Construção de um protótipo de comunicação por tecnologia Bluetooth para o servidor Open Simulator

(Technical report APEX-TR04)

PTDC/EIA-EIA/116069/2009

André dos Santos Leal Gomes

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR

Índice

Capítulo 1 - Introdução

Este projecto insere-se numa bolsa de iniciação científica e tem por objectivo construir um protótipo base de um programa que seja capaz de estabelecer a comunicação sem fios através de um dispositivo móvel e um servidor de animação de mundos virtuais. É intenção que no final o programa comunique com um servidor através de bluetooth e que este, por sua vez, seja uma ponte até ao servidor de mundos virtuais. Deve conseguir-se que sejam trocadas mensagens nos dois sentidos.

O presente documento está dividido em cinco grandes capítulos para além deste que compõe a sua introdução. No Capítulo 2 será apresentada a plataforma OpenSimulator - o que é o OpenSimulator, para que serve e em que medida se encaixa neste projecto. No Capítulo 3 é feita uma breve referência à tecnologia bluetooth num âmbito geral. Explica-se em que consiste esta tecnologia e como se pode preparar um ambiente de trabalho para trabalhar com ela. No Capítulo 4 é apresentada a biblioteca .Net que foi usada e como se poderá instalá-la. No Capítulo 5, o mais importante, é onde é explicada a arquitectura da aplicação bem como o seu código para que se possa perceber todo o funcionamento da mesma. Por fim, no Capítulo 6, é feita uma breve conclusão sobre o trabalho desenvolvido.

Capítulo 2 - OpenSimulator

O projecto OpenSimulator, normalmente apelidado de OpenSim, é um projecto que consiste numa aplicação-servidor open source muito semelhante ao conhecido Second Life, que tem como função principal o alojamento de mundos virtuais (ver Figura 1). Através de uma aplicação cliente, o utilizador liga-se ao mundo virtual e acede a um mundo que é carregado pelo servidor. Neste mundo, o utilizador pode interagir com outros e viver uma vida totalmente virtual.

O OpenSimulator usa uma biblioteca do Second Life para cuidar da comunicação entre o cliente e o servidor, e, por esse motivo, é possível connectar um servidor do OpenSim através de um cliente do Second Life. Outros clientes para o Second Life podem também ser connectados uma vez que o Second Life e o OpenSim usam os mesmos protocolos de comunicação.

Figura 1 - Screenshot do mundo virtual num servidor OpenSimulator.

O OpenSimulator pode funcionar em dois modos distintos: individual ou em modo grid. No modo individual, um único processo trata de todo o simulador. No modo grid, vários aspectos de simulação são separados entre múltiplos processos, que podem ser executados em diferentes máquinas. O modo individual é simples de configurar mas é, por outro lado, limitado a um pequeno número de utilizadores.

O modo grid tem como potencial o facto de ser bastante escalonável conforme o crescimento do número de utilizadores.

O OpenSimulator é escrito na linguagem de programação C#, e pode correr sob a plataforma open-source Mono ou Microsoft.Net. Graças à sua construção limpa e modularizada, o OpenSimulator é uma plataforma facilmente extensível em termos de funcionalidades através da adição de módulos *plug-in*.

No âmbito deste projecto, é esta a plataforma sob a qual será trabalhada a aplicação a construir. A aplicação deverá, de algum modo, efectuar uma comunicação com o servidor do Open Simulator. Mais adiante será visto como isto é feito.

Capítulo 3 - Tecnologia Bluetooth

O Bluetooth é um protocolo proprietário que usa redes sem fios para troca de dados em curtas distâncias. Através de pequenas ondas de rádio, ele cria redes pessoais sem fios (do inglês *wireless personal area networks* - WPAN's) e com isto consegue que tanto dispositivos fixos como dispositivos móveis comuniquem entre si. Nos exemplos de dispositivos que usem Bluetooth nos dias de hoje estão os telemóveis, PDA's e pocket PC's, impressoras, computadores portáteis, máquinas digitais, consolas de videojogos, entre muitos outros.

A título de curiosidade, a empresa responsável pelo desenvolvimento e patenteamento desta tecnologia é a *Bluetooth Special Interest Group*. Este grupo não é mais do que um aglomerado de mais de 2000 empresas de todo o mundo que se juntaram com o propósito de produzir as especificações para o Bluetooth 1.0, em 1999. Entre elas estão nomes como: Agere, Ericsson, IBM, Intel, Microsoft, Motorola, Nokia e Toshiba.

Figura 2 - Dispositivos Bluetooth.

