

Open-Source Game Development with the Multi-User Publishing Environment (MUPE) Application Platform

Riku Suomela¹, Eero Räsänen¹, Ari Koivisto² and Jouka Mattila¹,

¹ Nokia Research Center, P.O. Box 100,
33721 Tampere, Finland
{riku.suomela, eero.k.rasanen, jouka.mattila}@nokia.com
² Tampere University of Technology, P.O. Box 527,
33101 Tampere, Finland
borg@iki.fi

Abstract. The Multi-User Application Platform (MUPE) is a platform for rapid development of mobile multi-user context-aware applications. MUPE server implements a persistent user-authenticated service that can be customized into a game server. The game logic is written to the MUPE server and the end-users download the game User Interface (UI) to their terminals. This paper studies how MUPE can be used to create mobile multi-player games. This paper analyzes the important aspects of MUPE in game development and the different parts involved in developing games with MUPE. Two games made with MUPE are introduced and analyzed. The games presented in this paper and the MUPE system are available at the MUPE website <http://www.mupe.net> under the Nokia open source license version 1.0a.

1 Introduction

Mobile phones and devices are carried by millions of users worldwide. The devices are connected, thus enabling the construction of multi-player games that can be played by millions of mobile users. The development of multi-player games is a time and resource-consuming task as a multi-player game requires a solid software platform before the actual game development can start. Multi-User Publishing Environment (MUPE) is an open-source platform for creating mobile multi-user context-aware applications [1], such as mobile games. MUPE aims to lower the threshold of creating mobile multi-player games with a solid application platform that enables the application developers to directly concentrate on the game development.

The MUPE application platform has a client-server architecture, where the end-users use a MIDP 1.0 midlet to connect their mobile devices to the MUPE server in the Internet. A single MUPE client is used in all MUPE applications to enable easy connection to different MUPE applications and services. Any external information source in the Internet can be added to MUPE with the context components. The MUPE connection middleware connects the different parts of MUPE. The MUPE structure is seen in Fig. 1.

MUPE applications are written to the MUPE server, which is the only component requiring programming in a basic application. The client User Interface (UI) is created with client UI scripts that the client downloads from the server. With the first connection, the client creates a persistent user account to the server and the client also stores the service address, username and password for enabling easy subsequent connection. Each application that the user connects to is displayed on the MUPE client services list.

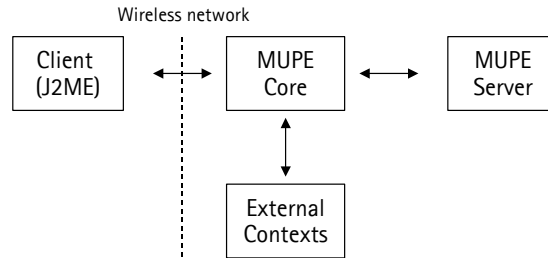


Fig. 1. The MUPE application platform structure

All parts of MUPE are available under the Nokia open source license version 1.0a [2]. Open sourcing the platform enables application developers to modify the system for their specific needs. Each game is different and has different requirements that can be well achieved with an open-source platform.

This paper studies how MUPE can be customized for mobile multi-player games. The application developers are free to modify MUPE and create their own games and services. The paper does not focus on game design, but rather looks into properties of MUPE that affect game design.

The paper is organized as follows. First, mobile and multi-player games as well as other technologies that influenced the development of MUPE are studied. After that the MUPE system and the essential parts to gaming are explained followed by two example games developed with MUPE. Finally, the work is concluded and future work is discussed.

