

# Interactive Cell Phone Games

## Abstract

Mobile devices, and most especially cell phones, are now ubiquitous across the globe. The enormous customer base combined with audio and visual interfaces has attracted a wide range of game developers to the cell phone market. At the same time, the screen size and performance limitations inherent in cell phones have led to a glut of low quality games that capture passing interest at best. The nature of single player video games today requires a semi-immersive environment for success, which is generally beyond the capability of cell phones to deliver. In this paper, we explore the added element of player to player gaming. We believe distributed gaming on cell phones with anonymous opponents adds an extra dimension to game play that may make up for the lack of appeal of cell phones as gaming devices. We explore the creation of a distributed tic-tac-toe game as a case study. A working proof of concept is demonstrated along with an exploration of development costs, total power consumption and end user satisfaction.

## 1. Introduction

Until now, most cell-phone games have been designed to be only for a single person. A mobile user plays against virtual opponents, with game play limited by rudimentary AI based on inexpensive phone processors. Recently, the mobile gaming industry has explored multiplayer games via bluetooth and high speed wireless connections [4][5][6]. Bluetooth suffers from proximity limitations making it difficult to find gaming partners.

High speed wireless data via cell phone is beginning to enter the mainstream with all major service providers selling high speed data plans. In addition, portable devices such as the Nokia 770 offer native WiFi support along with improved graphics [7]. We are interested in exploring multiplayer cell phone gaming over high speed wireless networks with an emphasis on limited dependency on proprietary servers.

Today, as more and more cell phones provide connection to the Internet via WiFi (as Nokia 770 does) it is easy for cell phone users to have access to a distributed platform and play against each other.

The goal of this work is to create a distributed game application over IP that enables mobile users to create, join and play a game interactively with other users. We use the tic-tac-toe game as a case study and we demonstrate our application on the Intel PXA27X platform. We show two versions of this application. In the first generation, we implement a very basic text based and server dependent tic-tac-toe game in order to build familiarity with the tools and determine whether such an application is viable in terms of power consumption. In the second generation, we reduce server dependency dramatically (only used for player lookup) and incorporate Trolltech's QT/Embedded GUI libraries to provide a more user friendly experience.

Both generations of the tic-tac-toe case study application communicate using the Common Object Request Broker Architecture (CORBA). We adopted Python as our development language of choice due to its ease of development and strong cross-platform support.[1,2]. This in turn led to the selection of the freely available Python based fnorb for our Object Request Broker (ORB) implementation.

Fnorb is very lightweight and supports the CORBA 2.0 specification [3].

## **2. Technologies and Tools**

### **2.1 Distributed Objects**

In a distributed environment of embedded systems, the “client-server” communication model tends to be substituted by a model according to which each object is server and all objects calling the methods of the server are considered to be the clients. In the Distributed Object model, object oriented techniques are applied simplifying the use of advanced techniques, such as the asynchronous communication. The basic communication operation of a system consisting of distributed objects is the Remote Method Invocation (RMI), where a method of an object is remotely called. The most popular prototype of distributed objects is CORBA (Common Object Request Broker Architecture), which has been introduced by the Object Management Group (OMG) [1].

#### **2.1.1 CORBA in Embedded Systems.**

CORBA has been used as middleware in distributed embedded applications during the last decade. One of the early applications using CORBA is in [10] where an API is developed for device-independent robotics control. Since the dawn of the 21<sup>st</sup> century, research has been focused on the enhancement of CORBA ORBs with features useful for the today's embedded system needs. Real Time CORBA [8] provides a framework exploiting the capabilities of the embedded processors, the communication resources and the OS schedule mechanisms in order to provide efficient end to end real time communication. Application specific CORBA ORBs [9] have been specialized for optimum behavior in varying environments, with the example being limited memory on the target platform. From these research activities, it is clear that CORBA is a successful middleware solution meeting many diverse requirements for distributed embedded applications.

### **2.2 Multi-Platform Language Selection**

Embedded system development requires multi platform support as well as quick development turnaround. We selected the Python scripting language based on its support for both Windows and Linux. In addition, Python is a language designed for ease of development and has a large user base and plentiful online documentation. Because of its scripting nature, it has a lower startup time than C++ and Java and can accomplish more with fewer lines of code. The interpretive nature of Python also assists in ease of development on an embedded systems by allowing the developer to avoid the complexity of compiling and linking applications.