O objectivo desta tecnologia é conseguir transmitir dados entre equipamentos que possuam um circuito de rádio de baixo custo. O limite máximo para o estabelecimento de comunicação entre eles é normalmente um raio de algumas dezenas de metros para emissores de classe II e ligeiramente menos de que uma centena de metros para emissores mais capazes, de classe I. Neste sentido, e ao contrário as ligações infra-vermelhos (IrDa), os dispositivos que usem Bluetooth não necessitam de estar alinhados directamente um com o outro para se poderem

comunicar, o que faz com que esta tecnologia seja mais flexível na sua utilização e permita a comunicação entre duas salas que estejam próximas, por exemplo. Assim, devido a estas características e tal como foi referido anteriormente, a tecnologia de Bluetooth serve essencialmente para ligar periféricos sem a necessidade de se estar a usar fios para o efeito. Dos exemplos já mencionados talvez os telemóveis sejam o caso mais popular no uso da mesma, com a finalidade de trocar informações como contactos, fotos e media diverso com outros telemóveis e computadores. O auricular Bluetooth que substitui o tradicional auricular com fios é outro exemplo do uso desta tecnologia.

Microsoft Bluetooth Stack no Windows

O sistema operativo Microsoft Windows XP, a partir da versão Service Pack 2, já inclui a sua própria *Bluetooth Stack*, ou seja, o software necessário para que se possa usar a comunicação por Bluetooth entre um dispositivo externo e o computador. Comparando a versão da *stack* do Windows com outras implementadas por outras companhias esta tem, provavelmente, menos suporte que as demais. Porém, é suficientemente capaz de cobrir a grande maioria das necessidades do dia-a-dia sejam elas a ligação a uma impressora ou comunicação com um dispositivo móvel (telemóvel ou PDA). A *stack* do Windows tem outra vantagem em relação à concorrência que é o facto de estar fortemente integrada com o sistema operativo.

Para instalar este software não é necessário fazer nada além de connectar o adaptador Bluetooth. Como foi referido, a partir do Microsoft Windows SP2, a *stack* já vem com o sistema operativo. Assim, alguns instantes depois do adaptador ter sido connectado ao Windows pela primeira vez, este vai detectá-lo e irá instalar/activar a *stack* automaticamente. Durante o processo será pedido ao utilizador que atribua um nome à sua máquina que servirá para, no futuro, outros dispositivos a identificarem e saberem se essa máquina está visível para eles ou não.

Emparelhamento

Muitos dos serviços fornecidos através de Bluetooth podem expôr dados privados e confidenciais ou permitir que parte da conexão (um ou outro utilizador) possa controlar o dispositivo a que se conectou. Por razões de segurança é por isso necessário controlar que dispositivos podem ser conectados a um outro dispositivo dado. Ao mesmo tempo, e sempre que estejam no alcance, é útil para os dispositivos Bluetooth que seja possível estabelecer uma ligação entre aparelhos que já se conhecem e que saibam que é seguro estabelecer essa mesma

ligação sem a intervenção do utilizador. Para resolver este problema, esta tecnologia usa um processo chamado emparelhamento (do inglês *pairing*). Dois dispositivos necessitam de estar emparelhados para poderem comunicar um com o outro. O processo de emparelhamento é desencadeado geralmente da primeira vez que um dispositivo recebe um pedido de conexão por parte de um outro com quem nunca foi emparelhado. Uma vez estabelecida esta "ponte", os dispositivos lembrar-se-ão no futuro a quem se poderão conectar sem a necessidade de autorização por parte do utilizador. Se for desejado, a relação de emparelhamento pode ser removida a qualquer altura.

O modo como este processo acontece na prática é muito simples: o dispositivo que deseja estabelecer a conexão tem de introduzir uma palavra-passe (*passkey*) que o outro dispositivo ao qual se deseja conectar também deverá conhecer. Desta forma, garante-se que não existem ligações não desejadas e perigosas. Assim, e depois dos dois dispositivos terem introduzido a mesma palavra-passe, o emparelhamento é concluído com sucesso e a conexão é feita. A partir deste momento cada um dos dispositivos conhece o outro e ligar-se-á a ele de cada vez que este esteja no seu alcance. Como foi referido anteriormente, estes perfis de emparelhamento poderão ser removidos a qualquer altura.

Capítulo 4 - In The Hand - 32feet.Net

Por forma a conseguir o objectivo deste projecto, ou seja, a comunicação a partir do mundo virtual do Open Simulator com um dispositivo móvel real, através de tecnologia sem fios bluetooth, foi necessário proceder a alguma pesquisa com o intuito de encontrar algum tipo de ferramenta que fosse útil neste contexto.

Foi encontrado o 32feet.Net, que é um projecto concebido pela empresa In The Hand com o objectivo de tornar as tecnologias de rede de área pessoal como o Bluetooth, IrDa (Infra-Vermelhos) e outras, facilmente acessíveis a partir de código .Net.

Este projecto suporta aplicações desktop, móveis e sistemas embebidos. Actualmente abrange as seguintes tecnologias:

- Bluetooth
- IrDa (Infra-Vermelhos)
- ObjectExchange

O suporte a Bluetooth requer um dispositivo com a Microsoft Bluetooth Stack instalada. Requer também a .Net Compact Framework v2.0 ou acima ou a Windows CE.Net 4.2 ou acima para dispositivos móveis. Para ambientes desktop com Windows XP ou Vista ou acima é necessário ter instalada a .Net Framework v2.0, pelo menos.

