

Issues related to Development of Wireless Peer-to-Peer Games in J2ME

Alf Inge Wang

Dept. of Computer and Information Science
Norwegian University of Science and Technology
N-7491 Trondheim, Norway, alfw@idi.ntnu.no

Michael Sars Norum

BearingPoint Norway AS
N-0051 Oslo, Norway
michael.norum@bearingpoint.com

Carl-Henrik Wolf Lund

Bekk Consulting AS
N-0102 Oslo, Norway
Carl-Henrik.Lund@bekk.no

Abstract

This paper describes and discusses challenges related to development of peer-to-peer games in J2ME using the available Bluetooth API (JSR82). By using Bluetooth on wireless devices, new types of personal network peer-to-peer games can be developed. In this paper, we present a classification framework for wireless peer-to-peer games that divides types of games into two different dimensions. The first dimension groups games according to the user interaction patterns and the second dimension groups games according to how data is updated. Further, the paper investigates problems that must be solved before peer-to-peer games using the existing Bluetooth API in J2ME can be developed. Here issues related to the Bluetooth API and limitations in the Bluetooth standard are discussed. Finally, the classification framework is used to reveal what types of games that can be implemented using current J2ME virtual machines and current Bluetooth API. The results presented in this paper are based on experiences from development of cooperative peer-to-peer applications for mobile phones using a framework called Peer2ME.

Keywords: *Peer-to-peer games, Bluetooth, J2ME, JSR82*

1 Introduction

Computer games played on consoles or PCs have become one of the most successful type of commercial applications, and is today one of the biggest branches in the entertainment industry. As mobile devices have become more and more powerful with improved colour screens and graphics, the game industry has started to focus on game development for mobile and wireless devices. It is expected

that by 2006, wireless gaming will generate \$17.5 billion in annual revenue worldwide [7]. The development of games for wireless devices brings new challenges to the developer, such as minimizing game data, make games for small screens, adjust the control of games to fit the keypad on wireless devices etc. [3]. Today, most mobile phones have support for running J2ME applications. In an ideal world, this would solve all portability issues of creating games for various devices. However, this is not the case. In practice, game developers using J2ME must handle variations in screen sizes, colour depths, audio features, and variation in how the virtual machine is implemented on different devices supporting different APIs that can be manufacturer independent or dependent. However, there are also benefits from using J2ME when developing wireless games. J2ME applications are packed in a standardised way making it easy to buy and install games on the wireless devices using over-the-air provision. It is even possible to use active network support (active WAP) to deploy J2ME games [8].

Today, most PC and console games support network gaming. For mobile games, this is not the case and especially for J2ME games. There are examples of J2ME games using the network for storing high-scores and game statistics, but also some games that focus on interactions between users like Samurai Romanesque [6]. In this paper, we have looked at the opportunity of development of peer-to-peer games in J2ME, and the challenges we face in doing so. There are examples of simple peer-to-peer games in J2ME today like chess and monopoly, but few of them are commercial. Through our experiences in development of peer-to-peer applications in J2ME, we want to investigate whether the Bluetooth API in J2ME and the J2ME technology is mature enough to create peer-to-peer games for wireless devices. In the work of evaluating the peer-to-peer gaming, we have created a classification of peer-to-

peer games that we use in the evaluation. The results presented in this paper are based on experiences from developing peer-to-peer applications using the Peer2ME framework [9].

The paper is organised as follows. Section 2 describes our classification framework that divides peer-to-peer games into different types depending on user interaction pattern and how data is updated. Section 3 describes general issues that must be taking into account when developing games using the Bluetooth API in J2ME. Section 4 investigates how well the different types of games grouped according to our classification framework can be supported using J2ME and the Bluetooth API. Section 5 describes related work in the area of J2ME and Bluetooth. Section 6 concludes the paper.