### **2.3 GUI Selection**

We had several GUI options available. Python has support for Tcl/Tk via the Tkinter module. PyGames uses the free and widely used Simple Directmedia Library. We also had the option to incorporate other GUI implementations via Python's support for incorporating C applications as modules. In the end, we chose to go with the popular third party QT/Embedded GUI implementation that is already widely used in Embedded Linux. Our reviews of literature in this area indicate that integrating several GUI implementations on a single embedded system is problematic and we didn't want to conflict with QTE [11]. In addition, the free version of QT/Embedded is available under the GPL license which fosters wider adoption and better community support.

### 3. First Generation

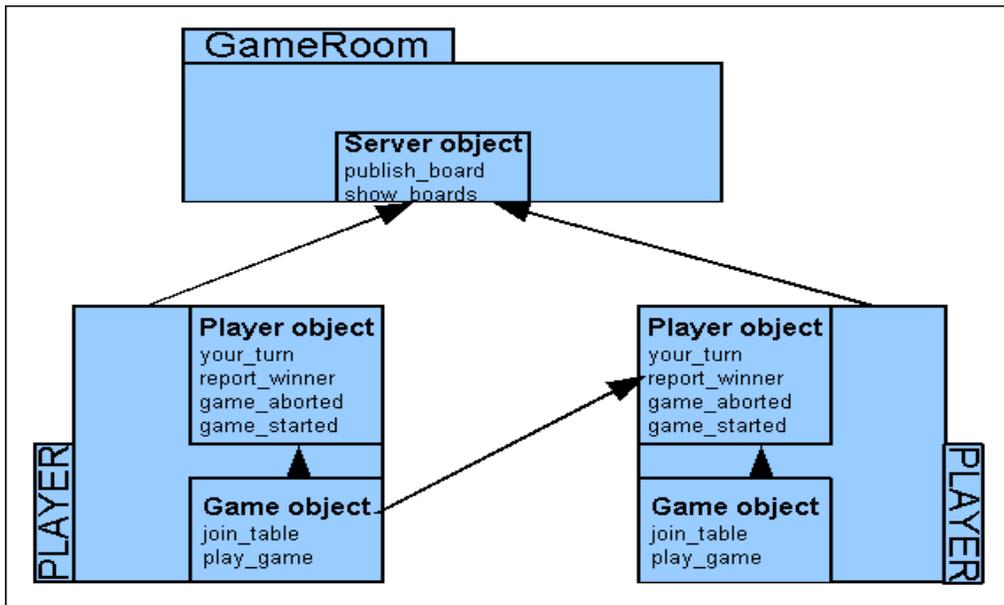
#### 3.1 Description

There is a central service, called *GameRoom* which brings two mobile users (the players) in contact. This service gives players the following choices: Either they can publish a new tic-tac-toe game in which they are the hosts and they will wait for an opponent, or they can query the system about the already published games and if they wish, they can join one of these games.

Each player contains the implementation of the tic-tac-toe game, so there is no need for the GameRoom to participate in the game interaction. The CORBA server is used only for establishing initial contact between gamers. We can easily envision using other services for player lookup, including Web Services, HTTP, and even FTP. Any technology that provides access to a stringified client IOR can be used to coordinate players.

#### 3.2 Distributed Architecture and Game Play

Figure 1 shows the architecture of the distributed objects of the first version of the application. In this figure there is an entity called GameRoom, which implements a distributed object called Server Object. Each Player instance, implements the player interface and a tic-tac-toe game object. In this figure the remote method invocation is represented as an arrow from the caller interface (not necessarily a distributed interface) to the callee object, which always need to be a distributed interface.



**Figure 1. First Generation Tic-Tac-Toe Architecture**

According to this figure, when a player wants to begin a new game, it creates an instance of the *game* object and it publishes a reference on the server. When another player wants to join a game, they request a list of available boards from the server. Then this player decides which board to join. A game starts when two players have joined the same game. The game object is responsible for tracking moves and notifying players when it's their turn. Movements are passed via the `play_game` method of the game object. When the game ends (with a win, a draw, or an abort), the game object notifies both players that the game has finished and reports the winner (if any).

