Architectures for Rich Internet Real-Time Games

Matthias Häsel
University of Duisburg-Essen, Germany

INTRODUCTION

Many researchers regard multiplayer online games as the future of the interactive entertainment industry (Brun, Safaei, & Boustead, 2006; El Rhalibi & Merabti, 2005; Sharp & Rowe, 2006). In particular, due to advances in game design and the availability of broadband Internet access to the end-user, multiplayer online games with real-time interaction have come into wide use (Aggarwal, Banavar, Mukherjee, & Rangarajan, 2005; Claypool & Claypool, 2006; Yasui, Yutaka, & Ikedo, 2005). The majority of these games are made up by classic software titles that need to be installed on the players' machines (El Rhalibi & Merabti, 2005). Browser-based multiplayer games, on the contrary, can be run instantly from a Web site, but have, due to technical limitations, long been round-based, strategy-focused games. However, with the ongoing evolvement of Rich Internet Application (RIA) technology (Allaire, 2002) such as Adobe Flash and Java, browser-based online game development has reached a point where also *real-time* games can be produced and distributed to a large audience quickly and easily. Browser-based games can be utilized in conjunction with e-business offers in a very simple way and hold a number of exciting possibilities for new online business models, new markets, and new growth (Kollmann & Häsel, 2006; Sharp & Rowe, 2006). However, as the browser is a very different operating environment and interactive experience from that of classical game software, browser-based multiplayer real-time games involve gaming architectures that are distinct from their classical counterparts. A major challenge when designing and implementing such architectures is that multiplayer online games are highly vulnerable to propagation delays resulting from redundant communication, bottlenecks, single points of failure and poor reactivity to changing network conditions (Ramakrishna, Robinson, Eustice, & Reiher, 2006). As latency from input of information to its output determines gameplay and fairness (Brun et al., 2006), the game architecture has to be designed in a way that it mitigates latency effects and meets the expectations of the players (Claypool & Claypool, 2006). Elaborating on the example of an online tabletop soccer game with two remote players, this article discusses two architectural models that can be applied to implement browser-based multiplayer real-time games using RIA technology.

BACKGROUND

Online games that give the player the ability to compete against other players over a network emerged strongly in the middle of the last decade. Traditionally, multiplayer online games have been implemented using client-server architectures (GauthierDickey, Zappala, Lo, & Marr, 2004). Thereby, a copy of the software is installed on each player's machine, which connects directly to a central authoritative server designed to handle game logic (Claypool & Claypool, 2006; El Rhalibi & Merabti, 2005; Guo, Mukherjee, Kangarajan, & Paul, 2003). The server deals out information individually to each client as it is requested and keeps all the players up to date with the current state of the game (El Rhalibi & Merabti, 2005). Whenever a client performs an action (such as firing a gun or kicking a ball), the data is sent to the server, which calculates the effects of that action and sends the updated game state to the clients (El Rhalibi & Merabti, 2005). Client-server architectures have the advantage that a *single* decision point orders the clients' actions, resolves conflicts between these actions and holds the global game state (GauthierDickey et al., 2004). Unfortunately, they also have several disadvantages. The most obvious one is that client-server architectures introduce delay because messages between players are always forwarded through the server (GauthierDickey et al., 2004). This adds additional latency over the minimum cost of sending commands directly to other clients (Cronin, Kurc, Filstrup, & Jamin, 2004). Moreover, traffic and CPU load at the server increases with the number of players, creating localized congestion and limiting the architecture by the computational power of the server (GauthierDickey et al., 2004).

To address the problems associated with client-server architectures, many authors and game designers have developed fully distributed peer-to-peer architectures for multiplayer games (El Rhalibi & Merabti, 2005; GauthierDickey et al., 2004). In a peer-to-peer architecture, players send their actions to each other and react on the received action (Guo et al., 2003). Each peer acts as a decision point that has exactly the same responsibilities as every other peer (El Rhalibi & Merabti, 2005). Peer-to-peer architectures have a lot of advantages. Firstly, there is neither a single point of failure nor an expensive server infrastructure. Moreover, the amount of bandwidth required is reduced dramatically as there is a direct communication between two peers. This also reduces latency on the network, as the bottleneck caused by a server is eliminated (El Rhalibi & Merabti, 2005). However, unlike games using a client-server architecture, where there is a single authoritative copy of the game state kept at a central server, peer-to-peer architectures require an up-to-date copy of the entire game state to be kept at each peer. Consequently, these architectures require some form of distributed agreement protocols between the peers that prevents the peers' game states from diverging over time and becoming inconsistent (Cronin et al., 2004; Guo et al., 2003).