2 Classification Framework for Mobile Peer-to-peer Games

In this section, we present a classification framework for mobile games that divides games into categories according to their characteristics and behaviour in a peer-to-peer environment. The motivation for creating the classification framework was twofold. First, we wanted to identify the various types of peer-to-peer games to examine their characteristics. Second, we wanted to investigate how the various categories of mobile peer-to-peer games could be implemented using J2ME and the Java Bluetooth API. For the latter, the classification framework would make it easier to focus on specific usages of the peer-to-peer network.

The usual way of classifying games is to divide games into genres that reflect the game experience or gameplay. An example of such a classification is to divide games into the following categories: Action, adventure, driving, puzzle, role-playing, simulation, sports, and strategy [2]. Such classifications are commonly used by game magazines, game shops and game web sites to make it easier for the reader to focus on his preferred type of games. In this paper, we suggest another classification that is not intended to replace the existing genre classification, but rather to provide a classification that can group games according how network users interact and how the peer-to-peer network is used in the games. The classification is inspired by work within the mobile computer supported collaborative work (mobile CSCW) domain.

Mobile CSCW can be defined as: "Working together at various sites with the use of mobile IT" [12]. In other words, mobile CSCW is to run CSCW applications on mobile devices like mobile phones that operate in a wireless, mobile environment. To run CSCW applications on mobile devices can be a challenge due to limited CPU power, small screens, limited input devices and limited power source (battery). Mobile CSCW has also some advantages compared to tra-

ditional CSCW. Mobile devices (mobile phones) are highly personal and most users carry their devices with them all the time. This means that the device can be used to identify the user. Also, since the device is personal, a user will typically store a personal profile on the device, enabling the device to function according to the user preferences when interacting with other users. Mobile devices can also be considered to always be connected to a network (cellular network), giving a high availability rate of the users. This is not the case for wireless peer-to-peer applications, since they use personal area networks like Bluetooth, infrared or WLAN. Interaction in such networks is dependent of the users to be geographically collocated (e.g. within a 10 meter radius) and the interaction between the users is highly dynamic. Proximity-based ad hoc interaction, enabled by mobile phones and personal area networks, is referred to as impromptu collaboration [4]. Impromptu collaboration is recognisable as being *opportunistic*, the technology enables people to take advantage of opportunities that present themselves; *spontaneous*, the collaboration effort is not planned in any way in advance; *proximity based*, the peers have to be physically collocated; and *transient*, the interaction between peers can be very short, e.g. a few minutes or seconds. Game sessions for games using ad-hoc peer-to-peer networks are generally shorter than e.g. console game sessions, but lasts at least a couple of minutes. There are several reasons the mobile peer-to-peer game sessions are generally shorter than standard gaming sessions. First, the gamers are generally on the move and typically play games while waiting for something else to happen. Second, the energy consumption of playing network games on mobile devices will limit the length of the session. Finally, the smaller screens and limited input devices will cause the gamer to get more easily tired.

Impromptu collaboration can involve different degrees of user interaction from situations where the device will do all the interaction on behalf of the users, to situations where the user actively interacts with nearby users. Further, the update frequency of the data sent between users is an important aspect of mobile peer-to-peer gaming. Based on these two facts, we have developed a peer-to-peer game classification framework consisting of two dimensions. The *first dimension* describes how the users interact and to what degree the devices interact on behalf of the user. User interaction of peer-to-peer games can be divided into the following categories:

- I1 Controlled:** In this category, the user interactions of the gamers are controlled through a well-defined protocol, where one of the peers in the network must be a master controlling the user interaction. For games in this category, it is not up to the gamer themselves when to interact, since the master peer controls the interaction. This category suits well games that are turn-based

or games where the users must interact in specific patterns or sequences. This category also covers games where all the users must follow a specific pattern or route (e.g. driving games).