Instalar a biblioteca 32feet.Net

Para se poder usar a biblioteca com os métodos e classes que ela poderá oferecer ao programador é necessário proceder a uma pequena e simples instalação. Esta instalação irá copiar para uma pasta um ficheiro *.dll* que deve ser referenciado no Microsoft Visual Studio. Para além disto, também ficam disponíveis uma série de demos de diversos projectos, utilizando esta biblioteca. O código-fonte dos demos encontra-se tanto em C# como em Visual Basic .Net.

Referenciar a biblioteca 32feet.Net num projecto Microsoft Visual Studio

Depois de instalada, a biblioteca 32feet.Net deve ser referenciada, dentro do Microsoft Visual Studio, para que os seus métodos fiquem disponíveis dentro da

classe que se está a construir. Para tal, basta ir ao menu de *Solution Explorer*, abrir a pasta com o título *References*, e com o botão direito do rato fazer *Add Reference*. Depois deste passo basta procurar na lista de referências a que tem o nome de *InTheHand.Net.Personal* e adicioná-la.

Capítulo 5 - BluetoothApp

A aplicação *BluetoothApp* é constituída por dois blocos fundamentais: o *BluetoothAppServer* e o *BluetoothAppClient* (ver Figura 3). Como os próprios nomes o indicam, nestes dois blocos, há um que funciona como o cliente e um outro que funciona como o servidor.

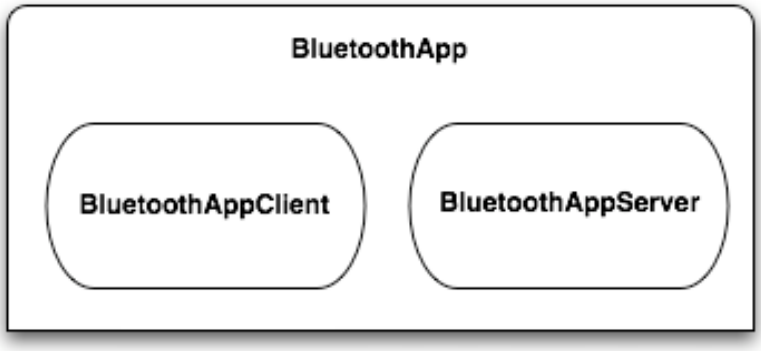

Figura 3 - Módulos do BluetoothApp.

Estes dois módulos são estritamente necessários. Como já foi mencionado, o objectivo é fazer a comunicação do avatar (personagem) que está a interagir no mundo virtual, dentro do servidor OpenSim, com um dispositivo móvel que se ligue a ele por bluetooth. Deverá ser possível enviar e receber mensagens do lado de quem está com o dispositivo móvel e também do lado de quem está a "jogar" no OpenSimulator.

Neste sentido, a aplicação foi pensada de forma a que o servidor OpenSim comunique com um outro servidor através de Sockets. Este outro servidor é o *BluetoothAppServer*, pois irá tratar de toda a parte de gestão das mensagens que circulam desde o dispositivo móvel até ao servidor OpenSim e vice-versa. O *BluetoothAppServer* comunica através de tecnologia bluetooth com o dispositivo móvel (um PDA, neste caso). Assim, a parte servidor da aplicação (*BluetoothAppServer*), será a responsável pela comunicação, por Sockets, com o servidor OpenSim e, tendo esta ligação estabelecida, deverá ser capaz de receber e enviar mensagens com sucesso para o mesmo. Ao mesmo tempo, deverá ser também capaz de enviar e receber mensagens do PDA (através da aplicação *BluetoothAppClient*) e reencaminhá-las correctamente para o servidor OpenSim.

Resumindo, conforme se vê na figura Figura 3, pode-se dizer que a aplicação principal é constituída pelos dois blocos apresentados em que um deles é executado num PDA e o outro é executado numa máquina Desktop. O bloco executado no PDA tem a função de receber e enviar mensagens para o bloco que está a ser executado na máquina Desktop. Este, por sua vez, faz o redirecionamento para o servidor OpenSim para que, quem esteja a jogar, possa ver as mensagens a aparecerem no visualizador do jogo.

Na Figura 4 que se segue, pode ver visualizada a arquitectura geral da aplicação que foi descrita:

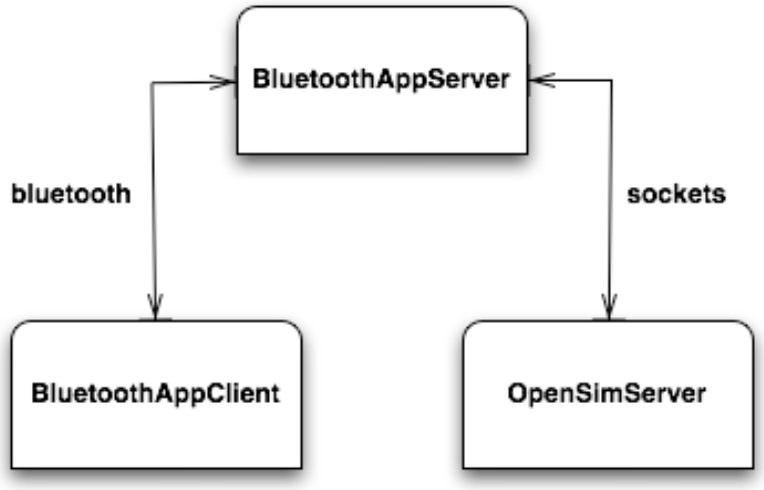

Figura 4 - Arquitectura da aplicação.

