The 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07)

FUELING GAME DEVELOPMENT IN MOBILE P2P ENVIRONMENTS

Aris Kosmopoulos, Ifigeneia Karamichali, Vasileios P. Kemerlis, George C. Polyzos
Mobile Multimedia Laboratory, Department of Computer Science
Athens University of Economics and Business
GR-113 62, Athens, Greece

ABSTRACT

Nowadays wireless networks are becoming increasingly popular in urban areas, university campuses and corporate environments. This, along with the widespread deployment of mobile devices with advanced computational capabilities, creates more opportunities for developing collaborative/social applications. The development of such applications often raises a number of challenges for the developer mostly due to synchronization issues, increased network failures of the wireless environment and limited resource availability. In this paper we present a novel framework that tackles the above problems and releases the developer from the burden of dealing with cross-domain problems. We focus on the development of a certain type of applications, inside the context of ubiquitous gaming, in a small scale peer-to-peer manner.

I. INTRODUCTION

The IEEE 802.11 standard for Wireless LANs (WLANs), also known as WiFi, is becoming increasingly popular worldwide in university campuses, corporate environments and many other public places.

Mobile, or nomadic, computing [1] is a new computing paradigm, recently established due to the latest advances in wireless communications, highly coupled with popular WLAN/WPAN standards such as IEEE 802.11. WiFi enabled devices (WLAN network interface cards or WNICs) are now becoming standard equipment for autonomous, inexpensive, and portable devices such as laptops, PDAs and advanced cell phones. Moreover, WiFi infrastructure devices (Access Points or APs) are increasingly used in households, public transport stations and other urban areas, providing seamless wireless coverage for small scale networks. This popularity, along with its easy and cheap deployment, indicates that WiFi technology is an integral part of any ubiquitous system [2] with aspirations to appeal to the masses.

The advanced computational capabilities of such devices make them suitable for supporting upcoming social applications such as ubiquitous gaming [3]. Our vision is that in the near future wireless access will be available seamlessly to the public through wireless ISPs, service aggregators or by using the contributions of individual AP owners [4], thus bringing the ubiquitous gaming concept one step closer to reality. For example, people waiting in a bus station or shopping in a mall centre, can easily discover others, by exploiting the wireless access capabilities offered in modern mobile devices, who are interested to play a game and could rapidly establish one in a peer-to-peer (P2P) manner.

This type of wireless applications faces important limitations mostly due to the dynamics of the wireless access medium, the resource implications imposed by the usage of mobile devices and the synchronization challenges between the game participants. The developer of a mobile P2P game, apart from facing the game's implementation puzzle, is also forced to deal with the above challenges. Thus, the development of small-scale mobile P2P games is impeded by the need for dealing with cross domain problems which in most cases are faced partially.

A. Overview

In this work we present a novel framework that tackles the synchronization issues, the increased network failures due to the dynamic wireless environment, and the limited resource availability of the mobile devices in order to release the developer from the burden of facing these challenges by himself. The benefits of this approach, separating the game development specific problems from the "networking" implications, are twofold: a) Different domain experts (e.g., networking experts and game developers) are free to provide solutions or optimize specific problems without considering the general context. b) This decompositional approach is expected to intrigue the interest of the networking community to standardize the network dynamicity challenges and give an aggregated solution for small-scale P2P social applications, thus fuelling the development of mobile P2P games.

The proposed solution is comprised of a layered approach of seven entities, named Game API, Framework Interface, Peer Discovery and State Maintenance Engine, Message Dispatcher, Synchronizer, Game Engine and Out of Band Communication Channel respectively, which are analysed thoroughly in Section III. Each entity accommodates the solution to a certain problem and provides its services to the other entities wherever necessary.

This hierarchical structure thus, demarcates the solution to each challenge so as to be studied and optimised solely by different research communities, without affecting the operation of the rest entities.

B. Paper Organization

The remainder of this paper is organized as follows. Section II reviews related work and compares it with our solution. Section III specifies the framework's architecture and presents the entities involved. Section IV presents the reference implementation of our proposal on top of the .NET compact framework 2.0 using QTEK 9100 mobile devices, along with two special P2P games that we developed as proof of concept and for evaluation purposes. Finally we present and analyse a full operation example in Section V and we discuss our findings and future plans in Section VI.