- I2 **User interaction:** In this category, the users have explicitly to trigger actions that will cause interaction (exchange of data) with other gamers. This category suits well games that include trading of e.g. weapons, equipment and other items between gamers. In addition, this category covers games where it is up to the gamer how he wants to interact with the other gamers.
- I3 **Automatic triggered:** In this category, the mobile device of a gamer searches for other gamers nearby, and if one is found it can trigger an action to get the gamers attention (sound or vibration). An example with such functionality is included in the game Nintendogs for Nintendo DS [10]. This game has a *bark mode* where your virtual dog in the game will start to bark if another player also in bark mode is within wireless range. The players can then interact and let their virtual dogs meet.
- I4 **Automatic:** In this category, the devices of the gamers interact without the user interacting with the game itself. In this category, the gamers typically configure and build autonomous characters that can interact with other users characters on behalf of their "owners". Such games can e.g. be role-playing games, where the gamer equip and train his characters, and the fighting between network gamers goes on without the gamers interacting.

As we can see from the description above, the first dimension assesses the interaction among the peers and to what degree the application can operate autonomously. The *second dimension* of the classification framework focuses on synchronisation and update of data between the peers. This dimension is divided into three categories:

- U1 **Asynchronous:** Asynchronous update is for network games where update between the peers is not time critical, but can be updated whenever possible. This category fits best games that are "slow-paced" and do need fresh data to proceed in the game.
- U2 **Synchronous:** In this category, the peers participating are depending on frequent update of data to be able to play the game. Further, for this category only small amount of information is necessary to be sent between the peers to keep the gamers in a consistent state. Such information can typically be game scores, lap-times, ranking, simple player states etc.
- U3 **Real-time:** In this category, the games rely on heavy data exchange between peers in order to give users a

game experience. The data exchange between peers can typically be position of character or a vehicle in a common world or track, state of character or vehicle, movements of character or vehicle etc.

As seen above, the second dimension identifies the amount network traffic between the peers, and whether the network is used all the time or only in intervals. Most personal area networks can cope with the U1 and U2 categories, but not necessarily U3 depending on the amount of data required to be sent between the peers. Another aspect of the U3 category is that the network lag must be minimised to avoid the game experience to stutter.

In Table 1, we have mapped various game genres that can fit into the different categories of the classification framework. Note that many games will span over many categories and will contain multiple aspects of the framework. As we can see from the classification framework, the different game genres can span over various categories. Also the same game can have elements of several categories. The classification framework can also be applied to analyse single games to assess the peer-to-peer characteristics of the game. As an example, the Nintendogs game for the Nintendo DS would fit into the following categories in the framework: (U2, I3) for bark mode, (U2, I4) for when network players let their dogs interact on their own, and (U2, I2) when network players control their dogs when interacting.

In the Section 4, we will investigate issues related to development of games in the different categories identified in this section using J2ME and the Java Bluetooth API.

3 Games, Bluetooth and J2ME

In this section we will look at issues that must be solved in order to develop peer-to-peer games using the Java Bluetooth API (JSR-82) in J2ME. The issues presented in this section are based on experiences from developing various peer-to-peer applications in J2ME.

3.1 Slow Device Discovery

The goal of the device discovery process in Bluetooth is to find all surrounding Bluetooth devices within the network range. After all devices have been found, the devices start to handshake and connect. We have conducted tests on three different mobile phones (Sony Ericsson P900, Nokia 6600 and Siemens S65) to measure the time to carry out a Bluetooth discovery and the handshake protocol. To check if there were differences between the three devices, we conducted 10 tests for three different configurations (in total 30 tests) where we found that the total time for discovery and handshake varied from 18.3 seconds to 25.4 seconds. The

Update/User interaction	I1 Controlled	I2 User interaction	I3 Automatic triggered	I4 Automatic
U1 Asynchronous	puzzle, strategy	strategy, RPG		
U2 Synchronous	simulation, sport	simulation, adventure	simulation	RPG, simulation
U3 Real-time	driving, sport	action, adventure	action	

Table 1. The P2P Game Classification Matrix

Device configuration	Bluetooth discovery (ms)	Handshake protocol (ms)	Total time (ms)
Nokia 6600 master, Sony Ericsson p900 slave	16889.7	3349.6	20239.3
Sony Ericsson P900 master, Nokia 6600 slave	18826.4	3139.8	21966.2
Siemens S65 master, Sony Ericsson P900 slave	16745.1	4508.4	21253.5

Table 2. The Bluetooth discovery and handshake times

average of the results of 10 tests per configuration is shown in Table 2.

