# Designing Peer-to-Peer Distributed User Interfaces: Case Studies on Building Distributed Applications

Eli Raymond Fisher, Sriram Karthik Badam, Niklas Elmqvist*

*School of Electrical & Computer Engineering, Purdue University, West Lafayette, IN, USA*

**Abstract**

Building a distributed user interface (DUI) application should ideally not require any additional effort beyond that necessary to build a non-distributed interface. In practice, however, DUI development is fraught with several technical challenges such as synchronization, resource management, and data transfer. In this paper, we present three case studies on building distributed user interface applications: a distributed media player for multiple displays and controls, a collaborative search system integrating a tabletop and mobile devices, and a multiplayer Tetris game for multi-surface use. While there exist several possible network architectures for such applications, our particular approach focuses on peer-to-peer (P2P) architectures. This focus leads to a number of challenges and opportunities. Drawing from these studies, we derive general challenges for P2P DUI development in terms of design, architecture, and implementation. We conclude with some general guidelines for practical DUI application development using peer-to-peer architectures.

*Keywords:* distributed user interfaces, case studies, design principles, lessons learned, implementation, DUI toolkits

## 1. Introduction

Parallelism—both on the local computer using multiple cores [1], as well as distributed across multiple virtual machines in the cloud [2]—has become the de facto solution to today's computational problems when Moore's law no longer is

*Corresponding author. Purdue University, 465 Northwestern Avenue, West Lafayette, IN 47907-2035, USA. Phone: +1 (765) 494-0364, Fax: +1 (765) 494-6951.

*Email addresses:* `fisher55@purdue.edu` (Eli Raymond Fisher), `sbadam@purdue.edu` (Sriram Karthik Badam), `elm@purdue.edu` (Niklas Elmqvist) able to help us stay abreast of the current data deluge facing society. However, the same limitations are also now starting to be felt in the user interface aspect of computer systems: while displays grow in size and shrink in price, the standard computers managing all these pixels are unable to cope. Furthermore, even as the number of mobile, embedded, and ubiquitous devices in our physical surroundings increases, we still do not have standard and widespread software infrastructures for binding all of these devices together into single, coherent interfaces where devices can reinforce instead of compete with each other.

*Distributed user interfaces* (DUIs) is an emerging research field studying this type of user interface architecture where components are distributed across different hardware devices in space and in time [3, 4]. Unfortunately, designing a distributed user interface is an order of magnitude more difficult than designing a standard single-device user interface due to issues such as synchronization, resource management, and data transfer. In practice, even if the literature of DUI systems is already rich with prime examples of distributed and situated interaction (e.g. [5, 6, 7]), there still exists very few concrete guidelines on how to design and build a distributed user interface application from the ground up. Beginning DUI designers are essentially left only with the alternative of trying to apply their knowledge from traditional user interfaces to the distributed setting.

In this paper, we address this shortcoming by deriving challenges, solutions, and design guidelines for developing DUI systems. While this treatment is inspired by the literature on distributed applications, we draw specifically from three in-depth case studies of DUI applications that we have built recently:

- SHARD: A distributed user interface media player designed for playing back media across multiple surfaces, speakers, and playback controls;
- MP-TETRIS: A multiplayer Tetris game designed for collaborative play across multiple input and output surfaces; and
- BEMVIEWER: A collaborative search system for multivariate data integrating both a digital tabletop and several mobile devices.

Informed by these three case studies, we discuss many of the common problems as well as their solutions encountered when designing DUI systems. Furthermore, we also enumerate general guidelines for designing, implementing, and evaluating DUI systems. All three of these case studies are based on a peer-to-peer (P2P) network architecture. Other network architectures have also been successfully applied to DUI development, particularly based on a client/server model (e.g. [8, 9, 10, 11, 12]). We delimit our treatment in this work to the unique challenges and opportunities afforded by a P2P architecture.

The remainder of this paper is structured as follows: We first introduce the three case studies and motivating examples for this paper. We then review the related work in distributed user interfaces, content redirection, and collaborative spaces. This literature review sets the stage for discussing the challenges for our case studies in particular, and distributed user interface applications in general. We describe the solutions we derived for our example applications, and how these were implemented. Finally, we draw upon the three case studies as well as the literature to discuss and formalize a set of design guidelines for building DUIs. We close the paper with our conclusions and ideas for future work.

## 2. Case Studies

Here we introduce each of the three motivating case studies that inspired this work. We also discuss the common usage scenarios, requirements, and design parameters for all three case studies. Following this section, we describe in more detail the challenges (Section 4) associated with DUI applications, as well as the concrete solutions (Section 5) we derived in meeting these challenges. We then generalize these ideas into guidelines and implications for design (Section 6).

### *2.1. Shard: A Distributed Media Player*

Rich device ecosystems are becoming increasingly common as mobile and ubiquitous computing are being embedded in our physical surroundings [4, 13]. An initial case study might focus on harnessing these device ecosystems for simple media playback. SHARD[1] is a truly distributed media player where digital media such as video, audio, and images would be entirely decoupled from its playback, allowing for highly flexible playback and control configurations.

### *2.2. MP-Tetris: A Multi-Player/Multi-Surface Game*

While the Shard case study above showcases many basic requirements of a distributed user application, it naturally does not cover the whole spectrum of possible applications. In particular, like many user applications, Shard is user-driven, which means that it merely responds to user input events such as button presses and menu selections. With the MP-TETRIS case study, our intent was to

[1]The name communicates our conceptual model of each display surface representing one of multiple shards of glass that all reflect the same digital media being played.

capture active simulation logic (a game engine automatically moving falling Tetris pieces) that is asymmetrically distributed in the system (only one participating peer will run the game engine). This gives rise to the challenge of transfering logic to other participants if the original peer disconnects or crashes.

Thus, MP-Tetris is a collaborative multiplayer Tetris game designed to be run on any configuration of display surfaces using any combination of input surfaces. It is cooperative in the sense that players must work together to eliminate lines on the playing field according to the classic Tetris, i.e., by filling lines completely with blocks. This task is made more challenging by the fact that pieces fall in real-time from the top of the screen towards its bottom, and that player pieces can optionally be subject to collisions.

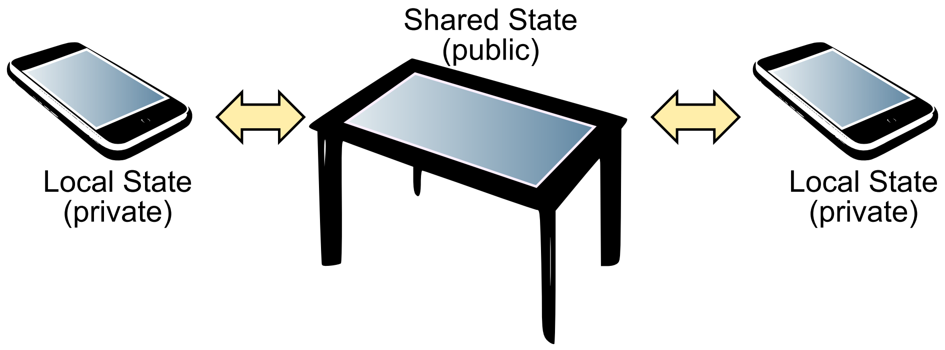


Figure 1: Public and private views in the BEMViewer system.

## *2.3. BEMViewer: Collaborative Search on Heterogeneous Devices*

MP-Tetris is a cooperative game, but collaboration is nevertheless not its main focus. In order to also capture any unique challenges and solutions generated by collaborative settings, we included the BEMVIEWER system as a case study as well. BEM is short for Branch-Explore-Merge [14] and is a protocol for collaborative search in multivariate datasets inspired by asynchronous revision control systems such as CVS, git, and Subversion. The protocol allows users to branch off from the current public query shared between all participants, explore data privately, and then merge back any new findings to the public state.

