in speech and gesture interfaces [3], into the framework. In principle, any technique based on the standard Kinect SDK could be integrated with XDKinect. Additionally, our proxemic API could be extended to accommodate modalities suggested by [1]. The new Kinect sensor to be released in the near future appears to provide sufficient data required for determining user and device orientation and measuring velocity and acceleration.

In summary, despite some limitations, XDKinect is already able to support a rich set of cross-device interactive applications exploiting a variety of user and device interaction dimensions. The immediate future work comprises systematic performance evaluations and collecting more user feedback to continue improving the development experience. To this end, we have started to embed XDKinect in several student and research projects with promising initial results.

Acknowledgements

We thank the anonymous study participants for their time and feedback. Special thanks to Alexander Huber and David Ott for their support in developing and evaluating XDKinect.

REFERENCES

1. Ballendat, T., Marquardt, N., and Greenberg, S. Proxemic interaction: designing for a proximity and orientation-aware environment. In *Proc. ITS* (2010).
2. Gjerlufsen, T., Klokmose, C. N., Eagan, J., Pillias, C., and Beaudouin-Lafon, M. Shared Substance: Developing Flexible Multi-Surface Applications. In *Proc. CHI* (2011).
3. Hoste, L., and Signer, B. SpeeG2: A Speech- and Gesture-based Interface for Efficient Controller-free Text Entry. In *Proc. ICMI* (2013).
4. Jetter, H.-C., Zöllner, M., Gerken, J., and Reiterer, H. Design and Implementation of Post-WIMP Distributed User Interfaces with ZOIL. *IJHCI* (2012).
5. Liebling, D. J., and Morris, M. R. Kinected Browser: Depth Camera Interaction for the Web. In *Proc. ITS* (2012).
6. Marquardt, N., Diaz-Marino, R., Boring, S., and Greenberg, S. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proc. UIST* (2011).
7. Marquardt, N., Hinckley, K., and Greenberg, S. Cross-device interaction via micro-mobility and f-formations. In *Proc. UIST* (2012).
8. Melchior, J., Vanderdonckt, J., and Roy, P. V. A Model-Based Approach for Distributed User Interfaces. In *Proc. EICS* (2011).
9. Morris, M. R. Web on the Wall: Insights from a Multimodal Interaction Elicitation Study. In *Proc. ITS* (2012).
10. Nebeling, M., and Norrie, M. C. jQMultiTouch: Lightweight Toolkit and Development Framework for Multi-touch/Multi-device Web Interfaces. In *Proc. EICS* (2012).
11. Nebeling, M., Zimmerli, C., Husmann, M., Simmen, D., and Norrie, M. C. Information Concepts for Cross-Device Applications. In *Proc. DUI@EICS* (2013).
12. Olsen Jr., D. R. Evaluating User Interface Systems Research. In *Proc. UIST* (2007).
13. Paternò, F., and Santoro, C. A Logical Framework for Multi-Device User Interfaces. In *Proc. EICS* (2012).
14. Pierce, J. S., and Nichols, J. An Infrastructure for Extending Applications' User Experiences Across Multiple Personal Devices. In *Proc. UIST* (2008).
15. Rekimoto, J. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. In *Proc. UIST* (1997), 31–39.
16. Vogel, D., and Balakrishnan, R. Interactive Public Ambient Displays: Transitioning from Implicit to Explicit, Public to Personal, Interaction with Multiple Users. In *Proc. UIST* (2004).
17. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proc. UIST* (2007).