Although, the total time is almost the same for the three devices, the tests showed great variations between the three. The main reason for this is the difference in CPU, memory and the implementation of the Bluetooth API. The Bluetooth API for J2ME forces the application to halt until a complete Bluetooth discovery is performed. This has a major impact on the usability of peer-to-peer games in J2ME. When gamers want to join a game, all gamers must wait until a complete discovery process has completed (from 18 to 25 seconds). For a mobile gamer, this is a long time to wait. Another and even bigger problem is that if new gamers want to join in a game, all gamers have to wait another 20 seconds before they can continue the game. This means that the management of joining and exiting gamers is very difficult.

3.2 Bluetooth Transfer Speed

The theoretical bandwidth for Bluetooth 1.0 is 1.1 Mbits/sec, but this bandwidth is not often reachable in practical use. We have tested the bandwidth using the Java Bluetooth API over the distances 1 meter, 6 meter and 10 meter. For 1 meter, the average transfer rate was 21 kilo Bytes per second (KBps). The variation in transfer rates of the 30 runs of the distance of 1 meter was very small. For 6 meters, the rate ranged from 19 up to 26 KBps and the average rate was also here 21KBps. For the test of 6 meters, 1 of the 30 runs failed. For the 10 meters test, the variation of transfer rates was significant. The data rate ranged from 4 to 38 KBps while the average was as low as 12.3 KBps. Two of the transmissions failed. The variation of transmission rates and the failed transmissions was probably caused by disturbances in the radio link and radio interference. All these tests were performed in an environment with little disturbance of other Bluetooth devices or other radio transmitters. This means that for games in J2ME using the Blue-

tooth API, the application cannot assume a higher transfer rate than 10KBps in an environment of various Bluetooth devices and other radio transmitters. Also, all the devices playing the same game should be within a 10 meters radius with clear sight between the devices.

3.3 Topology of Bluetooth Devices

The Bluetooth standard was not initially created to support peer-to-peer computing. The Bluetooth protocol assumes a master-slave paradigm where one of the devices must be the master of all the surrounding slaves. This means that a master must initiate a search for possible slaves, and let these slaves connect to the master. Since it is impossible for a master also to be a slave, scatternet where more than one network is connected is impossible on current Bluetooth devices. From a game-perspective, this means that only one group of gamers can be simultaneously connected, and one gamer can only be connected to one group at a time.

In theory, a Bluetooth master can connect to seven Bluetooth slaves at the same time. However, the number of connections allowed varies from phone to phone. Through tests performed in J2ME using the Bluetooth API on the three mobile phones Sony Ericsson P900, Nokia 6600 and Siemens S65, we found that these phones allowed a different number of connections. For the Sony Ericsson P900, it was only possible to connect to *one* other device, for the Siemens S65 it was possible to connect to three other devices, while the Nokia 6600 seemed to be the only phone implementing the Bluetooth specification fully and supports seven connections. This information is not documented by any of the phone providers and was found through experiments. The unknown number of Bluetooth connections supported by various phones makes it hard for the game developers to provide support for their network games. For the gamer, it can be hard to know if a network game session is failed because of lack of proper Bluetooth support in the de-

vice or bugs in the game software. Also it is more difficult for the game publishers to give out the necessary information about a game when the numbers of network players supported is unknown.

3.4 Extra Resource Consumption

One of the main challenges for game development in J2ME is to make a program that does not consume too much memory and CPU. For the development of a peer-to-peer J2ME game, this is an even bigger challenge. Memory and CPU resources that could have been used on pure gameplay must now be used to network management. Thus, it is important that the network management code is very compact and that it consumes very little resources.