In validating the BEM protocol, we implemented a DUI system called BEMViewer (Figure 1). BEMViewer allocates the public shared state to a common display, such as a digital tabletop device, and uses mobile devices for the private state of each participating user. While we have presented the BEM protocol in a previous paper [14], our focus in the present paper is on the software engineering aspects of the BEMViewer, which have not been previously published.

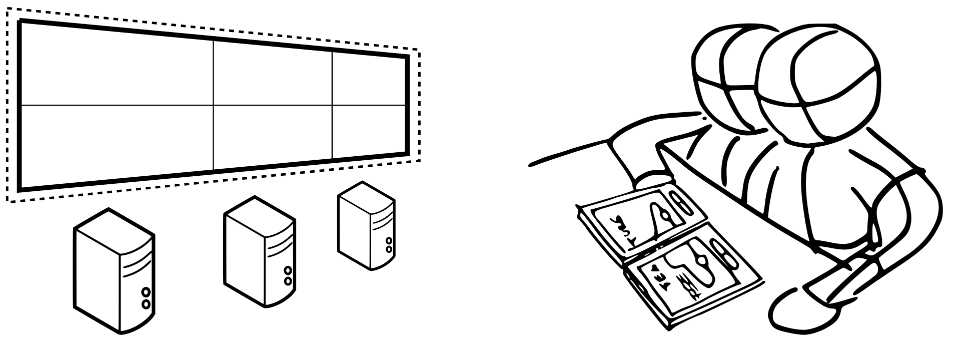

(a) Tiled-LCD displays. (b) Tiled mobile displays.

Figure 2: Example scenarios for the a distribured user interface application.

## *2.4. Common Usage Scenarios*

Some usage scenarios we envisioned for the case studies include the following:

- A display wall consisting of multiple tiled LCDs (Figure 2(a)) and multiple computers interacting with digital media that spans all of the displays;
- An output device displaying content located on another computer on the network without the need for a preconfigured server setup; and
- Two or more mobile devices rendering the same content that are placed side by side to form a larger picture spanning all of the displays (Figure 2(b)).

For example, two friends may want to place their tablets side by side to create a larger combined screen when playing MP-Tetris together. The playing field will now be split to span both screens instead of being replicated across both. A user may want to snuggle up in his bed with a tablet to watch an old DVD movie that he has loaded into his home desktop computer. Shard will now seamlessly stream the movie from the desktop computer to the user's tablet using the wireless network. Finally, a family may want to gather around their digital kitchen table to plan a trip by collaboratively searching for destinations, hotels, and restaurants that fit everyone's preferences. Here, the BEMViewer tool will allow the family members to work both independently on their personal mobile devices, as well as collectively on the multitouch kitchen table, to find an optimal destination in an efficient and timely manner.

The common theme for all of these scenarios is that they involve multiple (more than one) devices. More specifically, interaction in these use cases is distributed across different combinations of input, output, platform, space, and time [3]. Managing individual devices in such setups becomes increasingly tedious and error-prone as the number of devices increases. Therefore, a common goal for all these scenarios is to minimize the setup and initialization overhead involved in launching and controlling these device environments.

### *2.5. Common Requirements*

To maximize the value of the case studies in this paper, our goal is to be very explicit about the requirements, challenges, and solutions we faced during the design and development process. Based on the above usage scenarios, we were able to extract a number of design requirements that capture these scenarios:

R1 **Independent Storage and Computation.** One of the main common requirements for all three case studies is that there should be no dependency between the physical location of storage media, computation, and any input and output devices (besides a network connection).

R2 **Independent Output.** A key requirement is that distributed user interfaces should support output on multiple surfaces, potentially each showing only a portion of the visual output to support imagery spanning multiple displays. This requirement is of particular importance for collaboration where multiple individuals may need to view the output of the distributed user interface application on their own display.

R3 **Independent Input.** In the spirit of distributed interfaces, user input should also be decentralized so that any device can control an application, such as play, stop, or rewind, change collaborative queries, or move a Tetris piece. Again, this is particularly important for collaborative settings, where not only a single user but multiples ones may be interacting with the data using multiple devices simultaneously and independently.

Beyond the above design requirements, we also enumerate a set of non-functional requirements such as high rendering performance, platform independence, robustness, and broad support for multiple data and media formats.

### *2.6. Common Design Parameters*

All three case studies will be implemented as distributed applications running on multiple devices connected using the network (typically a TCP/IP network, although Bluetooth and other local networks are possible). Because the basic technical approach for all three applications will be the same, one goal of this work is to derive a common network infrastructure for DUIs that we can reuse for all three. Each participating device runs software that communicates using the network. Any device can choose to render all or part of the output (R2) as well as control the application (R3). Furthermore, to enable independent storage and computation (R1), the applications distribute content, media, and data from one source to multiple recipients in the local display environment.

## 3. Related Work

This paper spans topics in multi-display environments, distributed user interfaces, and output redirection. Below we review the literature in these fields.

### *3.1. Multi-Display Environments*

The notion of a distributed user interface can be traced back to early paradigms of multi-display (and multi-device) environments (MDEs) from fields such as CSCW, HCI, and computer graphics. An early example was the Spatial Data Management System (SDMS) [15] for multimodal interaction with a wall-sized projection display and a small touch display. Another seminal project, the Co-Lab meeting room [16], introduced fundamental concepts in collaborative interfaces, multi-display environments, and WYSIWIS (What You See Is What I See). Similarly, the DigitalDesk [17] combined an augmented desk, a camera, a tablet, and a projector. The i-LAND [18] system integrated several computational resources in the same physical space into what became known as a "roomware" system [19, 20]. Another example of such a roomware system was the office of the future project [21], which used the intrinsic geometry of an office to turn all surfaces into projected displays. In the same vein, the Stanford Interactive Workspaces project [22, 23, 24] focused on collaboration in technology-rich physical spaces; the implementation was called the iRoom.

### *3.2. Distributed User Interfaces*

Distributed user interfaces are those that distribute components across one or several of the dimensions input, output, platform, space, or time [3]. Several conceptual models exist for DUIs; examples include the CAMELEON-RT [25, 26], 4C [27], and the distributed model-view-controller (MVC) [28] models. The VIGO model [29], which lies at the core of the Shared Substance toolkit [5], is a clear influence to the DUI framework described in this paper.

While DUI applications can be built from the ground up, researchers have recently proposed various DUI toolkits intended to make this endeavor easier. The BEACH system [30] and Aura [31] frameworks were designed for managing dynamic resources in ubiquitous computing environments. Similarly, the Gaia [32] middleware infrastructure supports resource management in physical and interactive computing spaces (called Active Spaces). Additional frameworks include MediaBroker [33, 34], a recent toolkit for building peer-to-peer DUIs [35], and a visualization toolkit for distributed collaboration [36].

Some of the key observations we draw from these existing toolkits include transparent mechanisms for shared memory, peer-to-peer architectures to minimize setup and configuration, and the idea to replicate logic between peers.

### *3.3. Output Redirection*

Output or content redirection is the concept of decoupling the source and the display of digital imagery [37, 38, 39]; essentially, an application can execute on one device, yet display parts or all of its visual output on one or several other devices. Toolkits that support output redirection typically also support input redirection, where input events from one device can be handled by another device.

