Università Politecnica delle Marche

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Tesi di Laurea

Progettazione e realizzazione di un'app Xamarin per l'implementazione di un gioco sociale tramite bluetooth

Design and realization of a Xamarin app for the implementation of a social game through bluetooth

Relatore

Prof. Domenico Ursino

Candidato

Francesco Tontarelli

Anno accademico 2018-2019

Indice

Introduzione 3

1 Le app ibride 5
1.1 Quando e perché scegliere app ibride 5
1.1.1 Vantaggi e svantaggi 5
1.2 Framework 6
1.2.1 PhoneGap 7
1.2.2 Apache Cordova 8
1.2.3 Appcelerator Titanium 10
1.2.4 Sencha Touch 11
1.2.5 Kendo UI 12
1.2.6 AppGyver 12
1.2.7 Ionic 14
1.2.8 Xamarin 15
1.2.9 React Native 15

2 Xamarin 17
2.1 Storia 17
2.2 C# 19
2.3 Xamarin.forms 21
2.4 Strumenti per lo sviluppo 28

3 La gestione del bluetooth nelle app 31
3.1 Generalità 31
3.2 Scelta di un protocollo di trasporto 34
3.2.1 RFCOMM 38
3.2.2 L2CAP 38
3.2.3 ACL 39
3.2.4 SCO 39
3.3 Porte 40
3.4 Servizio ID 42
3.5 Comunicare via socket 43
3.6 Implementazione del Bluetooth nell'applicazione TrisBluetooth 44

4 Analisi dei requisiti e progettazione 47
4.1 Mobile Gaming 47
4.1.1 Importanza del Mobile Gaming 47
4.1.2 Connettività nel Mobile Gaming 48
4.2 Requisiti 49
4.2.1 Autorizzazioni 49
4.2.2 Ricerca 51
4.2.3 Istruzioni di utilizzo 51
4.2.4 Griglia di gioco 51
4.2.5 User Experience Design 51
4.3 Progettazione 53
4.3.1 Diagramma dei casi d'uso 53
4.3.2 Diagramma delle attività 54
4.3.3 Mockup 55

5 Implementazione 65
5.1 Implementazione di `SplashPage.xaml.cs` 65
5.1.1 Implementazione di `SplashPage.xaml` 66
5.2 Implementazione di `MainActivity.cs` 67
5.3 Implementazione di `MainPage.xaml.cs` 75
5.3.1 Implementazione di `MainPage.xaml` 76
5.4 Implementazione di `App.xaml.cs` 78
5.4.1 Implementazione di `App.xaml` 78
5.5 Implementazione di `Gioca.xaml.cs` 79
5.5.1 Implementazione di `Gioca.xaml` 85

6 Manuale utente 89
6.1 Avvio dell'applicazione 89
6.2 Ricerca dei dispositivi Bluetooth limitrofi 89
6.3 Avvio di una partita 90

7 Discussione in merito al lavoro svolto 99
7.1 Competenze maturate durante la realizzazione dell'applicazione TrisBluetooth 99
7.2 Analisi SWOT 100
7.2.1 Punti di forza 101
7.2.2 Punti di debolezza 101
7.2.3 Opportunità 102
7.2.4 Minacce 102

8 Conclusioni e uno sguardo al futuro 103

Riferimenti bibliografici 105

Ringraziamenti 107

Elenco delle figure

1.1 Logo di PhoneGap . . . 7
1.2 Logo di Apache Cordova . . . 8
1.3 Logo di Appcelerator Titanium . . . 11
1.4 Logo di Sencha Touch . . . 11
1.5 Logo di Kendo UI . . . 12
1.6 Logo di AppGyver . . . 13
1.7 Logo di Ionic . . . 14
1.8 Logo di Xamarin . . . 15
1.9 Logo di React Native . . . 16

2.1 Storia di Xamarin . . . 17
2.2 Componenti di Xamari e il loro legame con le SDK native . . . 19
2.3 Alcune delle librerie sviluppate da Xamarin . . . 20
2.4 Compilazione nelle diverse piattaforme . . . 21
2.5 Approccio tradizionale di Xamarin . . . 22
2.6 Approccio di Xamarin.forms . . . 22
2.7 Architettura di un'applicazione multipiattaforma . . . 24
2.8 Approccio SAP . . . 25
2.9 Approccio PCL . . . 26
2.10 Controlli in Xamarin.forms . . . 27
2.11 Layout in Xamarin.forms . . . 27
2.12 Pagine in Xamarin.forms . . . 27
2.13 Logo di Visual Studio . . . 28
2.14 Creazione di nuovo progetto in Visual Studio 2017 . . . 29
2.15 Finestra di dialogo "Nuovo progetto" in Visual Studio 2017 . . . 29
2.16 Esplora soluzioni in Visual Studio 2017 . . . 30
2.17 Toolbar per il testing in Visual Studio 2017 . . . 30

3.1 Icona del Bluetooth . . . 31
3.2 Connessione internet e Bluetooth a confronto . . . 32
3.3 Panoramica dei protocolli di comunicazione . . . 35
3.4 Due piconet che costituiscono uno scatternet . . . 36
3.5 Schema time-division duplex . . . 37

3.6 Pacchetti multislot 37
3.7 Protocolli Bluetooth 41
3.8 Confronto tra connessione tradizionale e con SDP 42
3.9 Utilizzo dei socket in una comunicazione client-server 43

4.1 Download di applicazioni di gioco nel secondo trimestre del 2019 48
4.2 Spesa dei consumatori nelle app di gioco nel secondo trimestre del 2019 48
4.3 "The Elements of UX" di Jesse James Garrett 52
4.4 Logo TrisBluetooth 52
4.5 Diagramma dei casi d'uso dell'applicazione TrisBluetooth 54
4.6 Diagramma delle attività dell'applicazione TrisBluetooth 55
4.7 Mockup di livello 0 della Splash page dell'applicazione TrisBluetooth. 56
4.8 Mockup di livello 1 della Splash page dell'applicazione TrisBluetooth. 57
4.9 Mockup di livello 0 della richiesta di autorizzazione Bluetooth 57
4.10 Mockup di livello 1 della richiesta di autorizzazione Bluetooth 58
4.11 Mockup di livello 1 delle informazioni della pagina di Gestione Bluetooth 58
4.12 Mockup di livello 0 delle lista dei dispositivi Bluetooth limitrofi della pagina di Gestione Bluetooth dell'applicazione TrisBluetooth ... 59
4.13 Mockup di livello 1 delle lista dei dispositivi Bluetooth limitrofi della pagina di Gestione Bluetooth 59
4.14 Mockup di livello 0 della pagina di gioco 60
4.15 Mockup di livello 1 della pagina di gioco 60
4.16 Mockup di livello 1 delle informazioni della pagina di gioco 61
4.17 Mockup di livello 0 della pagina di gioco durante una partita 61
4.18 Mockup di livello 1 della pagina di gioco durante una partita 62
4.19 Mockup di livello 0 della pagina di gioco a fine partita 62
4.20 Mockup di livello 1 della pagina di gioco a fine partita 63

5.1 Pushing di una nuova pagina dallo stack di navigazione 66
5.2 Popping di una nuova pagina dallo stack di navigazione 66
5.3 Funzionamento del `MessagingCenter` 68

6.1 Schermata home con l'icona dell'applicazione TrisBluetooth 90
6.2 Autorizzazione all'acquisizione della posizione da parte dell'applicazione TrisBluetooth 91
6.3 Animazione dell'icona dell'applicazione TrisBluetooth nella Splash Page 92
6.4 Main Page dell'applicazione TrisBluetooth 92
6.5 Istruzioni per la ricerca del dispositivo dell'avversario 93
6.6 Autorizzazione all'utilizzo Bluetooth da parte dell'applicazione TrisBluetooth 93
6.7 Lista dei dispositivi rilevati 94
6.8 Selezione del dispositivo avversario in seguito alla ricerca dei dispositivi Bluetooth limitrofi 94
6.9 Pagina di gioco 95
6.10 Istruzioni di gioco 95