3.5 Other Issues

In the previous section, we showed that the way the Bluetooth API is implemented and supported on different mobile phones varies. Another issue that is important for development of peer-to-peer applications is support for multitasking in the operating system. Some mobile phones have full support for multitasking, making it possible to run applications in the background, while others have no support for multitasking. For some peer-to-peer games, it is important to have an opportunity for the J2ME application to be run in the background to detect other gamers in the same area and trigger some action if other gamers are found. For current J2ME implementations, this is not possible. Currently, the applications in J2ME can be halted and later resumed, but not run in the background. This makes it impossible to develop some specific types of peer-to-peer games in J2ME.

In a perfect world, a J2ME application gives the same user experience when run on different mobile phones. This is of course not the case, since the mobile phones vary in screen sizes, number of colours on screen, keyboard, joysticks etc. In addition, the mobile phone manufacturers implement the J2ME virtual machine differently. This is also the cause for the J2ME Bluetooth API. From our own experiences, we found that the Sony Ericsson P900 has a faulty Java Bluetooth API implementation. When returning from a device discovery process, the P900 always returns a `DEVICE_DISCOVERY_ERROR`, even if devices have been found properly. This is apparently due to a mix up of two return values. This makes it difficult for the game developers to develop a J2ME peer-to-peer game that works for all kinds of mobile phones with the J2ME Bluetooth API.

4 Support for Framework Categories in the Bluetooth API

In this section, we will discuss how well the Bluetooth API in J2ME supports the different categories of peer-to-peer games identified in Section 2.

We will first look at the user interaction dimension of the classification framework. The first category, **I1 Controlled**, is fully supported in the Bluetooth API in J2ME. The *controlled* interaction pattern fits very well with how Bluetooth works and the way Bluetooth devices are communicating using the master-slave paradigm. A typical scenario will be that the device of one gamer will be the master and controlling the interaction of the slave devices. The interaction between the devices can either be turn-based or follow another protocol specified by the master device. The *I1 Controlled* group of applications will not demand much extra implementation in addition to the existing master-slave paradigm found in Bluetooth.

The second category, **I2 User interaction**, can also be implemented in the Bluetooth API in J2ME even though it is not directly supported through the master-slave paradigm. Here, the master device will route the events or messages of the slaves to the appropriate devices. Compared to *I1* type of applications, *I2* will demand some extra code for routing events and messages.

The third category, **I3 Automatic triggered**, can be implemented using the Bluetooth API but will not work optimally for the user. Firstly, the long discovery time will be a problem to detect other gamers before they are out of reach. Secondly, the mobile devices must run the game in foreground to be able to check for other nearby devices. This is not acceptable for most users, as they want to use their mobile phone for other purposes while waiting for other gamers. Such types of games will not be a success before the discovery time is less and J2ME applications can run in the background.

The fourth group, **I4 Automatic**, suffers from the same problems as *I3* because of long discovery time and lack of running J2ME applications in the background. For this kind of games, it is unacceptable if the games cannot run in the background - being the whole idea of such games.

We will now look at the data update frequency dimension in the classification framework. The two first groups, **U1 Asynchronous** and **Synchronous**, will not be difficult to support in the Bluetooth API in J2ME as a small amount of data is exchanged between the devices. However, the support for **U3 Real-time** applications is limited due to the maximum practical data transfer in Bluetooth for mobile phones is about 10KBps. The data transfer rate is also dependent on the number of devices being connected. It is likely that, real-time games with two players that exchange a minimum of data to be synchronised will work well. However,

games with more than two players is unlikely to work well if more than simple state data must be exchanged.

To summarise, we can see that the categories I1, I2 and U1 and U2 is well supported in J2ME Bluetooth API implementations in existing mobile phones. The other groups in the classification framework are not well supported.