Este diagrama poderia ter uma alteração e no lugar de *BluetoothAppClient* poderia estar apenas a palavra "PDA". Porém, a nível técnico e em relação à forma de implementação, esta é a maneira mais correcta de expressar o que realmente acontece. O módulo *BluetoothAppClient* é a parte que vai ser executada no dispositivo móvel pretendido e é a partir dele que se efectua o login no servidor OpenSim. É também graças à parte interactiva deste módulo que o utilizador consegue inserir e enviar mensagens, que serão enviadas para o *BluetoothAppServer* e posteriormente reencaminhadas para o servidor OpenSim.

BluetoothAppServer

Como já foi mencionado, esta parte da aplicação é responsável pela comunicação PDA (*BluetoothAppClient*) - *BluetoothAppServer* e *BluetoothAppServer* - OpenSim, cada uma delas nos dois sentidos.

Esta é a parte da aplicação que é executada num computador Desktop, enquanto que a outra parte (*BluetoothAppClient*) é a que vai ser executada num cliente, ou seja, num dispositivo móvel. Neste caso, o dispositivo móvel que foi utilizado foi um PDA/PocketPC com Windows Mobile 5.

A comunicação entre o PDA e o BluetoothServer é feita através da tecnologia bluetooth e da biblioteca anteriormente apresentada - 32feet.Net.

Apresentam-se, em seguida, as variáveis de classe respeitantes à classe *Server* do namespace *BluetoothAppServer*.

```
namespace BluetoothAppServer
{
    class Server
    {
        const int MAX_MESSAGE_SIZE = 128;
        const int MAX_TRIES = 3;

        public BluetoothClient btClient;
        public BluetoothListener btListener;

        public BluetoothDeviceInfo[] devices;
        public bool listening;
        public Guid ServiceName;
        public String str;

        public System.Threading.Thread t1;
        public System.Threading.Thread t2;

        public TcpClient socketForServer;
        public NetworkStream networkStream;
        public System.IO.StreamReader streamReader;
        public System.IO.StreamWriter streamWriter;

        public String deviceConnected;
```

A variável *btClient* do tipo *BluetoothClient* fornece um serviço para uma conexão de um cliente por bluetooth, enquanto a variável *btListener* do tipo *BluetoothListener* fornece um serviço que fica à espera de conexões por parte de clientes.

Estas variáveis são importantes na conexão que é feita entre a parte cliente da aplicação (*BluetoothAppClient*), a executar no PDA, e esta parte do servidor. Uma é importante para o envio de mensagens para o PDA (*btClient*), a outra é importante para a recepção das mesmas vindas do PDA (*btListener*). Atenção que isto que está a ser mostrado agora diz respeito apenas ao servidor de bluetooth. A parte da aplicação que irá executar no PDA terá que conter duas variáveis semelhantes, para que a troca de mensagens se processe com sucesso nos dois sentidos.

Mais adiante, serão também explicado um outro conjunto de variavéis igualmente importantes na inicialização desta classe que têm a ver com a conexão feita por Sockets com o servidor OpenSim.

Em seguida, apresenta-se o construtor da classe *Server*.

```
public Server()
{
            this.btClient = new BluetoothClient();
            this.devices = this.btClient.DiscoverDevices();
            this.btListener = null;
            this.listening = true;
            this.ServiceName = new Guid("{E075D486-E23D-4887-8AF5-
DAA1F6A5B172}");

            this.str = null;

            this.t1 = new
System.Threading.Thread(receiveMessageFromBluetoothLoop);
            this.t1.Start();

            this.t2 = null;

            this.socketForServer = null;
            this.networkStream = null;
            this.streamReader = null;
            this.streamWriter = null;

            this.deviceConnected = "";
}
```

De realçar a variável *devices* que servirá, mais adiante, para se poder saber quais os dispositivos detectados na área que estão com o bluetooth activado.

O primeiro método que se vai apresentar serve para obter o nome de um dado dispositivo. É o método *getBluetoothDevice*.

```
public BluetoothDeviceInfo getBluetoothDevice(String bluetoothDevice)
{
            foreach (BluetoothDeviceInfo device in this.devices)
            {
                if (device.DeviceName == bluetoothDevice)
                {
                    return device;
                }
            }

            return null;
}
```

Através de uma *String* passada como parâmetro, vai-se procurar, entre os dispositivos ligados que estão na área de alcance, aquele que contém um nome igual a essa *String*. Encontrado o dispositivo pretendido, ele é devolvido sob o tipo de um *BluetoothDeviceInfo*. Este tipo pertence à biblioteca 32feet.Net e serve para recolher um variado número de informações acerca de um dispositivo desde o seu nome, fabricante, endereço físico, data da última ligação estabelecida, entre outros. Este método é utilizado algumas vezes nesta classe devido à necessidade em efectuar redirecionamentos de mensagens para um determinado dispositivo.