Several notable output redirection efforts exist. WinCuts [38] allow for rearranging screen regions, essentially providing same-computer output redirection. IMPROMPTU [37] shares applications across multiple displays using off-the-shelf computers and software to facilitate collaborative software development. In computer graphics, the notion of output redirection has been adapted to rendering clusters (or farms): early examples include the WireGL [40] and Chromium [10] toolkits that distribute OpenGL rendering on a command level, and recent parallel rendering frameworks include Equalizer [9], and OpenSG [41].

## 4. Challenges

Realizing the design for all three case studies requires overcoming several challenges. Many of these challenges apply generally to any DUI application. In this section, we review these concrete problems and derive the generalized challenges from them.

### *4.1. Consistency*

A key aspect for running multiple DUI instances on different devices is to maintain consistency between each instance. For a distributed media player, playback state must be consistently shared between instances, including the media file (or URL), the state of the player (stopped, playing, rewinding, etc), and the current location in the media. In the context of a real-time games such as Tetris, there are high consistency demands on the state of the playing field, the position of each player's piece, and player input. Similarly, for a collaborative search system, the shared query must be consistent across all peers to support effective search.

C1 Each software instance running on a device involved in a DUI application must maintain a shared and consistent state using the network.

### *4.2. Synchronization*

Synchronization is a mechanism for maintaining consistency, which has already been listed as a challenge (C1), but it is a detail that is worth recognizing as a challenge of its own. For Shard, playing back media on multiple devices requires that the individual devices are on the exact same position in the media file to avoid inconsistencies across individual displays. The same is true for a real-time game such as MP-Tetris. Even minor latency, i.e., on the order of a few milliseconds, are easily detected by a user for both video (spanning imagery across multiple screens) as well as audio (handling different audio channels on different devices, e.g., left and right audio for stereo). Similarly, for BEMViewer, synchronization is required when merging the private state of one peer to the public state on the tabletop device, and vice versa.

C2 Synchronizing the actions of software components on different hosts is a core concern for general distributed systems, and DUIs are no different.

### *4.3. Heterogeneous Hardware*

None of the case studies discussed above impose restrictions on the hardware devices connecting to the shared environment: the peers can be both mobile devices as well as personal computers. This is a challenge because it requires a platform-independent network protocol that can be implemented on each platform. For standard personal computers, this is not a major problem for standard personal computers given technologies such as Java. However, on the mobile phone market, the two main platforms—Apple's iOS and Google's Android—are largely incompatible and require very different implementations.

A corollary to this statement is that differing hardware will also have different capabilities and available resources. For example, one device may have a large screen, whereas another has no screen and only loudspeakers. Perhaps the user chooses not to use the screen on a mobile phone because of its small size, but still uses the device to connect headphones and as a glorified remote control.

C3 DUI applications must run on multiple different hardware platforms, yet leverage the unique capabilities of each platform and device.

## *4.4. Volatile Device Ecosystem*

Distributed user interface environments are inherently volatile—while a tiled-display wall (Figure 2(a)) is stable, mobile devices or laptops participating in the environment may join or leave the networked space at any point. For example, a person may temporarily connect their smartphone to Shard to use it as a remote control, but could then shut it down when it is no longer needed. Similarly, a person may want to leave a three-person MP-Tetris session at any time, and the user experience of the remaining two players should not be affected. This means that the DUI environment cannot depend on individual devices to function properly.

C4 DUI applications must be robust against devices joining or leaving the shared environment at any point (sometimes not gracefully).

## *4.5. Limited Resources*

Some storage, computational, or hardware resources are limited and cannot be easily distributed. This challenge is best illustrated for the Shard media player. In Shard, the user may only possess a single copy of a DVD or Bluray movie, yet may still want to run the movie on multiple devices, such as several computers collectively driving a display wall. In fact, even when the media is digital and easily transferable using shared drives, e-mailing, or downloading from a public website, Shard should not require the user to manually perform such actions.

In other words, the physical media is a limited resource that only one of several devices has access to, yet which must clearly be shared with the other devices to enable the player to function properly. Of course, if the device holding the media abruptly leaves the shared environment, the playback should gracefully stop.

The limited resources challenge also applies to the BEMViewer case study. Perhaps the dataset being visualized only exists on the file system of one of the participating devices, such as the tabletop computer. One solution may be to either manually copy the file to all participating devices, or to use a shared filesystem connecting them. However, these approaches both require additional work to achieve, and a better alternative would be that the DUI framework directly supported data and resource sharing.

C5 The storage, computational, or physical resources available for each device in a DUI application may be different, causing that resource to be *limited* to one or a few devices.

*4.6. Data Transfer*

Given a limited resource (C5) such as a file, DVD, or Bluray disc, the contents must clearly be transferred over the network from one source device to multiple destination devices in the DUI environment. This data transfer challenge is different than state for supporting consistency (C1); instead of the bookkeeping used to synchronize devices (C2), this requires a high-capacity transfer mechanism.

C6 DUI applications need support for transferring *binary large objects* (blobs)– such as media, images, files, databases, documents etc—between devices.

*4.7. Physical Space*

Since a DUI environment is highly volatile (C4), with devices joining and leaving at any time, a distributed application cannot depend on any particular device. As a result, each individual device must be autonomous, being able to decide for itself how to render visual output, handle events, and react to input. Physical space is therefore a key aspect to DUI applications: the physical location of a particular device in relation to other devices will help the device in making these decisions. For example, a computer that forms part of a tiled-LCD display needs to know its tile position to decide which region of the visual output to render.

C7 Physical space must be managed on a global level for a DUI application so that individual participating devices can make autonomous decisions.

*4.8. Asymmetric Functionality*

The components of DUI applications are not necessarily uniform. While it is often desirable that different peers in a DUI space share as much code as possible from a pure software engineering and maintenance point of view, it is also true that certain aspects of a software system require asymmetric functionality. Resources differ and are limited between different peers (as already captured in C5), and even for otherwise uniform devices, there may be a need for one or several of the individual peers to have certain roles. For example, in the MP-Tetris game, there is a need for maintaining a single simulation engine responsible for enforcing the falling motion of the player pieces as well as for detecting collisions between these pieces and the current playing field. If this were not the case, maintaining consistency between multiple competing simulation engines would be challenging. Similarly, for the BEMViewer case study, there is a very clear division of labor between the device managing the public state (often a digital tabletop device) and the devices managing the private state of individual users (a smartphone or tablet).

Of course, asymmetric functionality gives rise to issues for fault-tolerance and robustness. What happens if a peer with unique logic crashes or shuts down? How can the system recover and which peer should take over after the missing peer?

C8 Distributed applications often contain unique components that are asymmetrically divided between participating devices.