2 Mobile and Multi-Player Gaming

MUPE Platform is designed to enable easy creation of mobile multi-user context-aware applications. There are several systems that inspired and influenced the MUPE application platform design. The most important influences were the MUDs (Multi-User Dungeons) and MOOs (MUD Object-Oriented) [3] that were an early form of computer supported multi-player gaming. The first MUDs were developed already in the 1970s, being virtual worlds where people could interact with other users of the system. The MUDs have evolved over the years to support the new technologies in the Internet and today there are several hugely popular Massively Multiplayer Online Games (MMOG), such as Ultima Online [4] and PlanetSide [5]. The first MMOGs are making their way to mobile phones as well, as seen by games such as Tibia Micro

Edition (TibiaME) [6]. Palm [7] gives a good overview of issues regarding the development of Mobile Massively Multiplayer Online Games (3MOG).

Mobile devices, however, are very different in nature compared to the desktop computers. The mobile devices are used in varying situations as the users move in the real world. The device's ability to react and adapt to changes in the users environment is called context-awareness. A good definition of context awareness is found in [8], which states that context is information that describes the situation of an entity. Context-awareness allows the environment to have an affect on the gameplay.

One of the oldest forms of context-aware gaming is the treasure hunt games played with a GPS receiver [9]. Several other context-aware games have also been built, such as the Pirates! [10], ARQuake [11], BotFighters [12] and Can You See Me Now? [13]. These games add new features to the games that truly take advantage of the mobility and the environment where the game is played. As the device is carried in the real world, the real world can and should influence the gameplay. A good study on how the real world can influence the gameplay can be found in [14].

MUPE is a platform for quick and easy development of new applications. MUPE contains the tools and ready-made technologies for creating new applications and games. The systems described here are good examples of how the games have evolved over time. The evolution will continue and MUPE aims to speed up the process of integrating new technologies to mobile gaming.

3 MUPE Application Platform for Mobile Gaming

MUPE application platform was designed to support different kinds of applications. This chapter explores the key technologies of MUPE that affect mobile game development. The key design issue when designing games with MUPE is the optimization of the client server communications.

Games, as all MUPE applications, are written to the MUPE server. The server is the main container for the application and it is the only component requiring programming when creating new games. The end-user client is a J2ME MIDlet that connects to the server. The client uses a custom script language that describes the UI and client side functionality. Any context source in the Internet can be connected to the system. The communication core of MUPE does not need any modification for a typical game application.

This chapter looks at how the MUPE application platform can be customized for mobile multi-player games. First, the client and its technology is discussed so we have a firm understanding on what kind of content can be shown to the end-users. After that, the server is analyzed and finally the context-awareness in the system is explained.

3.1 MUPE client for gaming

MUPE client is a MIDlet running on a mobile device such as a mobile phone. A single MUPE client is used in all different MUPE applications and the different UIs

and functionality is achieved with the MUPE client script language. The client is based on MIDP 1.0 specifications [15] and it can be optimized in many different areas making it suitable for networked mobile applications. The main optimizations are done at the network level and with the client-side script language

In the network level, the communication is optimized for GPRS level data transfers, which mean high latency, and mediocre data transfer capabilities. MIDP 1.0 implementation uses only HTTP protocol, which restricts the choice of data exchange between the client and server. The client communicates with server by making requests and on each request the HTTP connection must be opened, which results in a high latency. Often the connection to the server takes longer than the actual data transfer. The HTTP connection sets restrictions on game developers, but with proper game design these should not matter too much. A good overview on mobile multiplayer game design can be found in [16]. The document also points out the latency problem and how the developers need to prepare for it.

The script language implements two different UI styles and some client side functionality enabling the client to be somewhat standalone. The two UIs supported by the client are canvas UI, which is a Graphical UI (GUI), and form UI, which is a text based UI. The two different UIs are both suitable for gaming purposes, but they are very different in nature. The client can contain several UIs at the same time and only one of them is active at a time.

The client script language uses hooks to catch different events in the UI, for example selecting an object or pressing a button. The hooks enable client-side functionality that work without a need for network communications. For example, the press of a button can move or hide objects on screen. The script language also defines attributes, timers, loops and a set of conditional statements that make the client-side functionality richer and more independent from the server.