Na ligação entre o PDA (*BluetoothAppClient*) e *BluetoothAppServer* existem quatro métodos que são responsáveis pelas operações necessárias, sendo eles:

```
public void sendMessageToBluetooth(BluetoothDeviceInfo device, String msg);

private void sendMessageToBluetoothAux(BluetoothDeviceInfo device, int
NumRetries, byte[] Buffer, int BufferLen);

public String receiveMessageFromBluetooth(int BufferLen);

private void receiveMessageFromBluetoothLoop();
```

Na ligação entre o *BluetoothAppServer* e o servidor OpenSim os três métodos necessários à sua comunicação são:

```
public void connectOpenSim(String str);

public void sendMessageToOpenSim(String message);

public void receiveMessageFromOpenSim();
```

Estes dois conjuntos de métodos constituem o bloco da aplicação denominado de *BluetoothAppServer*.

Através da Figura 5 pode-se ver melhor como comunicam e em que sentido operam os métodos apresentados.

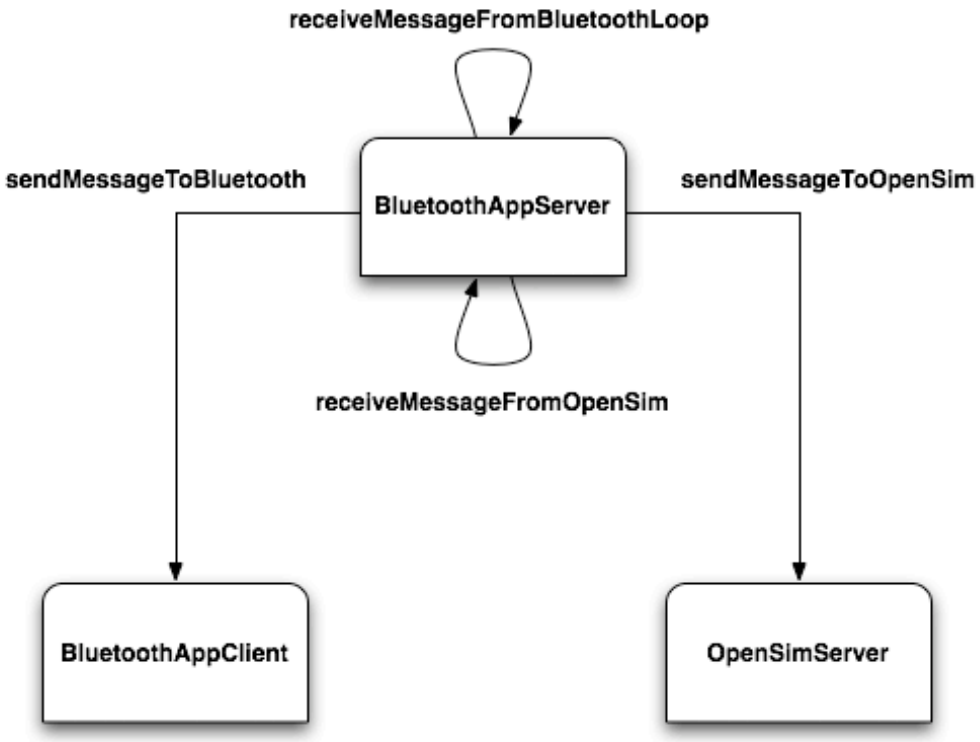

Figura 5 - Arquitectura e funcionamento dos métodos.

Os dois métodos de recepção, *receiveMessageFromBluetoothLoop* e *receiveMessageFromOpenSim*, são dois métodos controlados por duas *threads* que garantem que o *BluetoothAppServer* está sempre à espera de receber mensagens.

O método *sendMessageToBluetooth* apenas recebe o nome do dispositivo para o qual pretende enviar a mensagem e a mensagem propriamente dita, sob a forma de uma *String*. Porém, o que ele faz é apenas um *"call"* ao método operário, *sendMessageToBluetoothAux,* que realmente envia a mensagem.

```
public void sendMessageToBluetooth(BluetoothDeviceInfo device, String msg)
{
            this.sendMessageToBluetoothAux(device, MAX_TRIES,
System.Text.Encoding.Unicode.GetBytes(msg), msg.Length * 2);
}

private void sendMessageToBluetoothAux(BluetoothDeviceInfo device, int
NumRetries, byte[] Buffer, int BufferLen)
{
            int CurrentTries = 0;

            do
            {
                try
                {
```

```
                    this.btClient = new BluetoothClient();
                    this.btClient.Connect(new
BluetoothEndPoint(device.DeviceAddress, this.ServiceName));
                }
                catch (SocketException se)
                {
                    if (CurrentTries >= NumRetries)
                    {
                        throw se;
                    }
                }
                CurrentTries++;
            }
            while (this.btClient == null && CurrentTries < NumRetries);

            if (this.btClient == null)
            {
                Console.WriteLine("Error establishing conctact");
            }

            Stream stream = null;

            try
            {
                stream = this.btClient.GetStream();
                stream.Write(Buffer, 0, BufferLen);
            }
            catch
            {
                Console.WriteLine("Error sending to Client");
            }
            finally
            {
                if (stream != null)
                {
                    stream.Close();
                }
                if (this.btClient != null)
                {
                    this.btClient.Close();
                }
            }
}
```