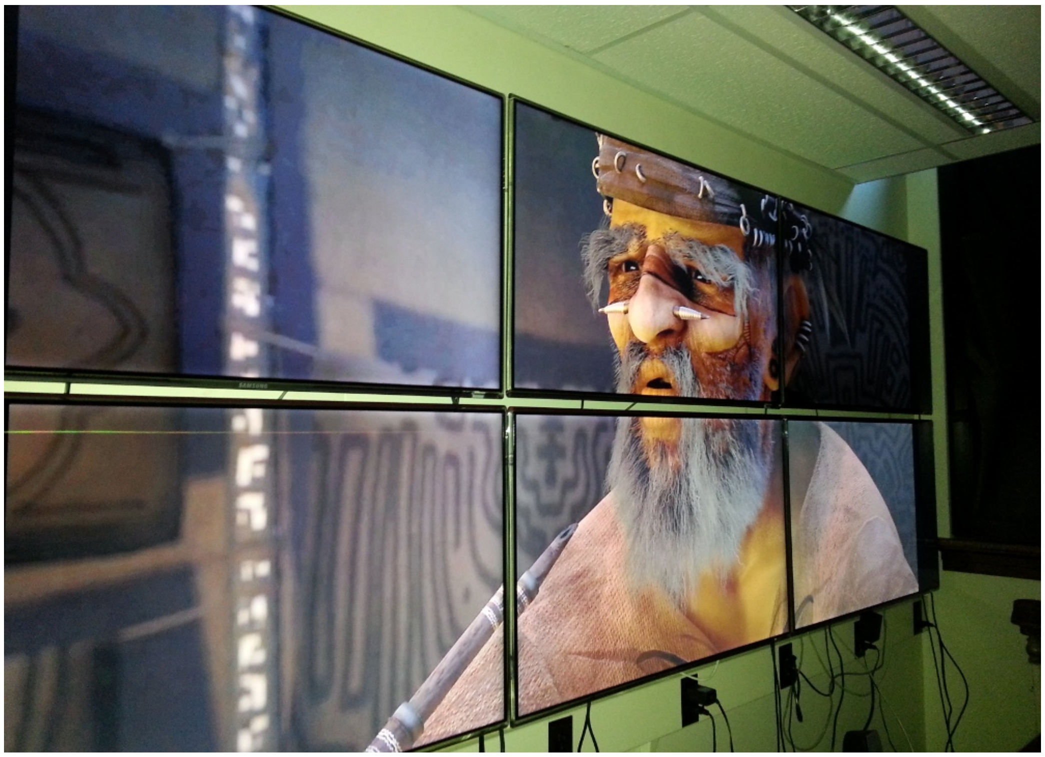


Figure 3: Shard running on a 3×2 tiled LCD display wall where each column of displays is driven by a separate computer. Image credits from the Creative Commons movie Sintel.

## 5. Solutions

We implemented all three DUI applications above as peer-to-peer distributed applications implemented in Java where each participating device runs *peer software* and are connected over the network in a common *shared space*. The environment contains both a shared and replicated *associative memory* for maintaining state, as well as a shared *event channel* for exchanging real-time information. Figure 4 shows the architecture of a DUI space designed in this fashion.

Naturally, each case study needed slightly different solutions for their task:

- **Shard:** Media decoding and playback is performed using the vlcj[2] Java bindings for the VideoLAN[3] vlc media player. Android peers use the standard Android SDK, including its video playback functionality.
- **MP-Tetris:** 2D vector graphics and animation is implemented using the the Piccolo2D [42] library. Android peers use the standard Android SDK.
- **BEMViewer:** The tabletop peer software uses the Piccolo2D [42] structured vector graphics library as well as the OpenStreetMap API. The Android peers use the Google Maps API.

### *5.1. Common Network Infrastructure*

In realizing the three DUI case studies discussed in this paper, we found ourselves reusing essentially the same network infrastructure for all implementations. The participating devices were connected over LAN or Wifi (for mobile devices). The core functionality of this infrastructure is replicating state across all participating peers, thus meeting the basic consistency (C1) challenge. The shared memory is a variant of an *event heap* [43, 44], which in turn is an adaptation of tuple spaces [45] and T-Spaces [46] for ubiquitous computing applications. It contains of shared objects of standard or user defined data structures. We implemented methods to create and update these objects in the shared memory, along with event handlers to inform other DUI peers when a shared object is created or updated. Each DUI peer has its own rendering engiene over this shared memory that handles the user interface.

[2]`http://www.capricasoftware.co.uk/vlcj/`
[3]`http://www.videolan.org/`

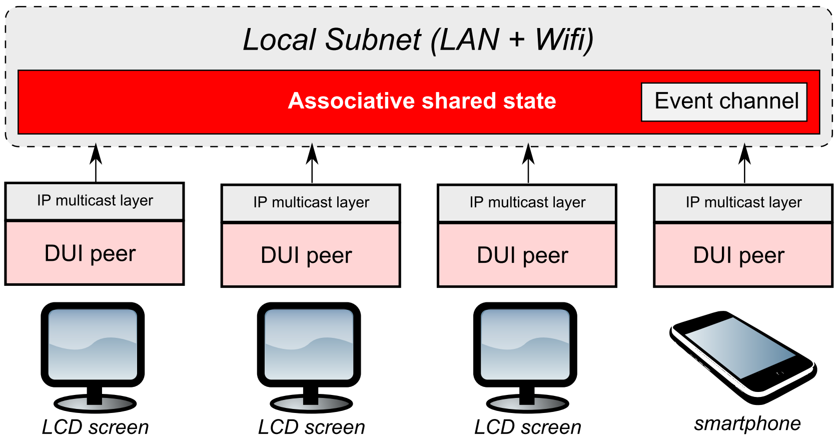


Figure 4: Example of the peer-to-peer network architecture common to all our case studies. This example shows three computers driving a tiled-LCD display and a mobile device connected to the shared display environment.

All the three case studies were implemented in Java with JGroups toolkit [47] for communication and various graphics libraries such as Piccolo2D, Java2D, and vlcj Java bindings for rendering. JGroups offers reliable multicast using a flexible protocol stack that supports different transport layer protocols like UDP (IP multicast), TCP, JMS. We used UDP for communication between peers connected over a local area network. In non-multicast environments, we used the TCP daisy-chain protocol to perform multicast through multiple $(n - 1)$ unicast messages. The choice of the rendering engiene was based on the case study. For Shard, vlcj Java bindings provide the ability to play a wide range of media types in a GUI window created using Java Swing. The vlcj bindings also allow creation of a server that streams the media from a source URL. The reason and implications of using this functionality have been discussed in section 5.3. As mentioned earlier, we used Piccolo2D and Android Java2D libraries for the rendering engienes in the other two case studies.

## 5.2. *System Architectures*

For Shard (Figure 3), each new peer creates a *player object* in the shared memory that all other peers automatically subscribe to, causing them to be notified when its state changes. All player objects contain the same data: the state of the player, the media URL, and the current position in the media being played back (if any). Peers are only responsible for modifying their own object state, but will automatically respond to changes in other player objects (by matching player state changes). This means that any participating device can change its own state, e.g., from playing to stopped, causing a cascade of changes in other player objects.

For MP-Tetris (Figure 5), the shared memory is used to store the playing field as a singleton shared object (created by the first peer to establish the DUI space), as well as player objects (similar to Shard) for each individual peer. The playing field object captures the current positions and colors of static blocks. Player objects, on the other hand, maintain the shape, color, position, and rotation of the player's currently controlled piece. Finally, a unique simulation engine component (discussed below) runs on one of the participating peers.

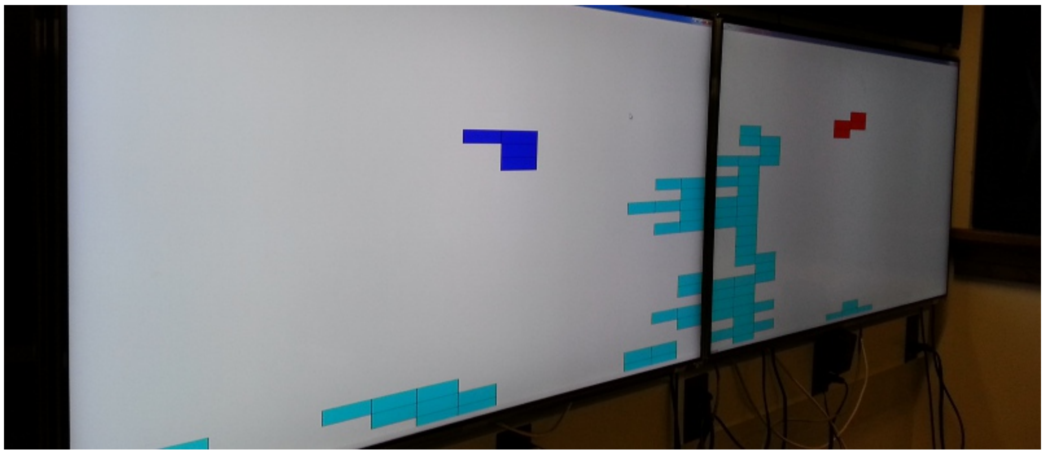


Figure 5: MP-Tetris played on a dual-display setup with two players. Colors are used to differentiate player blocks instead of block type.