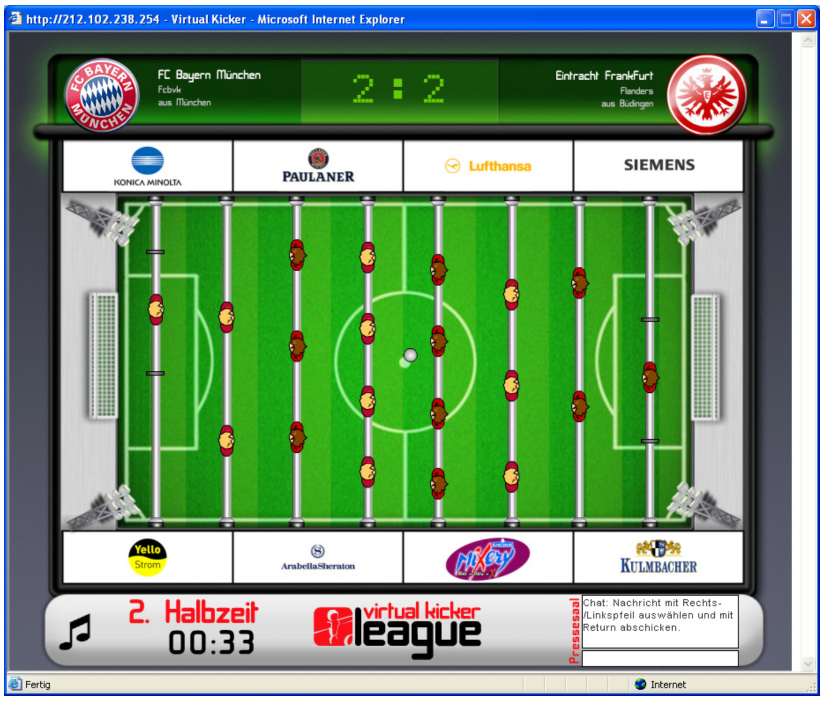
Although most online games have low bit-rate requirements sending frequent but small packets typically well within the capacity of broadband connections (Brun et al., 2006), deploying these applications over a large-scale infrastructure presents a significant technological challenge. In both client-server and peer-to-peer architectures, an increase in the geographical distances among participating clients or servers results in an unavoidable end-to-end delay that may render the game "unresponsive and sluggish even when abundant processing and network resources are available" (Brun et al., 2006, p. 46). Moreover, differences in game responsiveness to user input may give some players an unfair advantage (Brun et al., 2006; Aggarwal et al., 2005). If the latency between a client and the server (or between two peers) is large enough, the responsiveness of the game to a player's action decreases, and the player's performance is likely to degrade (Claypool & Claypool, 2006). A game can be regarded as playable if the players find its performance acceptable in terms of the perceptual effect of its inevitable inconsistencies, whereas fairness is concerned with relative playability among all players (Brun et al., 2006).

ARCHITECTURE DESIGN AND IMPLEMENTATION

After reviewing existing literature on architectural concepts and issues of multiplayer online games, the remainder of this article will examine to what extent these concepts and issues hold for *browser-based* multiplayer real-time games. As a matter of simplification, the focus will be on games featuring concurrent, pairwise real-time interactions between two remote players, such as it the case for the online tabletop soccer game that is depicted in Figure 1.

For the player-side RIA, Adobe Flash is assumed to be the best-suited technology since Flash, due to its common runtime environment across operating systems, browsers and chip architectures, enables easy deployment on multiple platforms and devices (Allaire, 2002). In contrast to Java as a possible alternative, Flash has a much higher diffusion rate and can be updated by the player in a very easy way. Moreover, Flash is very efficient in rendering

Figure 1. An online tabletop soccer game with two remote players



Chat: Nachricht mit Rechts-/Linkspfeil auswählen und mit Return abschicken.