### 3.3 Evaluation of the first generation.

The main problem of this application is that it is not user friendly. As there is a text based interface for the user interaction, a user is not aspired to play this game. Moreover, the game scenario according to which a user asks the server to list the available games and then decides which to join, is not a good idea considering the small size of the screen of the xscale. Assuming that there are a lot of published games, one could have trouble to see the entire list on such a small screen.

On the other hand, the power performance of the embedded application is satisfied. Using the *top* utility we realize that although each platform involves the running of an ORB on the background, it does not increase significantly the CPU usage, as the latter remains under 5% during the entire execution.

The combination of these perceptions gives us the notion that a better user interface with a more viable game scenario will result in a more appealing application, without a significant power overhead caused by the distributed objects.

## 4. Second Generation

### 4.1 Second Generation Architecture

Figure 2 captures the implementation of the second generation architecture. The second generation implementation only uses the Linux server for publishing player references. Requests for opponents are not initiated until the individual players have selected the “New Game” button on the GUI. A call to `getOpponent` will the player or return an opponent reference if someone is already registered. The player who gets an opponent reference establishes two-way communication by calling `greet`. Once two player communication is established, they can play an unlimited number of tic-tac-toe games.

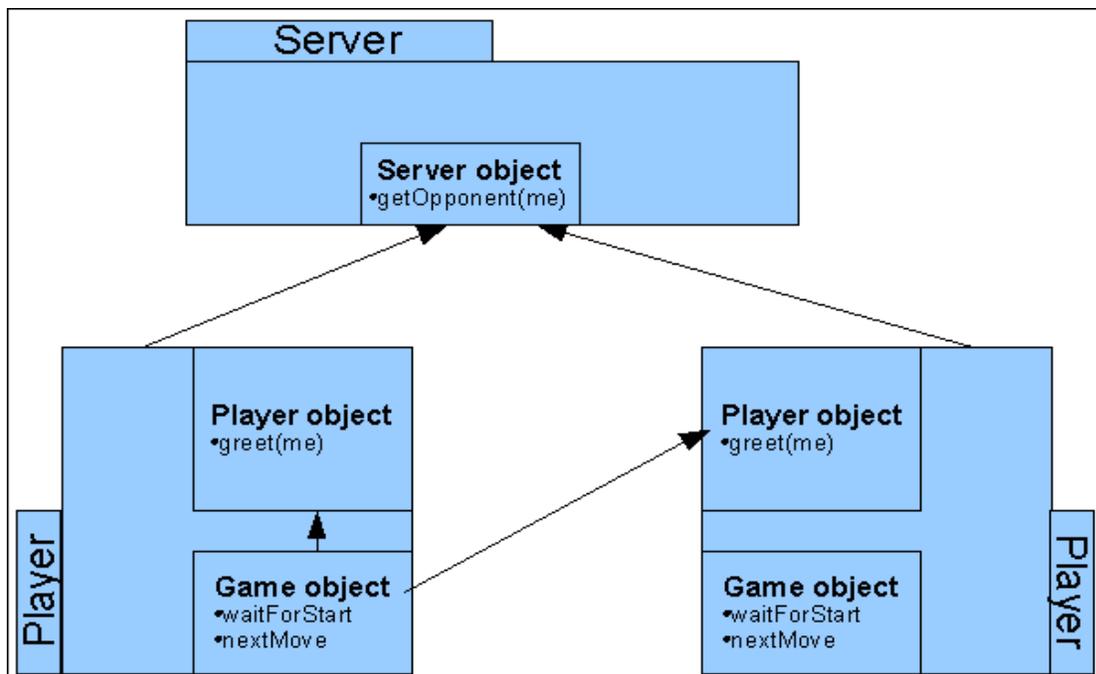


Figure 2. Second Generation Tic-Tac-Toe Architecture

## **4.2 Simplified Game Lookup**

After reviewing the first generation game publication and lookup design, we decided to simplify the player matching process. Our second generation implementation has a single button to initiate player lookup. Any two sets of players are automatically matched up to begin game play. We based this design decision on the simplicity of the distributed web game Google Image Labeler, which has a single button to initiate game play and only three buttons and a text entry field for game play [12].