The main point to remember when designing games with MUPE is that the transactions with the server create the most downtime and should be minimized. It is better to load the whole set of (or a certain subset of) user-interfaces into the client at one time than loading them on-demand all the time, since one can easily activate and disable UIs and UI widgets in the client-side when needed. This also applies to graphics, and all the required images should be loaded on startup, even though they are not all shown in the beginning. Naturally the client device's memory capacity sets some limitations on the size of these files. During the game, these pre-loaded images should be used, creating new image objects on the screen when necessary, and deleting them when they're not needed anymore. In some cases, rather than deleting, it's better just to hide the image objects, and show them again when they're needed. This removes the small delay in creating and deleting UI objects on the client side.

One can argue that a script language is not an optimal solution for the client side. However, the script language was implemented due to the limitations of MIDP 1.0 specifications, which do not allow the MIDlet to download Java classes over the network. This would be beneficial in many cases, as each game could use a dedicated client. One of the main goals of MUPE was to develop a single client that can be used in many applications. This eliminates the need to install a new client for every application.

MIDP was chosen for the client implementation due to its popularity. There are millions of client devices capable of running MIDP 1.0 MIDlets providing a wide

range of end-user platforms that can run MUPE. MIDP is also constantly evolving, as many devices support MIDP 2.0.

For detailed information on the client script language, refer to the MUPE website <http://www.mupe.net>.

3.2 MUPE server for gaming

MUPE server contains all of the game functionality and logic. The server knows the current state of the game and the connected players. The clients make requests and the server replies to the requests with a valid UI script. The UI script either generates a new UI or updates the existing UI. The reload of the whole UI should be avoided in games using a GUI whenever possible and only the changes should be updated. The server needs to know the state of each player in the system and be able to update the necessary changes to all.

The basic MUPE server structure [17] contains a *World* class for storing the content inside the application and four classes representing the content: *User*, *Item*, *Service* and *Room*. The *User* class represents a single connected end-user in the system, the *Room* a location in the game world, *Services* provide add-on features for other objects, and an *Item* is any other data inside the application. The server structure can be easily mapped to common game elements: *User* for each player, *Room* for each game location, and *Item* for each object used by players or contained in the game world. *Services* provide add-on features to other objects, such as group forming. The game world can be in a single *Room*, or the game world is divided into several areas, each represented by *Rooms*. The *Item* class should be derived to add the necessary functions and items needed in the game. Different type of players in the game should be derived from the *User* class for the each different type of character needed in the game. The basic MUPE server structure is seen in Fig. 2.

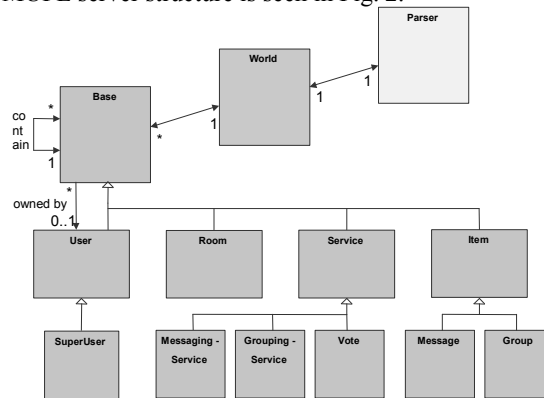


Fig. 2. The basic MUPE server structure

As MUPE clients only support HTTP as the connection protocol, server cannot push any data to clients in order to update the UI. Instead, clients must poll the server constantly to fetch the latest updates. To make things easier, MUPE server has built-in commands for “pushing” data to users. Since current version cannot actually push the data to Clients, it is instead queued separately for each user, and returned to the Client with the next request the Client makes. Another built-in command also enables pushing data to every user within a certain *Room* in the game world to make updating the views even easier. The push times could vary for different users, depending on many things such as the network load. As the latency of each individual user can vary, the users push data queue can contain varying amount of information for various users.