Finally, for BEMViewer (Figure 6) the key data structures include the search query term that the collaborating users are progressively building, as well as the dataset being analyzed. Furthermore, there are two distinct and asymmetric types of peers in the BEMViewer system: the *public peer*, which has a large public display and is used for coupled work, and *private peers* that are restricted to individual users.

## *5.3. Replicating State*

Transfering state and data across different peers (C6) takes slightly different forms depending on the particular application. For example, the BEMViewer application makes use of the shared associative memory for its dataset, which causes the data to be automatically replicated to all participating peers. MP-Tetris uses shared memory in the same way for its playing field and player objects. This

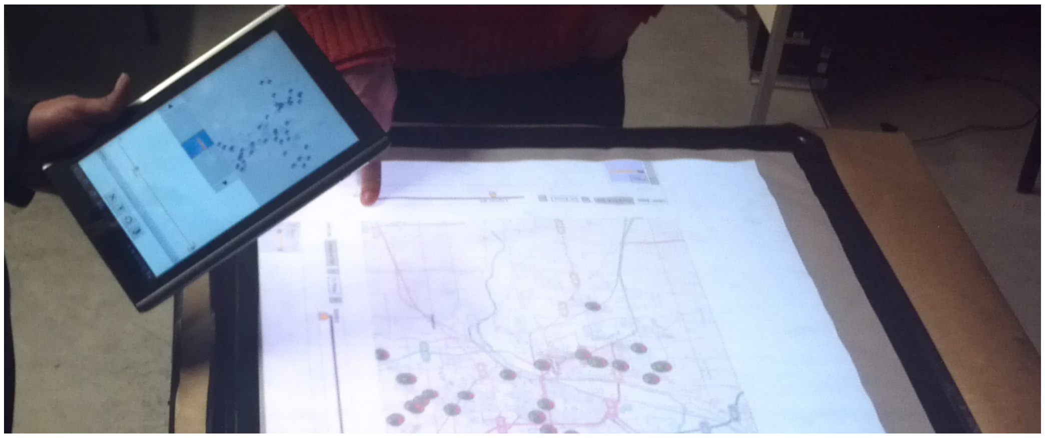


Figure 6: Two participants using an Android tablet and digital tabletop display to interact with BEMViewer for a real estate collaborative search task.

approach may not be scalable for large data volumes, however, because our peer-to-peer network infrastructure is not optimized for high-performance data transfer.

Shard is particularly interesting due to the large volume of data associated with digital media playback. Using the vlcj bindings, Shard can play back any form of media, including DVD, Bluray, local files, and remote URLs (even Youtube videos). However, as described above (C5), many times a particular media only exists on one of the devices connected to the Shard space. Even if the media is available on the network as a URL, it would be wasteful for each individual device to stream the media from that URL.

Instead, Shard replicates media by creating an ad hoc RTP (Real-time Transport Protocol) [48] server that streams the media from the source URL, and then in turn locally streams the media to the other peers in the Shard space. This media replication is asymmetric, since one of the devices becomes an ad hoc server, and the others become clients. All peers contain the functionality to either run as a client or as a server; the determination is done at run-time by whichever device a user launches the media from (in fact, for the server case, the peer still opens a client that connects to the local server). If the media is a DVD or Bluray disc local to the computer, the media is replicated from that single disc to all connected peers. Roles can also change if a user launches media on another device.

Beyond solving the limited resource challenge (C5), this functionality also highlights the Shard solution to the data transfer challenge (C6). The ad hoc RTP server is an out-of-band data channel that does not use the shared memory described above. Instead, the shared memory is used to communicate the IP address and port of the newly launched server, but the heavy-duty data transfer happens using the RTP protocol through separate TCP/IP ports.

## *5.4. Synchronization*

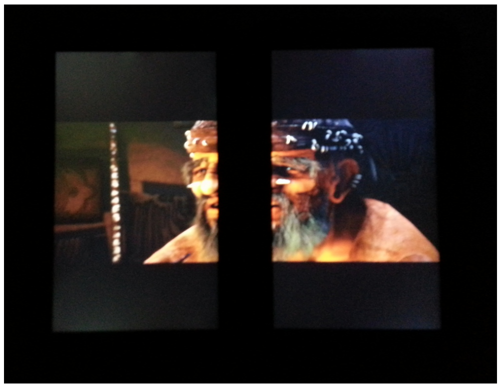
While the notification mechanism in the common network infrastructure ensures synchronization between different player peers on a high level, it is not sufficient for the fine-grained synchronization necessary for media playback in Shard (C2). The RTP client and server logic used for streaming media inside the Shard space does contain synchronization mechanisms that corrects for minor lag, but this is not always sufficient for large differences in media position.

To remedy this, each Shard peer monitors the media position of all other players. If the difference between the position of the leading player (i.e., the player that is furthest along in playing back the media) and its own position becomes too large, the player will attempt to jump ahead to catch up. This can be difficult since the player must take into account network latency as well as the impact of potentially trashing and rebuilding the stream buffer. Jumping too short may mean that the player does not catch up, whereas jumping too far will overshoot the media position and may cause leapfrogging behavior as other players now try to catch up to the new leader. Our solution is to maintain latency statistics and jump moderately to within the tolerance where the RTP functionality provided by vlcj can correct for the difference in media position.

## *5.5. Space Management*

Our DUI network environment maintains a global configuration for the physical space in which the distributed application is being executed (C7). This configuration includes the physical location, size, and orientation of each fixed display that may potentially participate in the environment. Individual devices are thus aware of the entire display environment as well as their own properties. This knowledge is necessary for peers to independently determine which 2D region of the visual representation to render. Without a global space configuration, each device will have to be manually configured every time it is launched.

Figure 3 shows a screenshot of Shard running on a $3\times2$ tiled LCD display wall. In this example, taking both the geometry of the entire surface as well as the dimensions and aspect ratio of the media into account allows each Shard peer to determine exactly the coordinates of the region to render on its own display.


(a) Two tiled Android tablets.

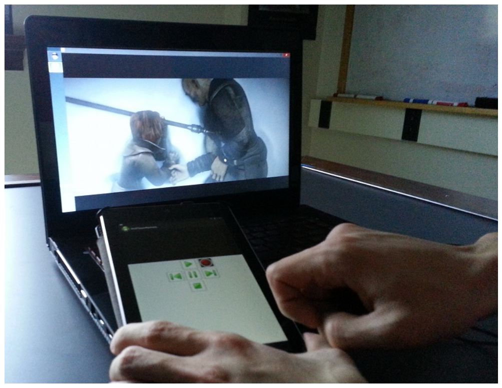

(b) Shard on laptop with Android remote.

Figure 7: Shard running on Android. Image credits from the movie Sintel (Creative Commons).

### *5.6. Mobile Devices*

Space management is important for fixed displays, but is less useful for mobile devices that have no predefined position in a physical space. Instead, we let the user configure the display layout. Because JGroups currently has no iOS port, our peer-to-peer network infrastructure currently only supports Android devices. Figure 6 showcases the BEMViewer system spanning both a digital tabletop device powered by a personal computer running Java, as well as an Acer Iconia tablet running our Android software. Two additional examples are given in Figure 7. Figure 7(a) depicts a Shard playback spanning two Samsung Galaxy Tab tablets, whereas Figure 7(b) shows an Android tablet being used as a Shard remote.