6.11 Il display afferma che il dispositivo ha il turno di gioco 96
6.12 Il display afferma che il dispositivo attende la mossa dell'avversario . . 96
6.13 Scritta animata in seguito alla vittoria 97
6.14 Scritta animata in seguito alla sconfitta 97
6.15 Scritta animata in seguito al pareggio 98
6.16 Pulsante per l'avvio della rivincita a fine partita 98

7.1 Schematizzazione dell'analisi SWOT 100

Elenco dei listati

2.1 gestione multipiattaforma SAP 25
3.1 Interfaccia `IBluetooth.cs` 44
3.2 La classe `Bth.cs` che implementa in Android l'interfaccia `IBluetooth.cs` 44
5.1 Il codice relativo a `SplashPage.xaml.cs` 65
5.2 Il codice relativo a `SplashPage.xaml` 66
5.3 Definizione del metodo `OnReceive()` del file `MainActivity.cs` 67
5.4 Definizione del metodo `OnCreate()` del file `MainActivity.cs` 69
5.5 Definizione del metodo `OnDestroy()` del file `MainActivity.cs` 69
5.6 Definizione del metodo `reset()` del file `MainActivity.cs` 69
5.7 Definizione del metodo `checkPermission()` del file `MainActivity.cs` 70
5.8 Definizione del metodo `registerReceivers()` del file `MainActivity.cs` 70
5.9 Definizione del metodo `setMailBoxes()` del file `MainActivity.cs` . . . 71
5.10 Definizione del metodo `AbilitaDiscoverabilty()` del file `MainActivity.cs` 71
5.11 Definizione della classe `ConnectThread` del file `MainActivity.cs` 72
5.12 Definizione della classe `AcceptThread` del file `MainActivity.cs` 73
5.13 Definizione della classe `ConnectedThread` del file `MainActivity.cs` . . 73
5.14 Definizione del costruttore del file `MainPage.xaml.cs` 75
5.15 Definizione del metodo `Cerca()` del file `MainPage.xaml.cs` 76
5.16 Definizione del metodo `MainListView_ItemTapped()` del file `MainPage.xaml.cs` 76
5.17 Definizione del metodo `Gioca()` del file `MainPage.xaml.cs` 76
5.18 Definizione del metodo `Info()` del file `MainPage.xaml.cs` 76
5.19 Il codice relativo al file `MainPage.xaml` 77
5.20 Il codice relativo al file `App.xaml.cs` 78
5.21 Il codice relativo al file App.xaml 79
5.22 Il codice relativo al costruttore del file `Gioca.xaml.cs` 79
5.23 Il codice relativo al metodo `Info()` del file `Gioca.xaml.cs` 80
5.24 Il codice relativo al metodo `Client()` del file `Gioca.xaml.cs` 81
5.25 Il codice relativo al metodo `Server()` del file `Gioca.xaml.cs` 81
5.26 Il codice relativo al metodo `Rivincita()` del file `Gioca.xaml.cs` 81

5.27 Il codice del metodo `ButtonClicked()` del file `Gioca.xaml.cs` 82
5.28 Il codice relativo al metodo `reset()` del file `Gioca.xaml.cs` 82
5.29 Il codice relativo al metodo `ButtonEnable()` del file `Gioca.xaml.cs` . 83
5.30 Il codice relativo al metodo `xoisEnable()` del file `Gioca.xaml.cs` ... 83
5.31 Il codice relativo al metodo `ButtonClick()` del file `Gioca.xaml.cs` .. 83
5.32 Il codice relativo al metodo `victory()` del file `Gioca.xaml.cs` 84
5.33 Il codice relativo al metodo `draw()` del file `Gioca.xaml.cs` 84
5.34 Il codice relativo al metodo `messageReceived()` del file `Gioca.xaml.cs` 84
5.35 Il codice relativo al metodo `OnDisappearing()` del file `Gioca.xaml.cs` 85
5.36 Il codice relativo al file `Gioca.xaml` 86

Introduzione

In questo elaborato di tesi vengono presentate la progettazione e la realizzazione del progetto TrisBluetooth, un'applicazione che implementa un gioco sociale che utilizza la connessione Bluetooth tra dispositivi.

L'idea di progettare ed implementare questa applicazione nasce dalla volontà di offrire agli usufruitori la possibilità di divertirsi con giochi di società, nel caso specifico tris, anche in luoghi dove la connettività è assente oppure non permessa.

Gli smartphone e i tablet sono ormai in grado di compiere operazioni complesse che, fino a qualche anno fa, erano ritenute impensabili; i dispositivi sono diventati sempre più potenti, efficienti, innovativi ed alla portata di tutti.

Il mondo delle applicazioni mobili ha subito un'enorme espansione che non accenna ad arrestarsi, in quanto il mercato è in continua evoluzione. Non si parla soltanto di applicazioni ludiche, ma di software applicabili a diversi ambiti, come la vita privata, la sanità, l'amministrazione pubblica e aziendale etc.

Dal 2007, l'anno in cui venne distribuito l'iPhone, ad oggi, lo sviluppo dei dispositivi mobili è in crescita esponenziale; gli smartphone più basilari possono annoverare, tra le proprie caratteristiche, la telefonia, il bluetooth, il microfono, la fotocamera, la multimedialità, la grafica 3D, il touchscreen, notifiche ed allarmi, il networking e la presenza di svariati sensori (accelerometro, giroscopio, bussola, GPS) etc; che consentono innumerevoli funzionalità. L'applicazione TrisBluetooth funziona proprio grazie alla tecnologia Bluetooth ormai implementata in tutti i dispositivi.

Le due aziende che attualmente dominano il mercato degli smartphone sono Apple e Google, con i loro sistemi operativi iOS e Android.

Android, essendo open source, è riuscito ad essere presente su terminali di diversi marchi; proprio per questo Android ha superato iOS nelle vendite dei dispositivi mobili.

Ovviamente, colui il quale trae più vantaggio da questa situazione è l'utente, in quanto ha a disposizione una considerevole quantità di dispositivi con diverse caratteristiche e differenti fasce di prezzo.

Tra i diversi settori riguardante lo sviluppo mobile, quello che risulta più pertinente all'applicazione TrisBluetooth è il Mobile Gaming.

Il Mobile Gaming è un settore in pieno sviluppo ed espansione a dimostrazione di questo ci sono molte indagini di mercato.

La presente tesi intende mostrare elementi di analisi e progettazione di un'applicazione e, conseguentemente, la loro implementazione e il loro testing, mettendo in causa gli aspetti e le problematiche alla base del lavoro di un ingegnere informatico. L'applicazione è concepita in modo da essere fruibile da chiunque possieda uno smartphone con sistema operativo Android.

Si è deciso di utilizzare il framework Xamarin poiché i costi per la progettazione e lo sviluppo di un'applicazione sono spesso i punti critici che un'azienda valuta; proprio per questo motivo la possibilità di riutilizzare codice per le diverse piattaforme, seppur sacrificando le prestazioni del prodotto, possono risultare una valida idea per limitare il budget richiesto.