A verdadeira função deste método operário é conectar-se realmente ao cliente de bluetooth, através do endereço físico do *device* (dispositivo) para o qual se pretende enviar a mensagem. Esta conexão é feita à custa da variável anteriormente apresentada *btClient*. Depois da conexão estar estabelecida, é escrita num *Stream* a mensagem que se quer que seja enviada. É um processo relativamente simples.

O método *receiveMessageFromBluetooth* trata da recepção de mensagens vindas do dispositivo móvel para o *BluetoothAppServer* mas é auxiliado por um método importante que é o *receiveMessageFromBluetoothLoop.*

```
private void receiveMessageFromBluetoothLoop()
{
            String strReceived;

            this.btListener = new BluetoothListener(this.ServiceName);
            this.btListener.Start();

            strReceived =
this.receiveMessageFromBluetooth(MAX_MESSAGE_SIZE);

            while (this.listening)
            {
                if (strReceived != "")
                {
                    strReceived =
this.receiveMessageFromBluetooth(MAX_MESSAGE_SIZE);
                }
            }
}
```

Através da variável *btListener*, é posta uma *thread* à escuta de ligações por parte de clientes. Quando chega uma *String*, a mesma é posta na variável *strReceived*.

Em seguida, apresenta-se o código do método que realmente faz a recepção da mensagem, o *receiveMessageFromBluetooth*.

```
public String receiveMessageFromBluetooth(int BufferLen)
{
            int bytesRead = 0;
            System.IO.Stream stream = null;
            byte[] buffer = new byte[MAX_MESSAGE_SIZE];
            String messageReceivedAux = "";

            try
            {
                this.btClient = btListener.AcceptBluetoothClient();
                stream = this.btClient.GetStream();
                bytesRead = stream.Read(buffer, 0, BufferLen);
                messageReceivedAux =
System.Text.Encoding.Unicode.GetString(buffer, 0, bytesRead);

                if (messageReceivedAux.StartsWith("device"))
                {
                    String[] strArray = new String[250];
                    String connectionData;

                    strArray = messageReceivedAux.Split(' ');

                    connectionData = strArray[2] + " " + strArray[3];

                    Console.WriteLine("Sending the login - " +
connectionData + " - to OpenSim server");

                    this.connectOpenSim(connectionData);

                    this.deviceConnected = strArray[1];
```

```
                    Console.WriteLine("The device - " + deviceConnected + "
- is connected");
                }
                else
                {
                    this.sendMessageToOpenSim(messageReceivedAux);
                    Console.WriteLine(this.deviceConnected + " > " +
messageReceivedAux);
                }
            }
            catch
            {
                if (listening)
                {
                    Console.WriteLine("Error listening to incoming
message");
                }
            }
            finally
            {
                if (stream != null)
                {
                    stream.Close();
                }
                if (this.btClient != null)
                {
                    this.btClient.Close();
                }
            }

            return str;
}
```

Sempre que uma mensagem chega ao *BluetoothAppServer*, ou seja, sempre que é escrita uma mensagem no *Stream* de comunicação por bluetooth, ele detecta se é a primeira vez que o dispositivo móvel se está a ligar a ele e, caso seja, é feita a conexão ao servidor OpenSim e é enviado o login preenchido anteriormente no PDA (username e password) para que ele se possa ligar ao mundo virtual do Open Simulator. Se não for a primeira mensagem que é enviada, é sinal que ele já está ligado e aí a mensagem é correctamente encaminhada através de métodos que serão explicados em seguida.

Como foi visto, estes quatro métodos apresentados dizem respeito ao tratamento de mensagens (envio e recepção) entre o *BluetoothAppServer* e o dispositivo móvel/PDA, ou seja, o *BluetoothAppClient*. Agora, será apresentada a outra função, também importante, do *BluetoothAppServer* - a comunicação por *Sockets* com o servidor OpenSim. Sem esta parte, as mensagens escritas no PDA nunca chegariam ao visualizador do OpenSimulator, e vice-versa.

Anteriormente apresentaram-se os três métodos que compõem a comunicação entre *BluetoothAppServer* e o servidor OpenSim, sendo eles: *connectOpenSim*, *sendMessageToOpenSim* e, por fim, o *receiveMessageFromOpenSim*.