Including mobile devices in our case studies is a useful exercise due to their volatile nature (C4): a mobile device could easily join or leave a DUI space at any time. However, the use of multiple shared objects (associated to specific devices) in shared memory means that even if a device disappears without cleaning up, the operation of the remaining DUI space will not be affected. The individual shared objects associated with the lost device will no longer be updated, but this has no bearing on any of the other devices and their objects (even if the device rejoins the space, causing it to create new shared objects).

### *5.7. Managing Asymmetric Functionality*

For the Shard media player, if the peer playing back a limited media resource leaves, there is not much the system can do besides freeze the playback. In other words, while this is an example of an asymmetric (C8) and limited (C5) resource, the system cannot easily recover from this situation.

For the MP-Tetris game, on the other hand, it should certainly be possible to recover from the peer controlling the simulation logic disconnecting or crashing. Controlled exits are easiest to deal with: when a normal peer leaves a running game, the peer will first remove its own player object, thereby causing that piece to disappear from the playing field on all other peers. For a simulation peer, the only additional step is to automatically appoint a new simulation peer before leaving. This can be done randomly or using an ordered list.

Recovering from crashed nodes is more challenging, however. While our implementation currently has no optimal solution, one approach is to detect when the simulation logic is no longer being regularly updated. This would be an indication that the simulation peer has crashed, and the next peer in the list can self-appoint itself. A similar scheme can be used to detect crashed nodes and automatically clean up the corresponding player objects in response.

## 6. Implications for Design

The three case studies presented in this paper are instructive because their designs illustrate many of the key challenges inherent in DUI application design for peer-to-peer architectures. In this section, we generalize our implementations into design guidelines, discuss the limitations of our work, and discuss future extensions to the framework.

### *6.1. Design Guidelines*

Generalizing from our implementation in Section 5, we derive the following basic design guidelines on DUI application design based on peer-to-peer network architectures:

D1 **Prefer P2P.** It is often not practical for a distributed user interface to require the user to first configure and run a dedicated server. Client-server based frameworks for distributed systems face issues with scalability, robustness, security, and trust [49, 35]. Furthermore, multicast communication is known to decrease the bandwidth requirements for parallel rendering [50]. We observed that our shared memory based approach, handled by P2P IP multicast, provided flexible ways to overcome challenges (discussed in Section 4) in building distributed user interfaces. Several successful DUI systems [5, 35, 51, 52, 53] have used P2P framework (IP Multicast) for communication after considering the advantages it provides, thus making it a design decision that influences the architecture rather than just an implementation decision.

It should be noted that peer-to-peer is by no means a requirement for successful DUI development, and several excellent approaches based on client-server architectures exist [8, 9, 10, 11, 12]. Our focus in this paper is merely on peer-to-peer approaches for the above-mentioned reasons.

D2 **Replicate memory.** Given the use of a peer-to-peer architecture, message passing and more complex synchronization methods to maintain consistency (C1) is not always practical. Using a shared associative memory (event heap) inspired by tuple spaces [45] has been shown to be particularly powerful and flexible for ubiquitous computing environments [43, 44].

D3 **Replicate behavior.** While previous multi-target rendering systems such as Chromium [10], WireGL [40], and Equalizer [9] use centralized architectures that distribute commands to clients, today's device ecologies are easily powerful enough that virtually all of the application logic can be replicated on each individual device [5, 54]. This means that each device will run symmetric peer software in a flat peer-to-peer configuration and only require minimal state synchronization instead of command messages.

D4 **Focus on the network protocol.** The flexibility and robustness of any DUI application is largely dependent on its network protocol. A well-designed peer-to-peer network protocol provides high performance in support of consistency (C1) and synchronization (C2), platform-independence to allow for heterogeneous devices connecting to the environment (C3), and robustness against peers joining and leaving the environment at any time (C4).

D5 **Allow out-of-band communication.** Virtually all DUI applications need a mechanism to transfer binary large objects such as raw data, media, documents, and other files between devices (C6). While it is often possible to use the shared memory to transfer such data, this may become inefficient if the media size is large. For example, a distributed web browser needs to transfer images between peers, and a distributed media player, indeed, must stream media from one source peer to all of the others. This means that DUI frameworks should provide an out-of-band communication mechanism, such as a shared file system (easiest), cloud storage, or a dedicated out-of-band protocol (such as the RTP client/server solution used in Shard).

D6 **Manage physical space.** DUI applications should provide a space management mechanism to enable individual peers to utilize knowledge of their own location in physical space in relation to the rest of the environment. For example, in the context of Shard, allowing a particular computer to know the physical location of its output display makes deciding which parts of the media playback to render a trivial issue. Sensing the location of a mobile device in physical space [55] is key to realizing interaction techniques such as peephole displays [56, 57] and imaginary interfaces [58, 59].

## *6.2. Generalizations and Limitations*

The challenges, solutions, and design guidelines presented in this paper are essentially a synthesis of our previous experience building DUI toolkits and applications (e.g., [3, 54, 36]). The case studies here were chosen because they exhibit most of the common pitfalls, design dilemmas, and technical challenges encountered in general DUI design (although not necessarily at once). We believe that these challenges are unique to DUIs, yet very common whenever designing and building a DUI application. For example, any productivity DUI application will have to solve the issue of replicating binary data—such as images, documents, or general files—across hosts, either using a shared file system or a dedicated out-of-band data transfer channel.

Having said that, while our argument above is that the findings in this work go beyond the illustrative case studies presented here, we do not claim that this is an exhaustive review of challenges and design guidelines for distributed user interfaces. We are prepared to say that this paper reviews some of the most *important* of these challenges and guidelines, but the field of distributed user interfaces is still new and much more work remains to be done. Furthermore, we have gone to some lengths to cite and discuss related work from the DUI and related research areas that demonstrate much of the same challenges and successful solutions to addressing them. Additional technical challenges will no doubt emerge as traditional single-computer user applications increasingly begin to become distributed in scope. Several issues that we have not studied include migratory and migratable interfaces [8, 60], asymmetric application logic (see below), standardized infrastructures, security, privacy, etc.

## *6.3. Supporting Asymmetric Functionality*

The single significant challenge **not** fully addressed by our network infrastructure is managing asymmetric application logic (C8). A complex multi-device environment may include input devices that are decoupled from their event handlers, for example, a Microsoft Kinect depth camera detecting a user's full body interaction anywhere in a physical space. While the methodology for handling this type of events is similar to the event handling in our current framework, determining *which* peer should handle an individual event is not. This calls for asymmetric peer software, where certain devices run special-purpose software for simulation and event management.

However, asymmetric functionality gives rise to a new array of problems in terms of fault-tolerance and robustness. Some of this is already manifest in our case studies: for example, shutting down the peer controlling the simulation logic in MP-Tetris means that the simulation must now migrate to another peer. While MP-Tetris has some support for this by appointing the next peer in line to run the simulation, a more general approach is needed. Similarly, shutting down the Shard peer that temporarily is hosting the media (a limited resource—C5) will immediately halt playback on all the other devices. This is a direct effect of Shard peers not being fully symmetric across all devices at all times.

One approach to support asymmetric functionality is through the use of loosely coupled *services* that are specific to individual devices, yet which are launched as part of an *assembly* of services and communicate using shared memory. Exploring these concepts is left for future work.