La tesi è strutturata come di seguito specificato:

- *Capitolo 1*: verranno introdotte le applicazioni ibride descrivendo i vantaggi e gli svantaggi nonché i principali framework per la loro implementazione ovvero PhoneGap, Apache Cordova, Appcelerator Titanium, Sencha Touch, Kendo UI, AppGyver, Ionic, Xamarin, React Native.
- *Capitolo 2*: verranno ripercorse le principali tappe della storia di Xamarin per poi illustrare il linguaggio di programmazione C# adottato e gli strumenti per lo sviluppo adottati per la progettazione e l'implementazione dell'applicazione TrisBluetooth.
- *Capitolo 3*: verrà introdotto il Bluetooth e la modalità tramite cui è possibile implementare un'applicazione che utilizza la comunicazione Bluetooth, analizzando i principali protocolli di trasporto e le tecniche che utilizza per consentire una trasmissione sicura dei dati.
- *Capitolo 4*: verrà illustrato il mondo del Mobile Gaming, mostrando l'importanza nel mercato delle applicazioni per poi, dopo aver analizzato i requisiti necessari per lo sviluppo dell'applicazione TrisBluetooth, presentare i diagrammi delle attività e dei casi d'uso e i mockup di livello 0 e di livello 1.
- *Capitolo 5*: verrà illustrato tutto il progetto, analizzando ogni file che lo compone e spiegando passo passo il codice dalla parte cross platform a quella specifica per la piattaforma Android, non tralasciando i file XAML.
- *Capitolo 6*: verranno illustrati gli screenshot, analizzando tutte le pagine che compongono l'applicazione TrisBluetooth per facilitare l'utilizzo all'utente, riportando tutte le operazioni che l'usufruitore può effettuare.
- *Capitolo 7*: verrà ripercorso tutto il periodo di progettazione, evidenziando le tappe essenziali del lavoro svolto.
- *Capitolo 8*: verranno tratte alcune conclusioni in merito al lavoro svolto; successivamente, si riporteranno alcune ipotesi sui possibili sviluppi futuri.

1

Le app ibride

In questo capitolo verranno introdotte le applicazioni ibride descrivendo i vantaggi e gli svantaggi nonchè i principali framework per la loro implementazione.

1.1 Quando e perché scegliere app ibride

La progettazione e lo sviluppo di applicazioni mobile è un compito arduo; se si desidera sviluppare un'applicazione destinata ad una sola piattaforma il lavoro potrebbe risultare più semplice rispetto allo sviluppo multi-piattaforma. Tuttavia, l'obbiettivo di coloro che sviluppano applicazioni è distribuirle nel maggior numero possibile di app store per massimizzare le entrate. Per fare questo, gli sviluppatori hanno bisogno di utilizzare diversi linguaggi di programmazione, diversi SDK e tool, ad esempio, Objective-C / Swift per iOS e Java per Android. Si ha, anche, bisogno di gestire diverse basi di codice con funzionalità simili, ma implementate utilizzando diversi linguaggi di programmazione. È difficile massimizzare la riutilizzabilità del codice e ridurre le duplicazioni di esso su basi di codice differenti, anche per i più grandi esperti del settore; proprio da questa esigenza nasce lo sviluppo cross-platform. Questo nuovo tipo di applicazioni mobile si chiamano applicazioni ibride o anche applicazioni multipiattaforma.

1.1.1 Vantaggi e svantaggi

Rispetto alle applicazioni native, le applicazioni ibride hanno dei vantaggi e degli svantaggi. Il principale vantaggio è dovuto al fatto che gli sviluppatori possono utilizzare competenze di sviluppo web per creare applicazioni ibride e utilizzare solo una base di codice per diverse piattaforme. Sfruttando le tecniche di responsive web design, le applicazioni ibride possono facilmente adattarsi a diverse risoluzioni dello schermo. Lo svantaggio principale è dovuto alle minori prestazioni che le applicazioni ibride offrono rispetto alle applicazioni native, ciò è dovuto al fatto che le applicazioni ibride sono in esecuzione all'interno di un browser. Alcuni tipi di applicazioni, come i giochi o le applicazioni che si basano su funzionalità native complicate, non possono essere sviluppate come applicazioni ibride; ma molte altre

possono essere sviluppate come applicazioni multipiattaforma; ad esempio, la tecnologia ibrida è adatta per le applicazioni content-centric come, ad esempio, i forum online. Queste applicazioni si comportano come applicazioni web tradizionali con interazioni utente limitate.

Prima di prendere la decisione se sviluppare il progetto come applicazione nativa o applicazione ibrida, il team di sviluppo ha bisogno di capire la natura dell'applicazione. Un altro fattore importante da considerare sono le conoscenze del team di sviluppo. La maggior parte di aziende che sviluppano applicazioni possono avere bisogno di assumere sia sviluppatori iOS che sviluppatori Android, in modo da poter sviluppare applicazioni native spendibili in più piattaforme nel mercato. Per le applicazioni ibride l'azienda non si deve porre questo problema, poiché gli sviluppatori solo front-end sono sufficienti. È, generalmente, più semplice assumere sviluppatori front-end, piuttosto che sviluppatori Android oppure sviluppatori Swift/Objective-C. Inoltre, va notato, che le applicazioni ibride sono più rapide e meno costose da sviluppare rispetto a quelle native e, allo stesso tempo, offrono più possibilità di una web app. Infine, occorre porre attenzione al fatto che la manutenzione di un'applicazione ibrida è più facile rispetto alla manutenzione delle applicazioni native poiché esiste una sola versione da aggiornare per le diverse piattaforme.

1.2 Framework

I framework di sviluppo mobile offrono agli sviluppatori una base generica di funzionalità che può essere modificata a seconda del software specifico dell'applicazione mobile. I framework possono essere suddivisi in tre categorie: framework nativi per lo sviluppo specifico della piattaforma, framework per app Web mobili e framework per applicazioni ibride, che combinano le funzionalità dei framework delle app Web native e mobili. I framework nativi consentono agli sviluppatori di utilizzare tutte le funzionalità native di una piattaforma mobile (iOS, Android, Windows, Galaxy). Tuttavia, essi sono specifici della piattaforma e possono essere usati per creare app solo per una specifica piattaforma mobile. I framework ibridi non hanno la capacità di utilizzare tutte le funzioni API native, ma offrono comunque agli sviluppatori strumenti per accedere alla fotocamera, alla geolocalizzazione e al sistema di notifica di un telefono attraverso piattaforme di sviluppo mobile. I framework di sviluppo mobile devono:

- fornire librerie software che descrivono la struttura di base di un'applicazione mobile;
- possedere comportamenti predefiniti di un'applicazione;
- centralizzare il codice di base;
- possedere componenti strutturali bloccati ed espandibili;
- supportare i linguaggi di codifica HTML5, JavaScript, CSS o nativi.

I framework possono essere classificati in due categorie, ovvero:

- *Wrapper*: questi rappresentano i framework di prima generazione; tramite essi lo sviluppatore, senza occuparsi del linguaggio della piattaforma sottostante, può usare funzionalità come GPS, fotocamera e contatti.

- *Librerie*: i framework di seconda generazione offrono funzionalità capaci di far avvicinare al massimo il look and feel dell'app risultante a quello di un'app nativa. Le librerie, inoltre, favoriscono una semplice gestione di eventi e la manipolazione del DOM (Document Object Model).

Di seguito verranno riportati alcuni tra i principali framework presenti nel mondo della programmazione mobile.

1.2.1 PhoneGap

PhoneGap (figura 1.1) è un framework di sviluppo open source per creare applicazioni mobili rapide e facili, utilizzando HTML, CSS e Javascript. PhoneGap è un framework multipiattaforma affidabile, compatibile con diverse piattaforme di sviluppo di applicazioni mobili, tra cui iOS, Blackberry, Android, WebOS, Symbian, Tizen e Bada.