5 Related Work

In [5], a peer-to-peer middleware named Mobile Chedar for mobile devices is described. The middleware is an extension to the Chedar peer-to-peer network allowing mobile devices to access the Chedar network and communicate to other Mobile Chedars. The paper identifies a shortcoming in current Bluetooth implementations having a restriction that nodes can only be connected to one piconet at a time. For Mobile Chedar this meant that only one type of topology could be used (star).

In [1], a middleware for streaming audio to Bluetooth devices is presented. The paper describes tests of streaming audio between a workstation and some laptops using Bluetooth interfaces. The tests showed that for streaming between one master and one slave device, the bit rate ranged between 22KBps to 62KBps. As the number of slaves increased, the bit rate decreased. For 5 slaves, the bit rate ranged between 5KBps to 12KBps. These numbers fit well with the number we got on our tests, and confirm that the number of slaves will cause a major decrease in bit rate.

6 Conclusions

In this paper we have presented a classification framework that can be used to analyse issues that you must take into account when developing wireless peer-to-peer games. Further, we have identified shortcomings in current implementation of the Java Bluetooth API for J2ME. We have found that the Bluetooth API has limited support for creating peer-to-peer games that involve automatic exchange of data, and games that require real time updates of data. The main problems of the Bluetooth implementation of existing mobile phones are slow discovery time, low transfer rates and no support for scatternets. Will this change in the future? The Bluetooth specification version 2.0 promises to solve the problems presented in this paper. The version 2.0 is promised to provide up to 3 times faster transmission speeds, faster device discovery, lower power consumption, simplification of multi-link scenarios due to more available bandwidth [11]. It will also be backwards compatible to earlier versions.

References

- [1] P. Bellavista, C. Stefanelli, and M. Tortonesi. The ubiQoS Middleware for Audio Streaming to Bluetooth Devices. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 138–145, Boston, Massachusetts, USA, August 22–26 2004.
- [2] GameSpot. GameSpot: Video Games PC Playstation 2 GameCube PSP DS GBA PS2 PS3 Xbox 360 Playstation 3. <http://www.gamespot.com/>, 2005.
- [3] D. S. Kochnev and A. A. Terekhov. Surviving Java for Mobiles. *IEEE Pervasive Computing*, 2(2):90–95, 2003.
- [4] G. Kortuem, J. Schneider, D. P. Thaddeus, G. C. Thompson, S. Fickas, and Z. Segall. When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In *First International Conference on Peer-to-Peer Computing*, Linköping, Sweden, 27–29 August 2001.
- [5] N. Kotilainen, M. Weber, M. Vapa, and J. Vuori. Mobile Chedar – A Peer-to-Peer Middleware for Mobile Devices. In *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 86–90, Kauai Island, Hawaii, March 2005.
- [6] J. Krikke. Samurai Romanesque, J2ME, and the Battle for Mobile Cyberspace. *IEEE Computer Graphics and Applications*, 23(1):16–23, 2003.
- [7] N. Leavitt. Will Wireless Gaming Be a Winner? *IEEE Computer*, 36(1):24–27, 2003.
- [8] L. Lefèvre and A. Saroukou. Active network support for deployment of Java-based games on mobile platforms. In *First International Conference on Distributed Frameworks for Multimedia Applications*, pages 88–95, Besancon, France, February 6–9 2005.
- [9] MOWAHS. Peer2me. <http://www.peer2me.org>, 2005.
- [10] Nintendo. Nintendogs for Nintendo DS. <http://www.nintendogs.com/>, 2005.
- [11] B. SIG. Bluetooth Wireless - News. <http://www.bluetooth.com/news/releases.asp?A=2&PID=1437&ARC=1>, 2005.
- [12] M. Wiberg and Åke Grönlund. Exploring Mobile CSCW: Five areas of questions for further research. In *Proceedings of IRIS23 (Information Research in Scandinavia)*, Trollhättan, Sweden, 2000.