II. Related Work

Recent studies [5, 6] have demonstrated the suitability of coupling physical world objects with web resources. Klopfer et al. presented a system [3] that incorporates live world data for educational purposes. Our work does not involve any innovative sensing technologies; we focus on the game development procedure and how to uncouple this process from different domain challenges. Wolf and Wang presented a framework [7] for mobile P2P game development in the same context we consider, but their work is mostly focused on the design of the "peer fostering mechanism". Finally, Mohamudally [8] compared some existing collaborative frameworks for their behaviour in a particular game scenario. Although the context of this game scenario is similar to ours, the author is interested in studying the learning interactions of the players. Our main goal is to create a facilitating intermediate, in contrast with the former ones which were focused on researching certain mechanisms. Furthermore, this intermediate is designed to be a standard, something that demands a more sophisticated and global approach.

III. Framework Architecture

In this section we describe thoroughly the proposed framework's architectural design. Our framework accommodates the development of P2P software, multiplayer games and mobile applications. Its goal is to hide all networking-related challenges from the developer and let him focus on his application puzzles following a "game-centric" development approach. Furthermore, it orchestrates the synchronization of different games, a need that rises naturally in the P2P environments, where each player (gamer or opponent) can join or leave other games easily. Mobile devices have limited memory capacity, so it is harder for them to share a common application. Thus, the framework handles the exchange and deployment of mobile applications easily, in a P2P manner, among multiple opponents.

Each mobile game (i.e., the game application) needs to be "hooked-up" with the framework in order to establish a small-scale P2P multiplayer game and communicate with the other "peer games". Fig. 1 illustrates the proposed architecture. It is comprised of three primary and four secondary entities. The primary entities (i.e., Synchronizer, Game Engine and Peer Discovery and State Maintenance Engine) confront synchronization issues between games and framework instances, mobile devices' memory limitations and network instability issues respectively. The four secondary entities (i.e., Game API, Framework Interface, Out of Band Communication Channel and Message Dispatcher) arose from the game developer's subjective view (i.e., the needed-entities that arose while we ware developing our P2P games, see Section IV.B). All interactions between these entities are modelled as message transactions. Thus, many special message types exist and are analyzed in the following sections.

A. Game API

The Game API is one proposed framework's four supplemental entities. It implements the up-call "game-to-framework" interface that is visible by the developer and hides the environment's complexity (i.e., the dynamicity of wireless environment, synchronization issues and the constraints imposed by the mobile devices). Its main purpose is to forward messages from a game to the framework's Message Dispatcher entity, in order to further be routed to their destination, and vice versa. Each game must be "hooked-up" with this entity, seen as a software library by the developer, to use the framework. This way we standardize the game development and release the developer from the burden of facing problems outside the game's scope. The Game API is able to communicate with many games simultaneously, meaning that more than one application may use the same framework instance at once. This feature enables a gamer to communicate with multiple opponents and to play different games with each one of them at the same time.

By following this hierarchical - decompositional approach, the developer interacts only with the framework (through dedicated "method calls") and focuses only on the game's challenges. The synchronization, resource and network issues are handled by the rest entities.

B. Framework Interface

The interaction between different frameworks is done through the Framework Interface entity. This component is responsible for granting multiple and simultaneous connections with other frameworks, in order to support multiplayer gaming, by handling the message transportations. It resides in the lower class in the framework's hierarchy and tackles all possible transfer failures due to the wireless medium. Thus, a guaranteed transfer mechanism is provided to the rest framework entities.

Framework Interface is connected with the Message Dispatcher, in order to transfer messages between framework instances and with Peer Discovery and State Maintenance Engine, in order to initially discover, and following have estimate knowledge, of the available peers needed to communicate with.

C. Peer Discovery and State Maintenance Engine

When a peer gets connected to a wireless network or each time a small rapid P2P gaming network needs to be established, in order to support a multiplayer game, the framework has to be informed about its neighbour peers. Peer Discovery provides this information along with the "name-to-network address" resolution to the Framework Interface. During a game session some peers may lose connectivity, due to the dynamic nature of the wireless environment, or leave and others may join. Inside this "soft-state" environment State Maintenance tackles peers' churn events, either caused by the user or by an external and possible faulty cause and properly notifies the game application by utilizing a lightweight monitoring procedure.

D. Message Dispatcher

Message Dispatcher is the responsible entity for receiving messages from other framework entities and deciding where each message should be delivered. It can be considered that resembles the functionality of a message switching centre inside the framework that arbitrates all entities communications. Each entity generates and consumes messages asynchronously and consequently affects the operation of other entities. Thus, the message dispatcher centralized approach ensures the optimal operation of all entities inside the framework by eliminating starvation problems or aggressive entity behaviours, in terms of generated/consumed messages that reflect in consumed resources or bandwidth, using effective scheduling.