Figura 1.1. Logo di PhoneGap

PhoneGap fornisce agli sviluppatori di app mobili l'accesso alle API native in modo che l'app possa utilizzare la fotocamera, la geolocalizzazione, l'accelerometro, i contatti e tutte le altre funzionalità native. Esso aiuta gli utenti ad acquisire un'esperienza nell'utilizzo di app native che è di gran lunga superiore rispetto alle applicazioni web. Se si sta provando qualcosa di nuovo, allora è essenziale avere un supporto adeguato; proprio per questo motivo, la comunità di sviluppo PhoneGap è una delle più grandi comunità e sta crescendo costantemente. Grazie alle persone di questa comunità è facile ricevere una risposta per la maggior parte dei problemi: ci sono molti esempi e così tanti membri esperti che forniscono un grande supporto. Le app PhoneGap sono costruite usando alcuni dei linguaggi semplici, come CSS, HTML e JavaScript. Non è necessario imparare nuovi linguaggi di programmazione quando si ha intenzione di iniziare a sviluppare un'app mobile con PhoneGap: è possibile utilizzare i linguaggi esistenti e scrivere programmi con facilità. PhoneGap ha un robusto supporto back-end che accelera il processo di sviluppo e riduce gli sforzi dello sviluppatore. PhoneGap offre, anche, una guida per gli sviluppatori alle prime armi per comprendere facilmente l'intero progetto. Con le librerie UI presenti in PhoneGap, l'interfaccia utente delle app e delle pagine web è migliorata nel tempo. Questo aiuta a fornire una migliore esperienza di visualizzazione delle pagine web. Con l'aiuto di AEM mobile, gli sviluppatori sono in grado di abbinare tutte le attività delle loro app da un'unica posizione. La sincronizzazione dei contenuti di PhoneGap aiuta a recuperare il contenuto dell'app e lo memorizza nella cache. Con

l'aiuto di questo AEM si possono testare e visualizzare in anteprima le modifiche apportate all'app. PhoneGap è compatibile con diversi plugin Cordova e PhoneGap. Gli sviluppatori possono estendere le funzionalità native sviluppando app su PhoneGap.

Il più grande svantaggio di PhoneGap è che non supporta i plugin con hook, e questo può ostacolare la funzionalità. Ci sono degli hook in WordPress che aiutano gli sviluppatori a legare il loro codice con il codice base, i temi e i plugin in Word Press. PhoneGap è una piattaforma progettata per sviluppare semplici applicazioni, non è adeguata allo sviluppo di app di gioco perché si bloccano e hanno problemi di rilascio quando ricevono notifiche. PhoneGap richiede iOS SDK ed è impossibile sviluppare un'app senza SDK. Il download di questi SDK richiede un Mac. Non si è in grado di sviluppare un'app iOS senza Mac. Le seguenti app sono sviluppate sulla piattaforma framework PhoneGap e offrono un'esperienza utente lodevole per gli utenti: Wikipedia (questa app gratuita è un'enciclopedia progettata con il framework PhoneGap; l'app funziona come le applicazioni native), DHS program (Demographic and Health Survey; l'app è stata sviluppata da Adobe e fornisce i dati relativi a sondaggi), Paylution (aiuta a gestire i pagamenti internazionali con facilità; la tecnologia hyperWallet è incorporata in questa app per effettuare i pagamenti), TripCase (questa è un'app mobile intelligente che aiuterà gli utenti durante i loro viaggi. PhoneGap è alimentato da Apache Cardova e ora fa parte di Adobe. Dopo l'acquisto di PhoneGap da parte di Adobe Systems nel 2011, è stata rilasciata una versione open source chiamata Apache Cordova, la quale tuttora resta alla base di PhoneGap.

1.2.2 Apache Cordova

Dal progetto PhoneGap nasce Apache Cordova (figura 1.2) avviato da Nitobi Software e venduto nel 2011 ad Abobe. Nitobi donò alla fondazione Apache il progetto che, inizialmente, venne chiamato Apache Callback e, successivamente, Apache Cordova.

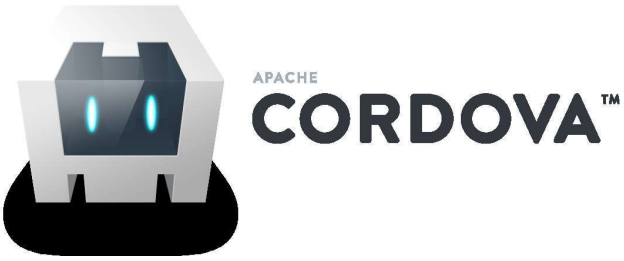

Figura 1.2. Logo di Apache Cordova

Apache Cordova è un framework di sviluppo per applicazioni mobili open source. Esso fornisce gli strumenti necessari a sviluppare software per i dispositivi mobili che utilizzano JavaScript, HTML5 e CSS3, senza che questi ultimi debbano utilizzare

API specifiche della piattaforma, come quelle di Android, iOS o Windows Phone. Nonostante Cordova sia un framework multipiattaforma, i requisiti software variano in base alla piattaforma mobile di riferimento; infatti, per sviluppare un'applicazione Android e compilarla, è possibile usare indifferentemente un PC Linux, Windows o un Mac; le cose cambiano per lo sviluppo di un'app iOS, la quale può essere compilata solo su Mac. Questo è dovuto al fatto che, per ciascuna piattaforma mobile di riferimento, occorre avere installato il relativo SDK di sviluppo.

Un'applicazione Apache Cordova è implementata come una pagina web. La pagina web può fare riferimento a file JavaScript, file CSS, immagini e altre risorse. Cordova mette a disposizione strumenti di sviluppo che consentono agli sviluppatori di costruire applicazioni utilizzando dei linguaggi specifici del Web: HTML, CSS e JavaScript. Esso supporta le seguenti piattaforme mobile: Android, iOS, Blackberry, Bada, Tizen e Windows Phone.

La componente chiave per capire come funziona Cordova è il WebView. Esso viene fornito da piattaforme native per caricare ed eseguire le pagine web. Le applicazioni Cordova in run sono all'interno delle WebView.

Apache Cordova è solo un ambiente di runtime per le applicazioni web su piattaforme native. È in grado di supportare qualsiasi tipo di pagine web. Quindi le applicazioni vengono eseguite in un wrapper specifico per ogni piattaforma, il quale utilizza le API messe a disposizione dagli ambienti specifici per accedere alle risorse hardware e software di ogni dispositivo.

Nonostante il framework si basi sul principio di unicità del codice è inevitabile che la fase ultima di deploy dell'applicazione debba essere effettuata su una piattaforma di sviluppo nativa. Una volta effettuato lo sviluppo basato su HTML, JS e CSS, l'ultima fase prevede la creazione, su X-Code per iOS e su Eclipse per Android, di un progetto che, usando le librerie native di Cordova, permetterà la fruizione e l'interoperabilità con l'applicativo scritto in linguaggio universale. Per la creazione di tali progetti quindi, non è richiesta alcuna conoscenza del linguaggio nativo, in quanto i passaggi sono illustrati nel dettaglio sul sito di Apache Cordova.

Una potente funzione di Cordova è la sua interfaccia plug-in, che permette al codice JavaScript in esecuzione in una pagina web di comunicare con i componenti nativi. Ci sono già molti plug-in disponibili nel Cordova Registry. Possiamo facilmente trovare i plugin utilizzati per scenari comuni. Con l'aiuto di plug-in, le applicazioni Cordova possono accedere alle seguenti classi di funzionalità:

- *Accelerometro*: questo plugin consente di accedere all'accelerometro del dispositivo. Quest'ultimo è un sensore di movimento che rileva il cambiamento (delta) nel movimento relativo all'orientamento corrente del dispositivo lungo gli assi x, y e z.
- *Camera*: tale plugin definisce un oggetto globale navigator.camera, il quale permette l'accesso alla fotocamera, fornendo la possibilità di salvare l'immagine scattata come file, oppure di ottenerla con codifica base 64 (codifica utilizzata per la trasmissione tramite chiamate HTTP). Esso permette, anche, alla maggior parte delle piattaforme, di accedere alle immagini già presenti nella libreria fotografica del dispositivo.
- *Capture*: questo plugin risulta più generico rispetto al plugin Camera. Esso permette l'acquisizione di file multimediali di tipo audio, immagine o video, consentendo di specificare diversi formati di acquisizione.