The basic MUPE server contains several useful features that can be used in almost any game. As the user connects to the MUPE system, she/he creates an avatar into the MUPE server. This can be used as the game character creation service. The server also remembers the user state between the connections, and the MUPE communications middleware takes care of typical game communication needs. These features make MUPE an appealing platform for game creation. The virtual world and the user accounts can be easily modified to fit many kinds of multi-player games.

3.3 Context-awareness

The context-awareness in MUPE is built into two separate parts. First, the connected user clients can send their context information to the server, as they are constantly connected to the service. Second, any information in the Internet can be sent to the MUPE application by writing a context producer that formats context information according to the Context Exchange Protocol (CEP) [18].

The connected user clients vary in their capabilities, so it is difficult to be sure that all the users have certain context capabilities present in their devices. If the game uses for example the location of the user, the capabilities of the different can vary a lot; some devices are not able to determine their location, while others have a varying accuracy for it.

The context producers on the Internet are not dependent on the client capabilities. They feed information to the MUPE applications no matter who is connected and also they can be totally controlled by the system administrators thus providing reliable information. External context producers can characterize the situation of any entity in the application, not necessarily that of an end-user. A single context producer can feed information to many MUPE services and when a producer is setup, other applications can also later connect to it.

Client-side context-awareness allows a lot more possibilities for the game developers. Each player can have personal context-information that describes his or her current state, which will be different for every user. At the moment though, the context producers are the only form of contexts that are supported by the MUPE application platform. There are several useful APIs for the J2ME platform that can be used, but at the time of writing this paper there were no devices available for us. The client script language will be updated with the context-aware features once they have been properly tested. At least the Bluetooth API (JSR-82) and Location API (JSR-

179) for J2ME should be useful when determining context information on the client side. For more information on upcoming APIs for the J2ME platform, please refer to [15].

4 Games with MUPE Application Platform

We have developed two games with the MUPE application platform that explore the strengths and weaknesses of MUPE. They are both modified from the basic MUPE server found in the MUPE website. Both of these games are also available as open source software and they can be modified and extended by others.

The basic MUPE server does three things that are useful for the multi-player games. First, the basic MUPE framework takes care of all the communications. Second, the server creates a persistent user account for each user. Third, MUPE client and server remember these connections and allow the users to reconnect to the services easily, enabling a persistent game. These things enable the game developers to concentrate on the most important task, the game development. We hope that MUPE enables new mobile games to be made with less effort and resources.

The games have context-aware information already built into them and simulators for generating the data. However, currently the games presented here do not use this data for anything. If a proper connection with real world data to these games can be found, the context awareness is easy to add to these games.

4.1 MUPE Dungeon

MUPE Dungeon is a game of dungeon exploring, heavily influenced by the MUDs. The users take the role of one of four character types: Fighter, Gnome, Wizard or Thief and together the players explore the dungeon and fight horrendous monsters. The game screen seen by the players is composed of 9x9 tiles and usually less than half of the screen is occupied with the structures. Players, monsters and empty tiles take up the remaining space. Each game screen is a single *Room* in the MUPE world. The users move between the *Rooms* in the game world and encounter other users and monsters in them. There can be one or more players and the number of players is not restricted by the game, but the limited screen space makes it difficult to have a very large number of players at the same screen. The system will not support an endless amount of players as there will be limiting factors, such as cellular base station capacity and server hardware.

Three classes are created for the dungeon game: player, mapsquare and monster. Every player is a subclass of *User*, every 9x9 square view of the dungeon is a subclass of a *Room*, and all the monsters within the rooms are derived from *Item*. If the game was extended to contain weapons and other items, these could be derived from *Item* class and be picked up and carried by Users.