## **4.3 GUI Implementation**

The QT/Embedded tic-tac-toe reference implementation was wrapped in a Python module for incorporation into our distributed gaming application. We stripped out the single player implementation and other unnecessary options from the GUI tool and added support for multi-threaded communication with the Python wrapper. The addition of lock and communication primitives allowed us to leave the GUI in a fully responsive state while the CORBA implementation ran in the background handling the distributed player matchup and game communication.

## **5. Evaluation of the system**

### **5.1 CPU Usage and Responsiveness**

The first generation of our implementation had minimal CPU requirements. We were able to run the application with imperceptible lag with the xscale CPU set to the lowest frequency settings. This demonstrates that Python and fnorb by themselves are very low impact and perform well in a power constrained embedded environment.

The second generation of our implementation also proved to be functional at the lowest CPU settings. However, it used over 90% of the CPU cycles, indicating that the QT/Embedded GUI introduced much more demanding system requirements. We attribute this to higher refresh rates and sampling for mouse movement. Memory requirements were the same for both implementations (around 5.6 M).

One unexplained behavior is that the application continued to use over 90% of the CPU for all frequency scalings. We attribute this to a bug in the QT/Embedded implementation but it may be due to some incompatibility between Python and QT/Embedded.

### **5.2 User Impressions**

For the evaluation of the system, we show the results of a small scale survey we conducted about the usability of this application. We asked 5 fellow students to play tic-tac-toe with each other in the two xscale platforms we have available in the Embedded System Lab. After that, we asked them to answer the following two survey questions: 1) Are they aware of similar applications and 2) Would they play this game or a similar game if available?

The students surveyed indicated awareness of IR based distributed games, but had no direct experience with WiFi based distributed gaming on cell phones. All students said they would play the game, but one indicated that it would need to be inexpensive or free.

## **6. Conclusions**

We are able to successfully design and implement a distributed tic-tac-toe game on the xscale platforms. The game was very easy to start and play for users, with player matchup handled in an intuitive manner. The GUI is clean and simple, but the game proved to be more entertaining for the designers and those surveyed because of its interactive nature.

The selection of Python as the development language turned to to be a boon, as its support for embedded C modules made integration of the QT/Embedded GUI possible. Python proved to be easy to learn and supported all of our programming needs, including object oriented programming and threading/thread safety considerations.

At the same time, we discovered that specialized build processes for various packages made porting applications to the xscale an extremely challenging process. Python's bootstrapping process and QT/Embedded's reliance on Trolltech's tmake required an extra level of effort beyond that required by packages based on gmake and the configure utility. Based on our experiences during the design and development process, we believe that much of the development effort in distributed games will be put towards configuration and cross compilation of the embedded systems themselves.

In the end, we conclude that the addition of a distributed component to gaming on cell phones is entirely feasible and need not depend on proprietary servers. Even with the challenges of cross compilation and learning a new system, we were able to come up with a game based on a lightweight middleware architecture that was both low power and interesting. The successful results of this case study give us the notion that games involving the interaction between real users can be developed if there is a lightweight middleware infrastructure.

## REFERENCES

- [1] <http://www.omg.org>
- [2] <http://www.python.org>
- [3] <http://sourceforge.net/projects/fnorb>
- [4] <http://www.pokerroom.com/mobile/>
- [5] [http://www.zgroup-mobile.com/games\\_bluetooth.html](http://www.zgroup-mobile.com/games_bluetooth.html)
- [6] <http://www.eamobile.com> [bejewelled multiplayer]
- [7] <http://www.nokia.com/770>
- [8] D.G Schmidt, and F. Kuhns, An overview of the Real-Time CORBA specification, *IEEE Computer*, Volume 33, Issue 6, June 2000, pp 56 - 63
- [9] V. Subramonian, G. Xing, C. Gill, C. Lu, and R. Cytron, Middleware specialization for memoryconstrained networked embedded systems, in the Proceedings of the 10<sup>th</sup> IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2004, 25-28 May 2004, pp 306-313
- [10] R.L Burchard, and J.T Feddema, Generic robotic and motion control API based on GISC-Kit technology and CORBA communications, in the *proceedings of the IEEE International Conference on Robotics and Automation*, 22-28 Apr 1996, Volume: 1, p.p 712-717
- [11] <http://www.tcl.tk/zaurus>
- [12] <http://images.google.com/imagelabeler/>