O primeiro serve para, como o nome o indica, estabelecer a comunicação com o servidor OpenSim. Como já foi várias vezes referido, esta comunicação é estabelecida através de um protocolo de *Sockets*. Assim sendo, é preciso fazer a criação de um *Socket* e ligá-lo ao respectivo endereço IP/Porta em que está o servidor. Se este passo for feito com sucesso, em seguida, vai ser posta a executar a *thread* igualmente mencionada acima que é responsável por receber as mensagens do OpenSim.

```
public void connectOpenSim(String str)
{
            try
            {
                this.socketForServer = new TcpClient("193.136.19.112",
9004);
                Console.WriteLine("Connection to OpenSim server
established");

                this.t2 = new
System.Threading.Thread(receiveMessageFromOpenSim);
                this.t2.Start();
            }
            catch
            {
                Console.WriteLine("Failed to connect to OpenSim server");
                return;
            }

            this.networkStream = socketForServer.GetStream();
            this.streamReader = new System.IO.StreamReader(networkStream);
            this.streamWriter = new System.IO.StreamWriter(networkStream);

            this.streamWriter.WriteLine(str);

            Console.WriteLine("Login - " + str + " - was sended to OpenSim
server");

            this.streamWriter.Flush();
}
```

Para além disto, os *Streams* de comunicação são também inicializados e é enviada uma mensagem de teste para o servidor OpenSim que contém o login do utilizador que se quer ligar.

O método seguinte, *sendMessageToOpenSim*, é o responsável por enviar uma mensagem para o servidor OpenSim. No fundo, esta operação já é feita uma primeira vez no método anterior, *connectOpenSim*. Ainda assim, a criação do método a seguir apresentado facilita a (re)utilização e leitura de código.

```
public void sendMessageToOpenSim(String message)
{
            this.streamWriter.WriteLine(message);
            this.streamWriter.Flush();
}
```

Como se pode observar no método anterior, a operação feita apenas consiste em passar a *String* recebida para o *streamWriter*.

No método seguinte, *receiveMessageFromOpenSim*, é onde é feita, efectivamente, a recepção de uma mensagem vinda do servidor OpenSim. Não esquecer que este método foi aquele que ficou associado à *thread* aquando da execução da conexão ao servidor. Assim, é preciso ter em conta que este método está sempre a correr.

Neste contexto, se o *Socket* estiver ligado, este método está constantemente à espera que "alguém" escreva no *streamReader*. Caso seja escrita uma mensagem a sua função é reencaminhar correctamente a mensagem para o dispositivo que a deve receber, através do método apresentado anteriormente - *sendMessageToBluetooth*.

De realçar que ele sabe para qual o dispositivo que deverá enviar a mensagem através da variável global *deviceConnected*. A intenção desta variável é conter sempre o nome do dispositivo que está ligado ao servidor OpenSim. Ela é preenchida assim que passa pela primeira vez, pelo *BluetoothAppServer*, o login com o nome de utilizador e a password que vão em direcção ao OpenSim. Nesta fase, é filtrado o dispositivo que enviou esta mensagem e é preenchida a variável global *deviceConnected*.

```
public void receiveMessageFromOpenSim()
{
            try
            {
                if (this.socketForServer.Connected)
                {
                    while (true)
                    {
                        String msgReceived;

                        msgReceived = this.streamReader.ReadLine();

                        Console.WriteLine("OpenSim > " + msgReceived);

                        if (msgReceived.Equals("login successful"))
                        {
                            String msg = "";

                            msg = msgReceived + " " + this.deviceConnected;


this.sendMessageToBluetooth(this.getBluetoothDevice(this.deviceConnected),
msg);
                        }
```

```
                        else
                        {

this.sendMessageToBluetooth(this.getBluetoothDevice(this.deviceConnected),
msgReceived);
                        }
                    }
                }
            }
            catch
            {
                Console.WriteLine("Exception reading from OpenSim server");
            }
}
```

Assim termina a apresentação e a sucinta explicação do código que compõem o bloco *BluetoothAppServer*. O bloco que, como já foi mencionado, é responsável por duas grandes tarefas:

- Receber e enviar mensagens para o dispositivo móvel/PDA. Na prática, tem de receber e enviar mensagens para o bloco da aplicação que é o *BluetoothAppClient*.
- Receber e enviar mensagens para o servidor Open Simulator.

Com isto consegue-se garantir o correcto fluxo de mensagens a circularem entre o utilizador no PDA e o utilizador que joga no visualizador do Open Simulator.

Em seguida, será apresentada a outra parte desta aplicação, que é aquela que é executada no PDA e que também tem as suas tarefas importantes.

BluetoothAppClient

Agora é apresentada a parte da aplicação que interage com o utilizador, que envia e que lhe mostra "resultados". Esta aplicação mais pequena é constituída por três classes importantes: *Client*, *Form1* e *Form2*.