The game aims to showcase how MUPE can be used to build semi real-time multi-player games that have a varying number of players. Each player has a fixed time to make a move, which is set to five seconds. After that, each player updates his/her

move to the server and the server responds with the new state of the UI. All graphics are downloaded at the connection time so graphics download does not slow down the gameplay. The communication between the client and server is done with HTTP, which usually takes a few seconds to get the response. During this time, the client animates the player's movement creating a feeling that something is constantly happening.

The user interface is shown in Fig. 3 and Fig. 4. The first screenshot in the Fig. 3 shows the login process where the user sets the password for connecting. The following image shows the user selecting the character for the game. The third image shows the four different player types and the three different monster types. There is also an animation shown on the screen that is marked with the star. The character next to the star has attacked a monster and while the client waits for the new updated information from the server, the animation is shown.

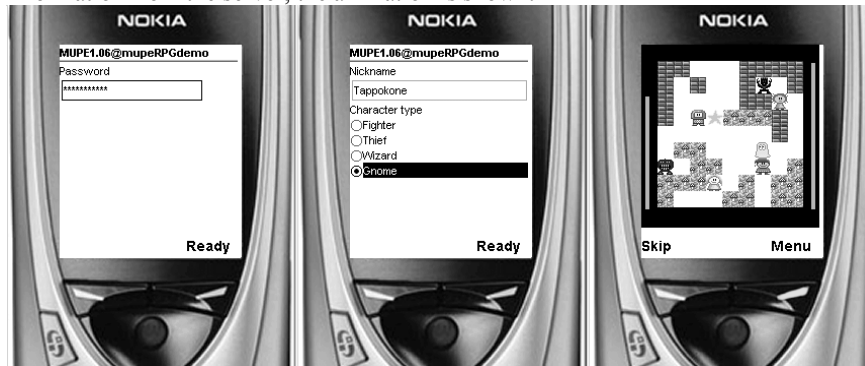


Fig. 3. Three screenshots from the MUPE Dungeon Game. All screenshots in this document are screenshots from the Wireless Toolkit 1.0.4_01 Emulator. The screen in mobile devices may vary

The animations are achieved by hooking the button presses of the device on the client side. As the user presses a button, the client knows what to do. If a monster is on the direction where the user is going to, the monster is attacked. If there is an empty space, the user is moved. This information is needed before the server data exchange to be able to create the animations while the user waits for the update.

The Fig. 4 shows two more images from the game. First, the character can see his or her stats by reverting to the character abilities screen. This screen and the game area are both contained in the memory of the client and no server connections are necessary when changing between the views. The user can also choose to say something to other players and the second image in Fig. 4 shows how the users see the comments of other players.

This game has been played several times with a varying group of players and the response has been the same. The user experience is heavily dependent on the latency of the connection, which varies a lot depending on many factors. As the user makes a move, the following move cannot be made before the server responds with the new state of the game. Whenever the latency is low enough, the update can be done while the animation is running on screen. If the latency is much higher than the animation time, the game is not very appealing.

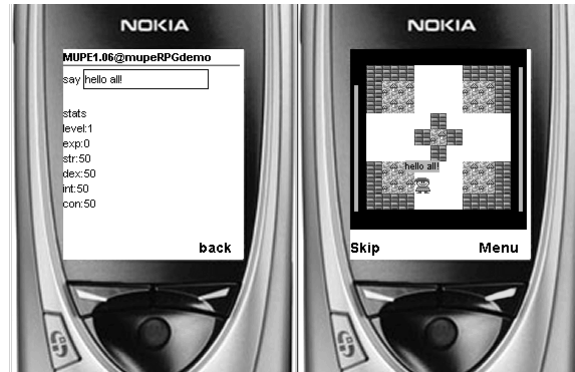


Fig. 4. Two more screenshots from the MUPE Dungeon game

4.2 First Strike

The second game made with MUPE is a turn-based strategy game for exactly four players and the game applies simultaneous turn taking. The players connect to the MUPE server and are placed in the game waiting room. The players select a color that represents them in the game and after enough players are connected, the game starts. When the four players start the game, the waiting room becomes empty and the next four players can start a new game.