### *6.4. Network Architectures: Peer-to-Peer vs. Client-Server*

Our focus in this work has been to study DUI software development through the lens of peer-to-peer network architectures. P2P architectures are attractive because they decrease centralization and thus increase scalability and robustness in a distributed system. However, they are often more difficult to implement and raise challenges for central tasks such as synchronization and consistency. Our intention with this work is not to promote one approach over the other, but rather to focus on DUI software development within a peer-to-peer network architecture. Nevertheless, in this subsection, we will discuss some of the pros and cons of choosing P2P over a client-server scheme.

Efficiency is one of the first advantages listed by P2P proponents due to eliminating a common potential bottleneck in the architecture. The shared memory in our case studies consisted of about 50 shared entities and we found that it typically took less than one second to create, share, and render them (in case of MP-Tetris) on two peers connected using a Gigabit Ethernet connection. Our solutions for state replication, synchronization, and space management take advantage of direct communication between devices in contrast to a client-server model. However, since this comparison was not a goal of our work, we did not conduct any performance measurements comparing P2P and CS implementations. We rely on ample existing work (e.g. [49, 35]) for such performance comparisons.

Asymmetric functionality such as handling exiting peers, maintaining consistency, and resolving conflicts is likely easier to perform with client-server model due to the centralization in this model. In our case studies, the frequency of handling asymmetric tasks was lower than synchronizing peer or maintaining consistency, so the overhead in performing the asymmetric functions at specific peers was outshadowed by the reliable and faster communication provided by JGroups IP multicast.

Communication within DUI systems corresponds to either data transfer or state management. Data transfer typically involves sending a dataset to one or more peers when needed, while state management ensures synchronization and consistency. As discussed before, sometimes the data to be transfered can be so large (e.g., streaming digital media) that it cannot be efficiently replicated over the peer-to-peer network. In the Shard case study, where high quality media present on one peer can be played on a display wall with multiple screens, our solution was to handle such media transfers by running a RTP server and streaming the videos to all the peers through an URL. This represents a hybrid model that incorporates a client-server model when necessary, and uses our peer-to-peer model otherwise. We think this is a good indication of how future DUI architectures and frameworks may be designed; selecting the optimal features from different models and combining them into hybrids as necessary.

## 7. Conclusion and Future Work

We have presented three case studies of distributed user interface applications, including a distributed media player, a real-time multiplayer game, and a collaborative search system for hybrid devices. Instead of focusing on specific aspects of DUIs such as visual representations, interaction, or collaboration, our emphasis in this paper has been on the practical software engineering challenges inherent with building DUI applications. Summarizing across all three case studies, we first derived challenges that apply widely across distributed user interfaces in general. We then presented the common solutions we found to address them. Finally, we took a step backward by deriving general design implications and guidelines that apply broadly to any DUI project, academic or commercial alike.

Our future work will entail further exploring the design space of distributed user interface applications using the network infrastructure developed here. In particular, whereas these case studies are standalone applications designed to run in exclusive mode, we are planning to introduce higher software layers to enable multiple distributed applications to run in parallel. Distributed applications are clearly integral to a ubiquitous computing future, and much additional development is required on infrastructures capable of enabling such a vision.

## Acknowledgments

This material is based upon work supported by the U.S. Department of Homeland Security's VACCINE Center under award no. 2009-ST-061-CI0001, as well as by the U.S. National Science Foundation under awards IIS-1249229 and IIS-1227639. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

We thank the Purdue University Summer Undergraduate Research Fellowship (SURF) program and the Semiconductor Research Corporation (SRC) for supporting the first author's internship during summer 2012. The movie images used in this paper are released under a Creative Commons license, copyright © Blender Foundation (`http://www.sintel.org/`).

## References

[1] G. Blake, R. G. Dreslinski, T. Mudge, A survey of multicore processors, IEEE Signal Processing Magazine 26 (6) (2009) 26–37.

[2] R. L. Grossman, The case for cloud computing, IT Professional 11 (2) (2009) 23–27.

[3] N. Elmqvist, Distributed user interfaces: State of the art, in: Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Springer, 2011, Ch. 1, pp. 1–12.

[4] J. A. Gallud, R. Tesoriero, V. M. R. Penichet (Eds.), Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem, Springer, 2011.

[5] T. Gjerlufsen, C. N. Klokmose, J. Eagan, C. Pillias, M. Beaudouin-Lafon, Shared substance: developing flexible multi-surface applications, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, 2011, pp. 3383–3392.

[6] N. Marquardt, R. Diaz-Marino, S. Boring, S. Greenberg, The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2011, pp. 315–326.

[7] H.-C. Jetter, J. Gerken, M. Zöllner, H. Reiterer, N. Milic-Frayling, Materializing the query with facet-streams – a hybrid surface for collaborative search on tabletops, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, 2011, pp. 3013–3022.

[8] K. A. Bharat, L. Cardelli, Migratory applications, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 1995, pp. 133–142.

[9] Equalizer, `http://www.equalizergraphics.com/`.

[10] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, J. T. Klosowski, Chromium: a stream-processing framework for interactive rendering on clusters, ACM Transactions on Graphics 21 (3) (2002) 693–702.

[11] H.-C. Jetter, M. Zöllner, J. Gerken, H. Reiterer, Design and implementation of post-WIMP distributed user interfaces with ZOIL, International Journal of Human-Computer InteractionTo appear.

[12] M. A. Nacenta, S. Sakurai, T. Yamaguchi, Y. Miki, Y. Itoh, Y. Kitamura, S. Subramanian, C. Gutwin, E-conic: a perspective-aware interface for multi-display environments, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2007, pp. 279–288.

[13] M. Weiser, The computer for the twenty-first century, Scientific American 3 (265) (1991) 94–104.

[14] W. McGrath, B. Bowman, D. McCallum, J.-D. Hincapie-Ramos, N. Elmqvist, P. Irani, Branch-explore-merge: Facilitating real-time revision control in collaborative visual exploration, in: Proceedings of the ACM Conference on Interactive Tabletops and Surfaces, 2012, pp. 235–244.

[15] R. A. Bolt, "Put-That-There": voice and gesture at the graphics interface, Computer Graphics (SIGGRAPH '80 Proceedings) 14 (3) (1980) 262–270.

[16] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, D. G. Tatar, WYSIWIS revised: Early experiences with multiuser interfaces, ACM Transactions on Office Information Systems 5 (2) (1987) 147–167.

[17] P. Wellner, Interacting with paper on the DigitalDesk, Communications of the ACM 36 (7) (1993) 86–96.

[18] N. A. Streitz, J. Geissler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, P. Seitz, R. Steinmetz, i-LAND: An interactive landscape for creativity and innovation, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, 1999, pp. 120–127.

[19] N. A. Streitz, P. Rexroth, T. Holmer, Does 'roomware' matter? investigating the role of personal and public information devices and their combination in meeting room collaboration, in: Proceedings of the European Conference on Computer-Supported Cooperative Work, 1997, pp. 297–312.

[20] N. A. Streitz, P. Tandler, C. Müller-Tomfelde, Human-Computer Interaction in the New Millenium, Addison Wesley, 2001, Ch. Roomware: Towards the Next Generation of Human-Computer Interaction based on an Integrated Design of Real and Virtual Worlds, pp. 553–578.

[21] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, H. Fuchs, The office of the future: A unified approach to image-based modeling and spatially immersive displays, Computer Graphics (SIGGRAPH '98 Proceedings) 32 (1998) 179–188.

[22] A. Fox, B. Johanson, P. Hanrahan, T. Winograd, Integrating information appliances into an interactive workspace, IEEE Computer Graphics and Applications 20 (3) (2000) 54–65.