A classe *Client* é responsável pelas tarefas de negócio. Neste sentido, esta classe vai conter os métodos responsáveis por receber e enviar as mensagens desejadas, tal como já acontecia na parte do *BluetoothAppServer*. Assim, a classe *Client* não é mais do que uma réplica de alguns dos métodos que são usados na classe *Server* do *BluetoothAppServer*, sendo eles:

```
public void sendMessageToBluetooth(BluetoothDeviceInfo device, String msg);

private void sendMessageToBluetoothAux(BluetoothDeviceInfo device, int
NumRetries, byte[] Buffer, int BufferLen);

public String receiveMessageFromBluetooth(int BufferLen);

private void receiveMessageFromBluetoothLoop();
```

Por esse motivo, não serão aqui explicados estes quatro métodos que constituem a classe em causa. Eles são cópias integrais dos mesmos métodos usados no *Server* do *BluetoothAppServer*. É natural que assim aconteca, visto que o meio de comunicação é o mesmo e pretende-se que haja troca de mensagens nos dois sentidos. A única diferença neste caso é a parte que interage com o utilizador, ou seja, a camada de apresentação da aplicação.

Neste sentido, aparecem as outras duas classes: *Form1* e *Form2*. Na realidade estas duas classes são apenas dois *Forms* em formato de janelas, para o dispositivo móvel, para facilitar a interacção e introdução de informação por parte do utilizador.

O primeiro, o *Form1*, apresenta-se na Figura 6 :

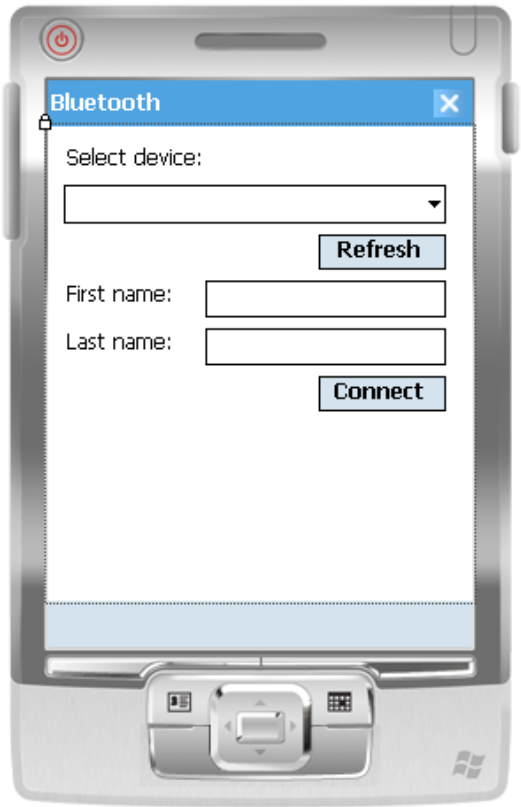

Figura 6 - Form de Login - Form1.

Este *Form* é responsável por apresentar ao utilizador a possibilidade de introdução dos seus dados pessoais de acesso à conta no servidor Open Simulator. Isto depois de ter a possiblidade de poder escolher a que servidor se deseja connectar por Bluetooth, a partir de uma lista de dispositivos ligados na sua área de alcance. O código respeitante a esta classe não serve os propósitos deste documento, sendo apenas código relativo à parte de apresentação, não contendo qualquer ligação ao tema do projecto. É apenas código necessário para que os dispositivos apareçam correctamente listados e para que os dados pessoais sejam correctamente reencaminhados.

O outro form, *Form2*, pode ser observado na Figura 7 :

Figura 7 - Form para visualização da conversação - Form2.

Este *Form* serve para que o utilizador possa escrever mensagens e direccioná-las para o servidor ao qual está ligado e serve para mostrar o log da conversação que ele, que está ligado por Bluetooth, mantém com o utilizador que está efectivamente a jogar no mundo virtual do Open Simulator. A não apresentação do código relativo a este *Form* segue os mesmos propósitos do anterior.

Capítulo 6 - Conclusão

Após a conclusão desta pequena aplicação o objectivo inicial que incluia a possibilidade de troca de mensagens nos dois sentidos entre um utilizador com um PDA e um outro utilizador a jogar no mundo virtual do OpenSimulator foi atingido.

De salientar o uso da biblioteca 32 feet.Net, deveras um óptimo impulsionador do projecto. É uma biblioteca bem construída e com muitos métodos e classes úteis à disposição de quem quiser desenvolver código .Net para trabalhar com redes bluetooth, ou infra-vermelhos.

Há que mencionar também o facto de que a fase em que a aplicação ficou neste momento representa ainda uma fase de pouca maturação. Isto é um início, uma base, como foi dito anteriormente, de algo que pode crescer a partir daqui. A intenção deste projecto é que se consiga que mensagens sejam automaticamente enviadas do servidor OpenSimulator para o PDA que está *"logado"* nele no momento. As possibilidades de aplicação deste tipo de ideias no mundo real são vastas.

Pessoalmente também posso acrescentar que gostei imenso de desenvolver o projecto pois trabalhei com tecnologia com a qual ainda não tinha trabalhado até à data, o que me fez aprender algo de novo.