First Strike contains four countries that are fighting for victory. Each player owns two cities and each player can choose one action from three options in a round. The choices are: 1. Attack another player's city. 2. Place shields on top of one city. 3. Hire the Nuclear Waste Army to visit a city (the result of the visit is unknown). Again, the users have a certain amount of time to select their actions after which the users data is sent to the server. During the update the user sees a notification: "waiting for other players". After the update is completed, the new game state is updated on screen, the actions are animated and the next round begins.

The flow of the game can be seen in Figs 5 and 6. The first screenshot in Fig. 5 shows the character creation screen. After this, the user chooses a color and waits for other players to enter the game. The third image shows the actual gameplay, where the user selects a target for the military action.

The Fig. 6 shows more screenshots from the game. The first screenshot shows the user selecting what action is performed on the selected city, followed by the UI informing the user about the server update. Finally, an animation shows the actions taken this round.

The game server has been played several times and this game is not as dependent on the latency of the network as the MUPE Dungeon game. The user does many things in a turn, like assessing the status of the game and deciding what the next action will be. This also goes on during the update of the game state and thus the game tempo is better for this game.

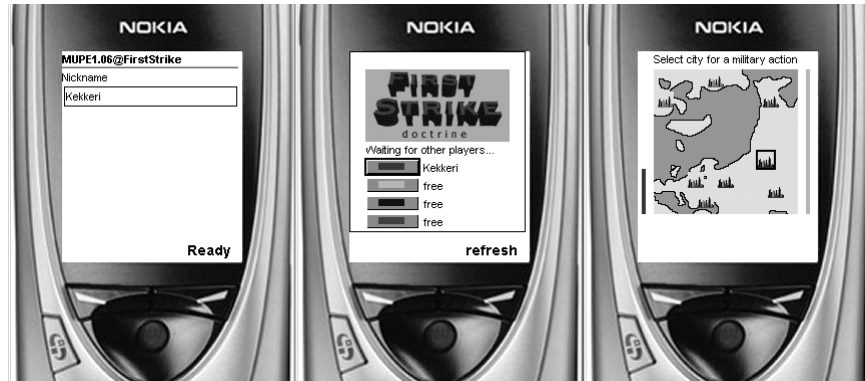


Fig. 5. Three screenshots of the First Strike game



Fig. 6. Three more screenshots of the FirstStrike game

The FirstStrike has a different server structure compared to the MUPEDungeon. The players are again *Users*, but this time the whole game is a single *Room*, and there are no *Items*. Even the *Users* are unchanged from the original MUPE release, making the whole First Strike game to be fit into only one *Room* subclass. Games such as FirstStrike can be moved to any MUPE application, where it is a special *Room* offering a gaming service.

One idea of this game was to create a mini-game server, where several multiplayer games can be selected. At the moment the FirstStrike is the only game available but new games can be added in a similar way, that is, deriving from the *Room* object.

4.3 Analysis

This chapter presented two different games made with the MUPE platform. The games were designed differently, but both fitted well to the MUPE application platform.

The MUPE server offers a flexible structure that can be exploited in many ways. The first game, MUPE Dungeon, creates a vast virtual world into the MUPE server where the users move between the game area rooms and explore the game world. The second game, First Strike, fits into a single room making it transferable to any MUPE server.

It seems simultaneous turn taking is a good strategy when designing games with the MUPE platform. Simultaneous turn taking is scalable, as the time for a round does not increase with the amount of players.

Both of the games could use many features of the basic MUPE server, to speed up the development process. The user login is unchanged and the character creation only needs a little modification. The communications and client do not require any programming.

5 Conclusions

This paper presented how MUPE – the Multi-User Publishing Environment can be used to create mobile multi-player games. The platform enables rapid creation of multi-player games by providing ready-made architecture that connects the players to a server and creates a persistent player account for the end-users. The platform is available as open-source software allowing application developers to create their own mobile games with maximum flexibility.