- *Contacts*: esso consente di accedere alle informazioni relative ai contatti, permettendo anche di definirne di nuovi o di modificare quelli esistenti.
- *Device*: tale plugin permette di accedere alle informazioni hardware e software del dispositivo in uso.
- *Events*: tale plugin dà la possibilità di gestire gli eventi che si verificano sul dispositivo. Gli eventi gestibili sono: dispositivo pronto (quando l'applicazione è completamente caricata), dispositivo in pausa (quando l'applicazione va in background), dispositivo online (quando viene rilevata la connettività), dispositivo offline (quando viene rilevata mancanza di connettività), rilevato livello critico o basso livello batteria, rilevata pressione di pulsanti hardware vari.
- *File*: questo plugin permette di interfacciarsi con il file system consentendo l'accesso in lettura e scrittura ai file e alle cartelle che risiedono sul dispositivo.
- *Geolocation*: tale plugin consente di utilizzare il sensore GPS e, quindi, di rilevare la posizione corrente del dispositivo anche quando risulta in movimento; in particolare, verranno fornite le coordinate in uscita dal sensore o, nel caso in cui queste ultime non fossero presenti, le coordinate calcolate in base alle specifiche W3C delle API di geolocalizzazione. Le coordinate vengono restituite sotto forma di latitudine e longitudine.
- *Globalization*: tale plugin permette di avere informazioni riguardo le lingue e le impostazioni internazionali correnti del dispositivo (lingua attuale, separatore delle migliaia, decimali, formato per la data, etc.).
- *Notification*: questo plugin permette di gestire le azioni di notifica, come la visualizzazione di messaggi (alert), la vibrazione e i suoni, impiegando audio e grafica nativi del dispositivo utilizzato.
- *Storage*: esso permette l'utilizzo dello storage del dispositivo basato sulla specifica W3C dei database SQL web.
- *Connection*: Tale plugin ci permette di accedere alle informazioni relative alla connettività; è possibile sapere se essa non è disponibile, se è di tipo Ethernet, Wifi, 2G, 3G o 4G.

1.2.3 Appcelerator Titanium

Titanium Appcelerator (figura 1.3) è un framework open source di sviluppo introdotto nel Dicembre 2008. Esso è prodotto da una società chiamata Appcelerator e permette di costruire applicazioni per iOS, Android e BlackBerry.

Sebbene il supporto BlackBerry esista, esso non è robusto come quello per iOS e Android. Anche Titanium fa uso di Javascript, HTML5 o CSS3. Con una pagina web è possibile modificare le proprietà CSS per modificare l'aspetto grafico degli oggetti; se si volesse modificare l'aspetto di un pulsante è possibile modificare i parametri relativi al pulsante. I concetti sono simili, ma è bene sapere che CSS non viene utilizzato affatto con Titanium. Quando si compila l'applicazione per iOS o Android, Titanium elabora il codice JavaScript e poi costruisce un progetto appropriato per la piattaforma per la quale si sta sviluppando. Per iOS questo significa che verrà creato un progetto nativo Xcode che verrà, necessariamente, compilato usando il compilatore di Apple. Un processo simile accade per le applicazioni Android; in particolare, viene creata un'applicazione nativa Java che viene compilata utilizzando il compilatore di Android. Appcelerator Titanium è una soluzione completa per

Figura 1.3. Logo di Appcelerator Titanium

creare app ibride con tutto quello che serve in un unico posto. Esso utilizza Alloy, un framework che adotta un'architettura basata sul pattern MVC e, tramite l'utilizzo di file XML e CSS-like, permette di separare la logica di business dalla User Interface e dal modello dei dati.

In un progetto Titanium Alloy, la relazione tra una view e il suo controller è data dal nome dei file che, ad esclusione dell'estensione, deve essere lo stesso per entrambi. Quindi, se abbiamo chiamato una view `example.xml`, il suo controller si chiamerà necessariamente `example.js`. I due file, però, non verranno salvati nella stessa cartella, perché Alloy prevede che view e controller abbiano una directory a loro dedicata. Titanium consente di creare applicazioni native che possono essere eseguite sia su iOS che su Android. Un'app sviluppata in Titanium può essere eseguita su entrambe le piattaforme senza cambiare una sola riga di codice. Oltre 50.000 applicazioni sono state rilasciate utilizzando la tecnologia Titanium.

1.2.4 Sencha Touch

Sencha Touch (figura 1.4) è un framework JavaScript avente architettura MVC; il suo utilizzo è particolarmente adatto alla realizzazione di applicazioni mobile ibride.

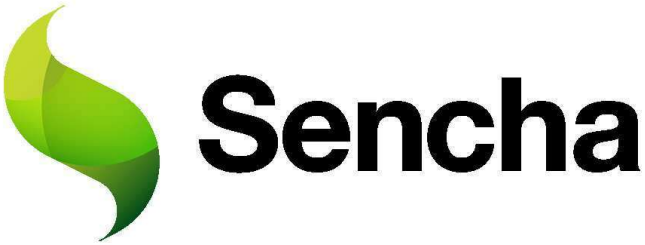

Figura 1.4. Logo di Sencha Touch

Sin dal suo lancio iniziale, Sencha Touch è diventato rapidamente un punto di riferimento per lo sviluppo di applicazioni web con HTML5. Sencha Touch è il framework che consente di sviluppare applicazioni che abbiano il look and feel nativo sia per iPhone che per Android. Sencha Touch è stato costruito, in particolare, per usare HTML5, CSS3 e JavaScript al massimo livello di potenza, flessibilità e ottimizzazione. Questo framework fa uso di HTML 5 per fornire componenti come audio e video, nonché un proxy local Storage per il salvataggio di dati offline.

Sencha Touch fa, anche, ampio uso di CSS3 nei suoi componenti e temi e fornisce uno styling layer incredibilmente robusto, che dà il controllo totale dell'aspetto grafico dell'applicazione. Tutto ciò avviene semplicemente caricando un apposito file CSS nella pagina `index.html` dell'applicativo. Sencha basa il proprio funzionamento su Ext JS e dispone di alcune componenti grafiche appositamente ottimizzate per un input di tipo touch. Molto interessante è il modo con cui gli sviluppatori di Sencha hanno cercato di definire un'esperienza d'utilizzo il più vicino possibile a quella di un moderno linguaggio OOP nonostante Sencha si basi su JavaScript.

1.2.5 Kendo UI

Kendo UI (figura 1.5) è un framework di sviluppo di applicazioni mobile cross-browser basati su jQuery in rapida crescita.

Figura 1.5. Logo di Kendo UI

Mentre altri framework si concentrano solo sulla parte dell'interfaccia utente, Kendo fornisce supporto allo sviluppo end-to-end, lato client. Infine, Kendo UI è compatibile con tutti i principali browser, anche i meno recenti, quali Internet Explorer 7+, Firefox 3+, Chrome, Safari 4+, e Opera 10+.

1.2.6 AppGyver

Alcuni anni fa gli imprenditori Marko Lehtimäki e Henri Vähäkainu, avendo la necessità di sviluppare applicazioni mobili per i clienti, e avendo constatato la mancata esistenza di un framework completo, crearono AppGyver (figura 1.6).

La mission aziendale di AppGyver è quella di rendere le applicazioni HTML5 ibride indistinguibili dalle applicazioni native nonché realizzare una piattaforma

Figura 1.6. Logo di AppGyver

completa per lo sviluppo, la quale permetta la creazione di applicazioni nel modo più rapido possibile. Proprio per queste ragioni AppGyver fornisce un set di strumenti completi, sia per principianti che per esperti sviluppatori, per costruire rapidamente, testare e distribuire le proprie applicazioni mobili.

Inizialmente, i creatori provarono ad utilizzare altre soluzioni preesistenti, ma non raggiunsero la qualità che si erano prefissati per le loro app; proprio per questo decisero di costruirsi una propria piattaforma.