Before forwarding each received message, Message Dispatcher transforms it appropriately (by doing the appropriate encapsulations/decapsulations), in order to be acceptable by the receiving component. To be more specific, Message Dispatcher is involved in the following receiving and forwarding situations:

- Messages received from the Out of Band Communication Channel entity and needed to be forwarded to the Framework Interface, so as to be delivered to the opponents' framework instances.
- Messages generated from the Synchronizer entity and needed to be forwarded to the Game API or the Framework Interface entities, (depending on their context).
- Messages sent from the Game API and needed to be forwarded to the Synchronizer, in order to inform it about synchronization issues, or messages from the Game API indented for the Game State component.
- Messages sent by the Game Transfer Engine or the Game List Discovery component to be dispatched to the Framework Interface, so as be to delivered in the opponents' framework instances.
- Messages received from the Framework Interface and destined to current framework, following the inversed path of all situations described above.

E. Synchronizer

Game applications are by nature interactive. That is why synchronization is necessary not only between applications and their managing framework, but also between different framework instances. For example in the tic-tac-toe game, once a player has his move completed, his must be blocked until his opponent replies with a move. The Synchronizer entity, which is highly coupled with Message Dispatcher, ensures synchronization consistency by orchestrating the Game API and the Framework Interface entities (i.e., the two main points of generating or consuming messages).

The synchronization between the framework and the active game set (i.e., all games interacting with the framework simultaneously) is accomplished by sending special orchestration messages to the Game API. Among others these can be:

- Messages that request a player to initiate or terminate a game.
- Messages for the handshaking process, which ensures that the initial options of a game have been adjusted in order to start gaming.
- Messages for resetting or blocking a game.

The synchronization between different framework instances is accomplished by sending dedicated messages to the Framework Interface entity such as:

- Messages indented to notify the opponents' framework instances for synchronization issues.
- Messages that request the game list of the opponents' framework instances.
- Messages that synchronize a game transfer.
- Signalling messages informing about the termination of a framework instance.

F. Game Engine

This core entity is responsible for the conduct of the game and is mostly used to provide a "game-centric" development approach to the game architect. It is comprised by three distinct sub-entities, named Game State, Game List Discovery and Game Transfer Engine respectively.

1) Game State

The Game State entity traces each game's state by handling the appropriate game state messages, thus supporting the synchronization process. These messages are used for informing other games and their respective framework instances about the state of opponents. For example if all players have reached a landmark point, then previously forbidden actions can be available. Additionally, this entity keeps and provides accounting information for each game (e.g., the number of wins and loses).

2) Game List Discovery

The purpose of the Game List Discovery entity is to obtain the game lists of all other peers and to provide the local game list to others upon request. This way, game applications can be easily disseminated in a P2P manner to all peers, thus facing the limited resource availability on the mobile devices.

3) Game Transfer Engine

The Game Transfer Engine entity is responsible for exchanging games between peers. It is a compound entity to the Game List Discovery entity presented before, and used for monitoring and carrying out the game application transfer.

G. Out of Band Communication Channel

This entity ensures that a peer is able to communicate with other peers at any time, regardless of the frameworks' state. For example special signalling messages, real-time or urgent data should use this channel to communicate with other frameworks, instead of following the standard message dispatching stream, which can randomly delay the delivery.

IV. IMPLEMENTATION

We have partially implemented the proposed framework along with two game applications for evaluating and further studying our proposal. The referenced implementation was developed with Pocket PC Emulator, Microsoft Visual Studio .NET 2005 SP1 [9]. The executable code was packaged using the CAB file format [10] and the installation was done in two mobile, QTEK devices [11] (QTEK 9100). Table 1 illustrates the devices' capabilities. The reference implementation is public available at: http://mm.aueb.gr/archive.html#Software