The MUPE architecture fits well for games. The basic platform implements all connections and the creation of a persistent user account that are essential parts in multi-player gaming. The UI of the client can be completely customized and functionality can be uploaded to the client allowing the end-users to interact with the client device without a constant need for updates to the game server.

Two mobile multi-player games made with MUPE were presented to highlight the various aspects of the MUPE platform. The two games had a very different server structure highlighting how different games can be built with MUPE. The other game used several Room objects where the user can move to create a large game world, whereas the other game uses a single Room object for the game allowing easy addition of multiple games into the same server. Also, the game can be easily moved to other MUPE applications.

The MUPE game design was mostly restricted by the HTTP transport protocol used by the client. This causes high latencies in the data exchange between the client and the server. With proper design, for example applying simultaneous turn taking, such effects can be reduced.

6 Future Work

All parts of MUPE platform are under constant development. There are constantly new features and APIs available for the J2ME platform that will be integrated to the client UI scripts. The MIDP 2.0 version of the client will support other connection

methods than HTTP, among other features. Java APIs for context-awareness, such as the Java location API (JSR-82) and the Bluetooth API (JSR-179) will be implemented in the future.

These and new games will be further developed in the Nomadic Media ITEA (Information Technology for European Advancement) project. We aim to create games that support public screens in public spaces.

Acknowledgements

This work is funded by the Nomadic Media ITEA project.

References

1. Multi-User Publishing Environment (MUPE) Application Platform. Available online at the MUPE website. <http://www.mupe.net> (2003)
2. Nokia Open Source License Version 1.0a (NOKOS License). Available online at <http://www.opensource.org/licenses/nokia.php>
3. Multiple User Dimension/Dungeon/Domain (MUD). Available online at <http://www.moo.mud.org/>
4. ORIGIN Systems Inc: Ultima Online. Electronic Arts (1997)
5. Verant Interactive: PlanetSide. Sony Online Entertainment (2003)
6. CipSoft: Tibia Micro Edition (TibiaME). T-Mobile. (2003)
7. Palm, T.: The Birth Of The Mobile MMOG. Available online at http://www.gamasutra.com/resource_guide/20030916/palm_01.shtml (2003)
8. Dey, A.K.: Providing Architectural Support for Building Context-Aware Applications, Ph.D. thesis. College of Computing, Georgia Institute of Technology, (2000). Available online at <http://www.cc.gatech.edu/fce/ctk/pubs/dey-thesis.pdf>
9. Geocaching. <http://www.geocaching.com/>
10. Björk, S., Falk, J., Hansson, R., Ljungstrand, P.: Pirates! - Using the Physical World as a Game Board. Proceedings of the Human-Computer Interaction INTERACT'01 (2001) 423-430
11. Thomas, B., Close, B., Donoghue, J., Squires, J., De Bondi, P., Morris, M., Piekarski, W.: ARQuake: an outdoor/indoor augmented reality first person application. Proceedings of the Fourth International Symposium on Wearable Computers. (2000) 139-146
12. It's Alive: BotFighters. It's Alive (2001)
13. Can You See Me now? <http://www.canyouseemenow.co.uk/>
14. Sotamaa, O. All The World's A Botfighter Stage: Notes on Location-based Multi-player Gaming. Proceedings of the Computer Games and Digital Cultures Conference, Tampere University Press (2002) 35-44
15. Java 2 Platform, Micro Edition (J2ME). <http://java.sun.com/j2me/>
16. Overview of Multiplayer Mobile Game Design. Available online at: <http://www.forum.nokia.com/main.html> (2003)
17. Koivisto, A.: Multi-User Publishing Environment Server. M. Sc. Thesis, Tampere University of Technology. Available online at: <http://www.mupe.net/> (2003)
18. Context Exchange Protocol. Available online at: <http://www.mupe.net/> (2003)