Int. J. Human-Computer Studies 72 (2014) 100–110

Contents lists available at ScienceDirect

Int. J. Human-Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

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, 465 Northwestern Avenue, West Lafayette, IN 47907-2035, USA

ARTICLE INFO

Article history:
Received 27 August 2012
Received in revised form
6 August 2013
Accepted 28 August 2013
Communicated by E. Motta
Available online 7 September 2013

Keywords:
Distributed user interfaces
Case studies
Design principles
Lessons learned
Implementation
DUI toolkits

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.

1. Introduction

Parallelism—both on the local computer using multiple cores (Blake et al., 2009), as well as distributed across multiple virtual machines in the cloud (Grossman, 2009)—has become the de facto solution to today's computational problems when Moore's law no longer is 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 competing 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 (Elmqvist, 2011a; Gallud et al., 2011). 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. Gjerlufsen et al., 2011; Marquardt et al., 2011; Jetter et al., 2011), 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

* Corresponding author. Tel.: +1 765 494 0364; fax: +1 765 494 6951.
E-mail addresses: fisher55@purdue.edu (E.R. Fisher), sbadam@purdue.edu (S.K. Badam), elm@purdue.edu, niklas.elmqvist@gmail.com (N. Elmqvist).

1071-5819/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.ijhcs.2013.08.011

(P2P) network architecture. Other network architectures have also been successfully applied to DUI development, particularly based on a client/server model (e.g. Bharat and Cardelli, 1995; Equalizer; Humphreys et al., 2002; Jetter et al., 2012; Nacenta et al., 2007). 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 (Gallud et al., 2011; Weiser, 1991). 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 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 transferring 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.

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 (McGrath et al., 2012) 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 (Fig. 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 (McGrath et al., 2012), our focus in the present paper is on the software engineering aspects of the BEMViewer, which have not been previously published.

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 (Fig. 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 (Fig. 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 (Elmqvist, 2011a). 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.

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