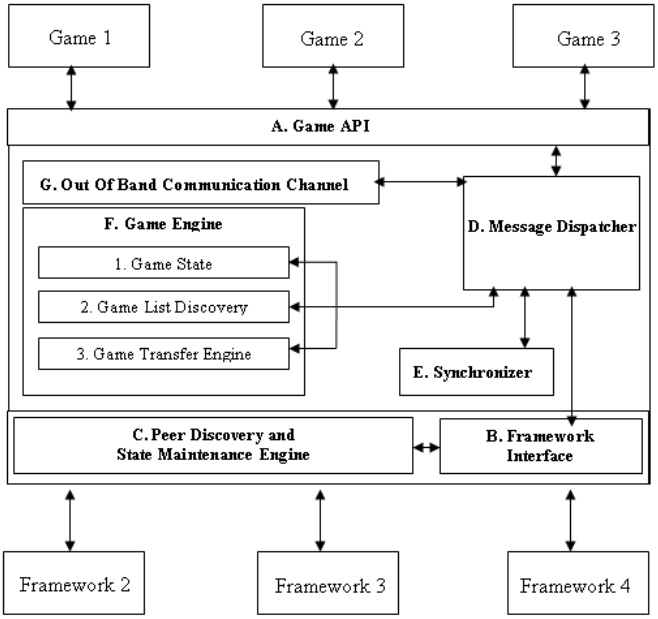

Figure 1: Framework Architecture

Table 1: QTEK 9100 technical specifications

CPU	200MHz
Memory	128KB ROM, 64KB RAM
Cellular	GSM/EDGE Quad-Band
WPAN **WLAN**	Bluetooth 1.2-IEEE 802.11 b/g
OS	Microsoft Windows Mobile v5.0

A. Framework

As far as the framework's implementation is concerned, we have implemented most parts of the proposed architecture described in Section III. We focused mostly on the interaction between two players only, although multiplayer games can still be supported with minor changes. The Peer Discovery and State Maintenance Engine is not implemented in this first implementation. Efficient ways of discovering peers inside our referenced environment and keeping a track of them is a future task, which should be studied independently in order to come out with optimal solutions. Thus, the "peer name-to-network address" mapping is statically defined from each game application[1], by using the respective method call.

[1] The application can still use dynamic discovery techniques implemented solely outside from the framework and provide the valid mappings to it.

Game API entity currently supports interactions with one game only in real-time. The implementation's environment (QTEK 9100 – Windows Mobile v5.0) is highly resource restrictive for enabling us to support multitasking game applications, which from their nature drain the device's resources. Thus, for the reasons analysed before and for simplifying the development procedure and the Synchronizer entity, the Game API is implemented as a singleton object coupled with "1-1" relationship with the game application.

The implementation of the Framework Interface entity uses TCP/IP for communicating with other peers. Although the TCP/IP stack is more resource demanding rather than the UDP lightweight alternative, the additional cost of implementing a reliable transfer solution on top of UDP datagrams imposes additional implementation cost, which was unnecessary to our testbed. Future work should definitely tackle this limitation.

Message Dispatcher, Synchronizer, Game State, Game List Discovery and Game Transfer Engine entities are fully implemented as described in the architecture section. Although the Game State component does not support the statistic information collecting mechanism, it gets informed about the actions of a game. We implemented the Game Transfer Engine, which exchanges games between frameworks as follows: every time that someone wants to get a new game from another peer, the executable file and the accompanied GINFO file of the game (described in IV.B) are transferred and stored in the framework's installation folder.

Finally considering the Out of Band Communication Channel, we used it to provide a chat mechanism, which enables the peers to communicate with each other even during a gaming session.

B. Game Application

As it is said before, we also implemented two special game applications in order to demonstrate the framework's usage and for further evaluating it. We considered each game application as a separate executable file, not installed in the mobile device, accessed and controlled only by the framework. Games supported by our framework are not standalone applications and need the framework (i.e., they need to be "hooked-up" with it) in order to operate. This way, a user is prevented from running the application without using the framework and tackling many problems by himself.

Each game intended for this framework should follow certain constraints, in order to be compatible with it. Firstly, it can never exchange information directly with another game; only by using the framework. This way, the framework manages the gaming synchronization. Secondly, it should use the types of messages described before, thus using already efficient ways to deal with the environment problems instead of "hacking" partially the raised challenges. Lastly, it should be accompanied by a GINFO file with the main characteristics of the game. Every GINFO file contains the following information:

- The name of the game.
- The identification number of the game.
- The name of the executable file.

- The network port for the incoming messages sent by the framework.
- The network port for the outgoing messages sent to the framework.
- The game description.

Following the above constraints, we developed two games. The first one is the well known tic-tac-toe game. The second one is called "Skirmish" and is a turn-based strategy game. Although Skirmish is more complex than tic-tac-toe, it operates smoothly with the framework and we did not observed any degradation in the framework's performance.

V. Operation Example

In this section we will demonstrate briefly the messages exchanged during a gaming session between two peers, in a simple scenario. We assume there are two persons in a cruise, namely Player A and B respectively, who use their mobile devices to rapidly establish a wireless P2P network in order to play a game. Player A has only the tic-tac-toe game, while player B has both the tic-tac-toe and Skirmish games. Fig. 2 illustrates the exchanged message sequence before and during the game. After the frameworks' initialization and having a short chat, both players agree in playing a game. The frameworks exchange their game lists and player A decides to play the Skirmish game. Due to the fact that he/she doesn't have this game, he/she requests it from the opponent's framework. His/Hers framework receives that game and they start playing it. When one of the players decides to leave the game the respective "clean-up" messages/actions take place.