AppGyver fornisce un nuovo strumento per lo sviluppo di applicazioni mobili che possiede funzionalità multi-piattaforma in grado di garantire prestazioni paragonabili a quelle derivate dalla programmazione nativa; esso garantisce, inoltre, la possibilità di creare rapidamente applicazioni completamente funzionali senza scrivere tantissime righe di codice.

AppGyver ha trovato una soluzione semplice al problema delle scarse prestazioni delle applicazioni scritte in HTML5, le quali sono state sovrastate dalle applicazioni native che presentano, generalmente, una performance di gran lunga migliore. Esso offre due opzioni differenti per la costruzione di applicazioni ibride, ovvero l'utilizzo di Steroids o di Composer.

Questa piattaforma, rispetto alle tecnologie che la precedono, presenta alcuni vantaggi importanti:

- A differenza di tante altre soluzioni per la creazione di app ibride, AppGyver è una piattaforma indipendente dal linguaggio, che permette l'integrazione dei dati e la distribuzione con un notevole supporto per plug-in.
- Utilizzando la struttura classica delle web app, Steroid possiede un supporto per built.io, Parse e l'integrazione con ogni REST API, il tutto mediante l'utilizzo di un'interfaccia utente point-and-click.
- La critica che più spesso viene attribuita alle applicazioni ibride sta nel fatto che non riescono a raggiungere né la performance né la completezza grafica delle applicazioni native, soprattutto quando si tratta di transizioni tra viste e animazioni; proprio per questo motivo AppGyver offre molti strumenti per rendere le applicazioni ibride indistinguibili da quelle native. Infatti, la piattaforma mette a disposizione degli sviluppatori gli strumenti necessari a costruire applicazioni cross-platform e pubblicarle nei marketplace. "l'AppGyver's Academy" fornisce soluzioni su tematiche comuni e guide sui framework a disposizione.

AppGyver mette a disposizione della comunity il Composer, ovvero uno strumento visuale basato sul browser che permette l'avvio e la modifica di applicazioni in modo rapido, semplice e senza un'eccessiva esperienza di programmazione. Il Composer è un tool gratuito, integrato con Adobe PhoneGap, messo a disposizione degli sviluppatori, in cui sono presenti i servizi di base. Attraverso l'acquisto è possibile utilizzare i componenti aggiuntivi, come un proxy di dati e i servizi cloud. Questo

dispositivo è dotato di un modello di programmazione "drag and drop", che permette di realizzare app mobile iOS e Android e fornisce un'interfaccia flessibile che consente anche a chi possiede conoscenze tecniche limitate di costruire applicazioni di alta qualità, sia dal punto di vista della logica di business che dal punto di vista della user interface.

Da quanto detto è facile dedurre in qual misura la programmazione mobile si sta sempre più spingendo verso una programmazione dove non si richiede una forte conoscenza di linguaggi di programmazione specifici. Alla creazione di un progetto viene fornita una libreria di componenti comuni per l'interfaccia utente dove è possibile trovare numerosi elementi, quali moduli, immagini, pannelli, elenchi, pulsanti; inoltre, viene garantita la possibilità di aggiungere codice HTML personalizzato.

Grazie alla possibilità di configurare controlli di interazione e logica condizionale per mezzo di un editor di logica visiva è possibile dar vita a questi componenti. Il Composer consente agli sviluppatori di utilizzare le API REST per aggiungere un back-end personalizzato o per integrare i servizi cloud di terze parti, come Parse e Kimono. Esso utilizza il framework JavaScript AngularJS assieme al framework Ionic.

1.2.7 Ionic

Ionic (figura 1.7) è un toolkit di interfaccia utente open source per la creazione di applicazioni mobili e desktop performanti e di alta qualità che utilizzano tecnologie web (HTML, CSS e JavaScript); è stato realizzato da Ben Sperry, Adam Bradley Lynch del team Drifty ed è considerato dagli autori stessi il "Bootstrap per le applicazioni ibride". Infatti, Bootstrap ha rivoluzionato la creazione di interfacce per siti web, e Ionic è nato con l'obbiettivo di diventare lo strumento principe per la creazione di app mobile ibride.

Figura 1.7. Logo di Ionic

Questo framework è nato nel 2013 e la sua popolarità è cresciuta molto rapidamente; infatti, oggi, vengono create migliaia di applicazioni con Ionic ogni mese. Ionic dà la possibilità di sviluppare la componente front-end di applicazioni mobile ibride. Per lo sviluppo di applicazioni tramite questo framework, si fa riferimento ad altri due componenti essenziali per il processo di sviluppo, ovvero AngularJS e Apache Cordova. La cooperazione di queste due tecnologie con Ionic rende la costruzione dell'app ibrida veloce e facile. Ionic ha diversi punti di forza:

- *Performance e reattività*: Ionic è stato progettato per garantire alti livelli di prestazione; infatti, elimina la necessità di modificare direttamente il DOM nonchè

di utilizzare librerie di terze parti per le animazioni e le transizioni; quest'ultima proprietà è ottenuta grazie al supporto dell'accelerazione grafica hardware.
- *Combinazione perfetta con AngularJS*: in Ionic la logica di business è gestita da AngularJS. L'unione tra le potenzialità offerte da AngularJS e le considerevoli funzionalità grafiche messe a disposizione da Ionic produce uno degli strumenti più adatti alla creazione di applicazioni mobili.
- *Orientamento nativo*: le componenti grafiche di Ionic sono state ideate per non discostarsi troppo da quelle native delle piattaforme mobili in modo da far avere all'applicazione un look and feel somigliante a quello di un'applicazione nativa.
- *Command Line-Interface (CLI) dedicata*: attraverso la digitazione di semplici comandi nella CLI, Ionic permette di eseguire numerose funzioni.

1.2.8 Xamarin

Xamarin (figura 1.8), acquisito recentemente da Microsoft e reso completamente gratuito, è un ambiente di sviluppo per applicazioni mobili cross-compilate che sta riscuotendo attualmente molto successo. Grazie all'utilizzo del linguaggio C#, esso permette al programmatore di realizzare applicazioni eseguibili su diverse piattaforme.

Figura 1.8. Logo di Xamarin

Xamarin offre tutte le API native delle diverse piattaforme come normali librerie C# permettendo, così, l'interazione con lo specifico dispositivo in maniera simile a come accadrebbe con il codice nativo. Alla base di Xamarin c'è Mono, un'implementazione open source del framework .NET di Microsoft, che consente di eseguire codice unico ottenendo tanti output diversi per quanti sono le piattaforme target, affiancandosi per tale scopo alla sua macchina virtuale nativa.

Il prossimo capitolo della tesi verrà completamente dedicato a questo framework.

1.2.9 React Native

React Native (figura 1.9) dà la possibilità di creare app mobili utilizzando solo JavaScript. Esso consente di creare un app più velocemente rispetto ad altri framework poiché, invece di ricompilare, permette di ricaricare immediatamente l'app.

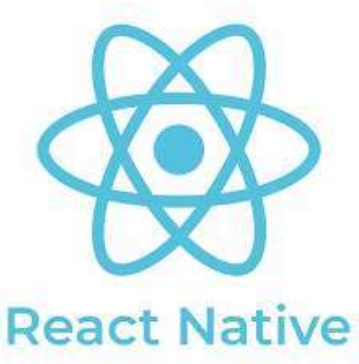

Figura 1.9. Logo di React Native

Con Hot Reloading è possibile persino eseguire un nuovo codice mantenendo lo stato dell'applicazione. React Native si combina perfettamente con i componenti scritti in Swift, Java o Objective-C. È semplice passare al codice nativo se si ha bisogno di ottimizzare alcuni aspetti dell'applicazione. È anche facile creare una parte dell'app in React Native e la restante parte usando direttamente il codice nativo; un esempio di quanto detto è l'applicazione di Facebook.

Attraverso il framework React Native è, quindi, possibile sviluppare applicazioni mobile ibride tramite l'utilizzo di JavaScript, il quale viene interpretato nel linguaggio nativo del sistema operativo.