[23] B. Johanson, T. Winograd, A. Fox, Interactive workspaces, IEEE Computer 36 (4) (2003) 99–101.

[24] B. Johanson, T. Winograd, A. Fox, Invisible computing: Interactive workspaces, IEEE Computer 36 (4) (2003) 99–103.

[25] L. Balme, A. Demeure, N. Barralon, J. Coutaz, G. Calvary, CAMELEON-RT: A software architecture reference model for distributed, migratable, and

plastic user interfaces, in: Proceedings of the Symposium on Ambient Intelligence, Vol. 3295 of Lecture Notes in Computer Science, Springer, 2004, pp. 291–302.

[26] J. Coutaz, L. Balme, C. Lachenal, N. Barralon, Software infrastructure for distributed migratable user interfaces, in: Proceedings of the UbiHCISys Workshop on UbiComp, 2003.

[27] A. Demeure, J.-S. Sottet, G. Calvary, J. Coutaz, V. Ganneau, J. Vanderdonckt, The 4C reference model for distributed user interfaces, in: Proceedings of the International Conference on Autonomic and Autonomous Systems, 2008, pp. 61–69.

[28] T. C. N. Graham, T. Urnes, R. Nejabi, Efficient distributed implementation of semi-replicated synchronous groupware, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 1996, pp. 1–10.

[29] C. N. Klokmose, M. Beaudouin-Lafon, VIGO: instrumental interaction in multi-surface environments, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, 2009, pp. 869–878.

[30] P. Tandler, Software infrastructure for ubiquitous computing environments: Supporting synchronous collaboration with heterogeneous devices, Lecture Notes in Computer Science 2201 (2001) 96–115.

[31] J. P. Sousa, D. Garlan, Aura: an architectural framework for user mobility in ubiquitous computing environments, in: Proceedings of the IEEE/IFIP Conference on Software Architecture, 2002, pp. 29–43.

[32] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, K. Nahrstedt, Gaia: A middleware infrastructure for active spaces, IEEE Pervasive Computing 1 (4) (2002) 74–83.

[33] M. Modahl, B. Agarwalla, G. D. Abowd, U. Ramachandran, T. S. Saponas, Toward a standard ubiquitous computing framework, in: Proceedings of the Workshop on Middleware for Pervasive and Ad-hoc Computing, 2004, pp. 135–139.

[34] M. Modahl, I. Bagrak, M. Wolenetz, P. W. Hutto, U. Ramachandran, MediaBroker: An architecture for pervasive computing, in: Proceedings of the IEEE Conference on Pervasive Computing, 2004, pp. 253–262.

[35] J. Melchior, D. Grolaux, J. Vanderdonckt, P. V. Roy, A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications, in: Proceedings of the ACM Symposium on Engineering Interactive Computing System, 2009, pp. 69–78.

[36] K. Kim, W. Javed, C. Williams, N. Elmqvist, P. Irani, Hugin: A framework for awareness and coordination in mixed-presence collaborative information visualization, in: Proceedings of the ACM Conference on Interactive Tabletops and Surfaces, 2010, pp. 231–240.

[37] J. T. Biehl, W. T. Baker, B. P. Bailey, D. S. Tan, K. M. Inkpen, M. Czerwinski, IMPROMPTU: a new interaction framework for supporting collaboration in multiple display environments and its field evaluation for co-located software development, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, 2008, pp. 939–948.

[38] D. S. Tan, B. Meyers, M. Czerwinski, WinCuts: manipulating arbitrary window regions for more effective use of screen space, in: Extended Abstracts of the ACM Conference on Human Factors in Computing Systems, 2004, pp. 1525–1528.

[39] J. R. Wallace, R. L. Mandryk, K. M. Inkpen, Comparing content and input redirection in MDEs, in: Proceedings of the ACM Conference on Computer Supported Cooperative Work, 2008, pp. 157–166.

[40] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, P. Hanrahan, WireGL: A scalable graphics system for clusters, in: Computer Graphics (Proceedings of ACM SIGGRAPH), 2001, pp. 129–140.

[41] OpenSG, `http://www.opensg.org/`.

[42] B. B. Bederson, J. Grosjean, J. Meyer, Toolkit design for interactive structured graphics, IEEE Transactions on Software Engineering 30 (8) (2004) 535–546.

[43] B. Johanson, A. Fox, The event heap: A coordination infrastructure for interactive workspaces, in: Proceedings of the IEEE Workshop on Mobile Computer Systems and Applications, 2002, pp. 83–93.

[44] B. Johanson, A. Fox, Extending tuplespaces for coordination in interactive workspaces, The Journal of Systems and Software 69 (3) (2004) 243–266.

[45] D. Gelernter, Generative communication in Linda, ACM Transactions on Programming Languages and Systems 7 (1) (1985) 80–112.

[46] T. J. Lehman, S. W. McLaughry, P. Wyckoff, T spaces: The next wave, in: Proceedings of the Hawaii International Conference on System Sciences, 1999.

[47] JGroups, `http://www.jgroups.org/`.

[48] The Internet Engineering Task Force (IETF), RTP: A transport protocol for real-time applications, Tech. Rep. RFC 1889, Audio-Video Transport Working Group (Jan. 1996).

[49] A. Datta, S. Buchegger, L.-H. Vu, T. Strufe, K. Rzadca, Decentralized online social networks, in: Handbook of Social Network Technologies and Applications, Springer, 2010, pp. 349–378.

[50] M. Lorenz, G. Brunnett, M. Heinz, Driving tiled displays with an extended Chromium system based on stream cached multicast communication, Parallel Computing 33 (6) (2007) 438–466.

[51] E. Pietriga, S. Huot, M. Nancel, R. Primet, Rapid development of user interfaces on cluster-driven wall displays with jBricks, in: Proceedings of the ACM Symposium on Engineering Interactive Computing Systems, 2011, pp. 185–190.

[52] Y. Shinjo, F. Guo, N. Kaneko, T. Matsuyama, T. Taniuchi, A. Sato, A distributed web browser as a platform for running collaborative applications, in: Proceedings of the IEEE Conference on Collaborative Computing, 2011, pp. 278–286.

[53] C. von der Weth, A. Datta, COBS: Realizing decentralized infrastructure for collaborative browsing and search, in: Proceedings of the IEEE Conference on Advanced Information Networking and Applications, 2011, pp. 617–624.

[54] N. Elmqvist, Munin: A peer-to-peer middleware for ubiquitous visualization spaces, in: Proceedings of the ACM Workshop on Distributed User Interfaces, 2011.

[55] K. Hinckley, J. Pierce, M. Sinclair, E. Horvitz, Sensing techniques for mobile interaction, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2000, pp. 91–100.

[56] K.-P. Yee, Peephole displays: pen interaction on spatially aware handheld computers, in: Proceedings of the ACM Conference on Human Factors in Computing Systems, 2003, pp. 1–8.

[57] E. Eriksson, T. R. Hansen, A. Lykke-Olesen, Movement-based interaction in camera spaces: a conceptual framework, Personal and Ubiquitous Computing 11 (8) (2007) 621–632.

[58] S. Gustafson, D. Bierwirth, P. Baudisch, Imaginary interfaces: spatial interaction with empty hands and without visual feedback, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2010, pp. 3–12.

[59] S. Gustafson, C. Holz, P. Baudisch, Imaginary phone: learning imaginary interfaces by transferring spatial memory from a familiar device, in: Proceedings of the ACM Symposium on User Interface Software and Technology, 2011, pp. 283–292.

[60] R. Bandelloni, F. Paterno, Flexible interface migration, in: Proceedings of the ACM Conference on Intelligent User Interfaces, 2004, pp. 148–155.