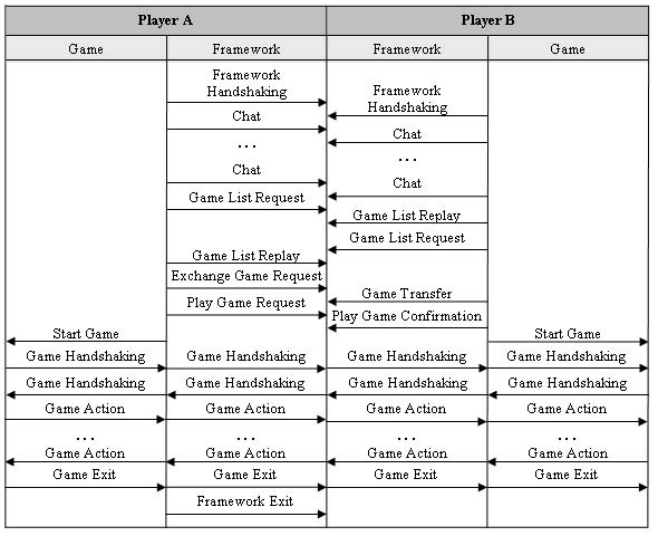

Figure 2: Exchanged messages sequence

VI. Conclusion and Future Work

In this paper we presented a novel framework that tackles the synchronization issues, the increased network failures due to the dynamic wireless environment, and the limited resource availability of the mobile devices in order to release the developer from the burden of facing these challenges.

Different domain experts are free to provide solutions or optimize specific problems without considering the general context. Moreover the decompositional approach followed in the design of the architecture is expected to attract the interest of the networking community to standardize this framework and thus provide a common skeleton for the development of small scale P2P social applications, such as mobile P2P games. Currently we are extending our proposed framework in order to face the security issues that were raised.

Finally, apart from facing all other minor limitations presented in the previous section, we plan to evaluate and benchmark the framework and its efficiency under many different dimensions. Our contribution is not to provide a solution to each of the important problems referred inside this paper, instead we try to create a standard scheme in order to support the "game-centric" development, and intrigue the research community and the developers to further assist each other so as to come up with optimal and robust solutions.

References

[1] T. Imielinski and B.R. Badrinath, "Querying in Highly Mobile Distributed Environments," in *Proceedings of the 18th VLDB Conference*, Vancouver, Canada, 1992.

[2] M. Weiser, "The Computer for the 21st Century," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 3, no. 3, pp. 3-11, July 1999.

[3] E. Klopfer, K. Squire, and H. Jenkins, "Environmental Detectives: PDAs as a Window into a Virtual Simulated World," in *Proceedings of the IEEE International Workshop on Wireless and Mobile Technologies in Education*, Växjö, Sweden, 2002.

[4] E. C. Efstathiou, P. A. Frangoudis, and G. C. Polyzos, "Stimulating Participation in Wireless Community Networks", *in Proceedings of the IEEE INFOCOM*, Barcelona, Spain, 2006.

[5] F. Bellotti, R. Berta, A. De Gloria, and M. Marganore, "User Testing a Hypermedia Tour Guide," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 33-41, April-June 2002.

[6] S. Pradhan, C. Brignone, J. H. Cui, A. McReynolds, and M. T. Smith, "Websigns: Hyperlinking Physical Locations to the Web," *IEEE Computer*, vol. 34, no. 8, pp. 42-48, August 2001.

[7] H. Wolf and M.Wang, "A Framework with a Peer Fostering Mechanism for Mobile P2P Game Development," in *Proceedings of IEEE International Conference on Mobile Business*, Sydney, Australia, 2005.

[8] N. Mohamudally, "A Massive Multiplayer Game Framework for Mobile Learning," in *Proceedings of IEEE International Workshop on Wireless, Mobile and Ubiquitous Technology in Education*, Cheju Island, Korea, 2006.

[9] "Visual Studio 2005 Developer Center", Microsoft MSDN, http://msdn.microsoft.com/vstudio/

[10] "Microsoft Cabinet SDK", Microsoft MSDN, http://msdn2.microsoft.com/en-us/library/ms974336.aspx

[11] "QTEK 9100 Pocket PC Phone Edition", WLAN-enabled, http://www.qtek.nu/europe/products/9100.aspx