React Native si basa sul framework React, che adotta un nuovo approccio allo sviluppo di applicazioni web. La filosofia di React si fonda sul concetto di componente, il quale possiede uno stato, può accettare delle proprietà attraverso il metodo `render()` e ha un suo ciclo di vita; questo approccio "atomico" agevola lo unit testing, migliora il riutilizzo del codice e permette lo sviluppo di componenti complessi tramite l'inclusione di altri più semplici.

Il metodo `render()` riceve in input le proprietà e lo stato del componente e restituisce un albero di contenuti da visualizzare, chiamato Virtual-DOM. Questo metodo viene eseguito ad ogni modifica dei dati del componente, generando, quindi, un nuovo Virtual-DOM. È degno di nota il fatto che React calcola le differenze tra il Virtual-DOM corrente e quello relativo allo stato precedente ed applica le minime operazioni necessarie per avere il DOM (Document Object Model) reale aggiornato. Lo sviluppo di un'applicazione mobile con il framework React Native risulta essere a tutti gli effetti quasi equivalente allo sviluppo di un'applicazione web; proprio per questo motivo la maggior parte del codice potrebbe essere riusato per lo sviluppo di applicazioni web con React, dovendo cambiare solo le componenti grafiche scelte dal momento che la logica del flusso dei dati risulta essere la stessa.

2

Xamarin

In questo capitolo, completamente dedicato a Xamarin, verranno ripercorse le principali tappe della storia di questo framework; verranno, inoltre, illustrati il linguaggio di programmazione C# adottato e gli strumenti per lo sviluppo da noi adottati.

2.1 Storia

La storia di Xamarin, seppur breve, è contraddistinta da molti avvenimenti che sono riassunti attraverso la time line mostrata in Figura 2.1.

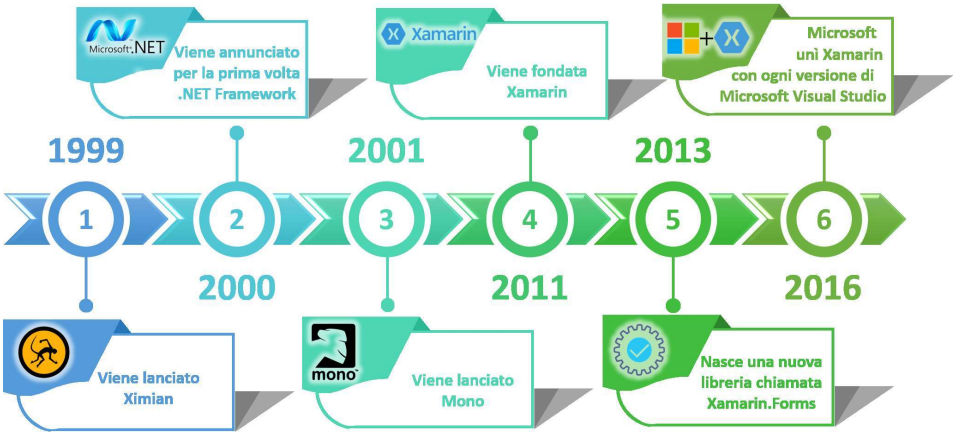

Figura 2.1. Storia di Xamarin

Xamarin è un'azienda statunitense produttrice di software con sede a San Francisco (California), fondata nel Maggio 2011 dagli ingegneri Nat Friedman e Miguel de Icaza. Questi, già nel 1999, avevano lanciato un programma, noto come Ximian, in grado di supportare e sviluppare software per il nascente progetto GNOME di de Icaza.

Nel Giugno 2000, dopo che Microsoft annunciò .NET Framework, de Icaza iniziò a studiare se fosse possibile una versione di Linux, credendo che .NET avesse il potenziale per aumentare la produttività del programmatore.

Riconoscendo che il loro piccolo team non poteva aspettarsi di costruire e supportare un prodotto completo, Nat Friedman e Miguel de Icaza hanno lanciato, il 19 Luglio 2001 alla conferenza O'Reilly, il progetto open source Mono, cioé un'implementazione alternativa del compilatore C# e del framework .NET di Microsoft su sistemi Linux e macOS.

Il 4 Agosto 2003, Ximian venne acquisito da Novell e, successivamente, nell'aprile 2011 da Attachmate. Dopo l'acquisizione, Attachmate annunciò centinaia di licenziamenti, inclusi gli sviluppatori Mono, mettendo in discussione il futuro di Mono.

Non molto tempo dopo, il 16 Maggio 2011, Miguel de Icaza annunciò, sul suo blog, che Xamarin, una società di nuova costituzione, che aveva in programma di rilasciare una nuova suite di prodotti mobile, avrebbe sviluppato e supportato Mono, giungendo, alcuni anni dopo, alla creazione dell'omonimo framework Xamarin. Il nome Xamarin deriva dalla sostituzione della T con una X nel nome della scimmia Tamarin. Questo é in linea con il tema di denominazione usato sin da quando Ximian é stato avviato.

Dopo l'annuncio di Xamarin, il futuro del progetto é stato messo in discussione, dal momento che MonoTouch e Mono per Android si trovavano in diretta concorrenza con le offerte commerciali esistenti di proprietà di Attachmate. Va posta particolare attenzione al fatto che, all'epoca, non si sapeva come Xamarin avrebbe dimostrato di non aver utilizzato, in modo del tutto illegale, tecnologie precedentemente sviluppate da Novell, quando erano impiegate per lo stesso lavoro.

Tuttavia, nel Luglio 2011, Novell, ora associata di Attachmate e Xamarin, concesse una licenza perpetua per Mono, MonoTouch e Mono per Android a Xamarin, la quale ottenne formalmente e legalmente la gestione ufficiale del progetto.

Nel Dicembre 2012, Xamarin rilasciò Xamarin.Mac, un plug-in per l'ambiente di sviluppo integrato MonoDevelop (IDE), che consentiva agli sviluppatori di creare applicazioni basate su C# per il sistema operativo macOS di Apple. Nel Febbraio 2013, Xamarin annunciò il rilascio Xamarin 2.0 contenente due componenti principali:

- Xamarin Studio, un re-branding del suo IDE Monodevelop open source;
- l'integrazione con Visual Studio, IDE di Microsoft per .NET Framework, che consentiva di utilizzare Visual Studio per la creazione di applicazioni per Android e iOS, nonché per Windows.

Xamarin 2.0 contenente Xamarin.Android e Xamarin.iOS rese possibile lo sviluppo nativo di Android, iOS e Windows in C#, con Visual Studio o Xamarin Studio. Gli sviluppatori ebbero così la possibilità di riutilizzare il loro codice C# esistente per lo sviluppo di applicazioni su diverse piattaforme.

Negli anni il progetto ha raggiunto l'approvazione non solo degli sviluppatori, ma anche delle aziende. Infatti, Microsoft, il 24 Febbraio 2016, acquisì l'azienda dando inizio a una nuova era per lo sviluppo di applicazioni multipiattaforma. Il 31 Marzo 2016 Microsoft ha annunciato che stava unendo tutto il software Xamarin con ogni versione di Microsoft Visual Studio, inclusa Visual Studio Community;

ciò ha aggiunto varie funzionalità Xamarin preinstallate in Visual Studio, come un emulatore iOS. Inoltre, i prodotti Xamarin sono diventati ufficialmente open source e gratuiti per sviluppatori indipendenti, piccoli team e studenti, che possono utilizzarli nei propri progetti.

2.2 C#

Da sempre Xamarin ha voluto offrire strumenti che dessero la possibilità agli sviluppatori di operare sui principali sistemi operativi per dispositivi mobili (Android, iOS e Windows), massimizzando il riutilizzo del codice e delle competenze acquisite; é proprio per questo che Xamarin sfrutta Mono, il quale permette di utilizzare C#.

C# é un linguaggio di programmazione orientato agli oggetti e sviluppato da Microsoft all'interno dell'iniziativa .NET, che é stato successivamente approvato come standard da ECMA (ECMA-334) e ISO (norma ISO/IEC 23270).

Mono e C# sono, quindi, il fulcro su cui Xamarin costruisce i propri strumenti e le proprie tecnologie di sviluppo. Questo é un aspetto fondamentale per gli sviluppatori .NET poiché possono riutilizzare le proprie competenze anche su sistemi non Microsoft. Per rendere tutto questo possibile, Xamarin ha sviluppato delle librerie che rielaborano ed espongono in forma .NET/Mono le API native delle varie piattaforme (Figura 2.2),

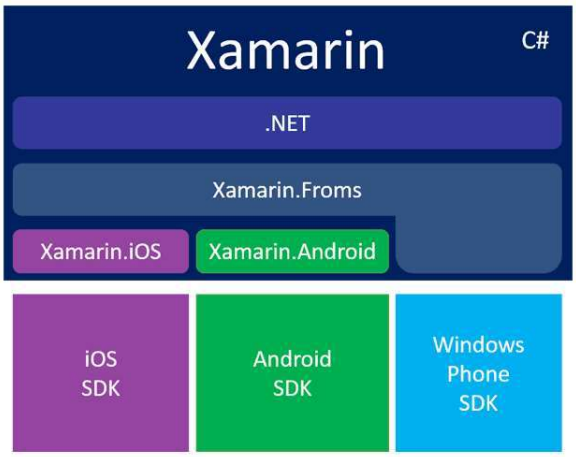

Figura 2.2. Componenti di Xamari e il loro legame con le SDK native

tali API native sono:

- *Xamarin.Android*: basato su Mono, consente lo sviluppo di applicazioni Android tramite un wrapping delle API native in C#;
- *Xamarin.iOS*: basato sulla libreria proprietaria MonoTouch, offre pieno supporto allo sviluppo di un'applicazione iOS tramite un wrapping delle API native in C#;

- *Xamarin.Forms*: si pone al di sopra dei precedenti componenti e permette la piena condivisione del codice.

Oltre a quelle appena elencate va aggiunta Xamarin.WinPhone che, per ovvi motivi, é stata sviluppata facilmente, in quanto, per lo sviluppo di applicazioni per Windows Phone, viene già fatto uso nativamente di C# e di API in forma .NET.

A tale proposito é facilmente deducibile come l'eventuale ampliamento futuro del supporto verso altri sistemi operativi possa essere effettuato in maniera più o meno rapida, aggiungendo esclusivamente una nuova libreria Xamarin.X. In aggiunta, sono stati pensati e creati appositamente degli ambienti di sviluppo e dei tool, oltre al runtime e alle librerie, ovviamente.

Nonostante Xamarin offra agli sviluppatori tutto ciò di cui hanno bisogno per sviluppare, pubblicare, analizzare e mantenere app native per Android, iOS e Windows, con riferimento a tutto il ciclo di vita dell'applicazione stessa, si deve tener presente che le librerie e i tool di sviluppo sono stati rilasciati come progetto open source a cui la comunità di sviluppatori può contribuire.

Qualsiasi cosa realizzabile in Objective-C, Swift o Java può essere fatta in C#. Xamarin ha scelto C# come unico linguaggio, permettendo, così, agli sviluppatori di ottenere applicazioni native senza necessariamente conoscere altri linguaggi di programmazione e utilizzando un unico ambiente di sviluppo. Le librerie sviluppate da Xamarin permettono tutto questo poiché consentono di accedere a qualsiasi API dei sistemi operativi nativi tramite meccanismi di binding e di realizzare un mapping in C#. Proprio per tale motivo tutte le API native sono utilizzabili in Xamarin tramite la sintassi e le naming convention di C# e del framework .NET (Figura 2.3).

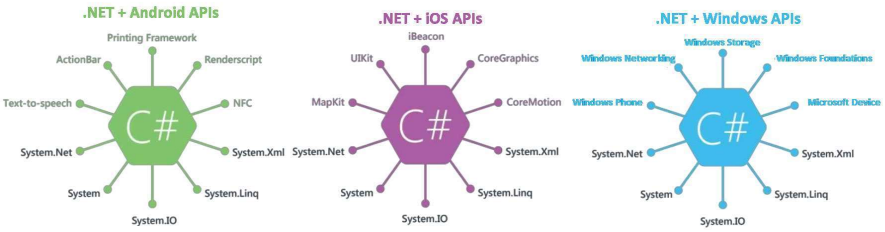

Figura 2.3. Alcune delle librerie sviluppate da Xamarin

Attraverso il binding nativo, inoltre, le librerie Xamarin danno la possibilità di chiamare da C# il codice esistente scritto nei linguaggi nativi, permettendo, così, il riutilizzo del codice.

Una volta scritto il codice sorgente in C# occorre compilarlo allo scopo di ottenere il file binario eseguibile dell'applicazione nativa. Inizialmente tale codice viene convertito dal framework in un linguaggio intermedio denominato CIL (Common Intermediate Language), che risulta indipendente dal linguaggio e comprensibile dal CLR (Common Language Runtime), l'ambiente di esecuzione .NET; successivamente, il codice CIL, non ancora eseguibile direttamente dal sistema operativo, subisce un secondo processo di compilazione differente al variare della piattaforma (Figura 2.4).

Figura 2.4. Compilazione nelle diverse piattaforme

Più specificatamente:

- Per la piattaforma Android, CIL viene inserito nell'APK e, al momento dell'esecuzione dell'applicazione, viene convertito in codice nativo tramite compilazione Just-In-Time (JIT) dal dispositivo con un approccio simile al framework .NET. Di conseguenza, Xamarin produce un pacchetto che offre le stesse performance di un'applicazione scritta in Java.
- Per la piattaforma iOS, per via di alcune restrizioni di Apple, la generazione dinamica di codice tramite un compilatore JIT é proibita e proprio per questo motivo, il codice CIL ottenuto viene ricompilato, ove possibile, direttamente in codice macchina (assembly ARM) tramite una compilazione Ahead-Of-Time (AOT). Il processo di build produce direttamente il codice nativo, offrendo le stesse performance di un'applicazione realizzata con Objective-C o Swift.
- Per la piattaforma Windows Phone, una volta ottenuto il codice CIL, non c'è bisogno di nessun tool particolare per la sua esecuzione, poiché l'ambiente di esecuzione CLR .NET é già presente nella piattaforma.

L'approccio tradizionale di Xamarin dà la possibilità di realizzare, per mezzo delle librerie, un layer comune condiviso tra le varie piattaforme e scritto in C#, il quale rappresenta il back-end, cioé la logica dell'app code behind. Esso, a seconda della complessità del codice dell'applicazione da realizzare, può coprire in media il 75% del codice nativo condivisibile (in alcuni casi questa percentuale raggiunge anche il 100%). Su questa base comune di back-end si poggerà, poi, la realizzazione dell'interfaccia grafica nativa, una per ogni piattaforma considerata, sfruttando i linguaggi di markup e i tool nativi. Nello specifico, quindi, si potrà avere (Figura 2.5):

- per Android: C# + XML;
- per iOS: C# + XIB/Storyboard;
- per Windows Phone: C# + XAML.

2.3 Xamarin.forms

Xamarin.forms é stato introdotto in Xamarin 3 il 28 maggio 2014 e consente di utilizzare sottoinsiemi di controlli portatili associati a controlli nativi di Android, iOS e Windows Phone.

Figura 2.5. Approccio tradizionale di Xamarin

L'obiettivo di Xamarin é il raggiungimento del 100% del codice condiviso; proprio per questo motivo Xamarin.Forms fornisce un livello di astrazione che consente di definire l'interfaccia grafica una solta volta attraverso la derivazione di XAML (eXtensible Application Markup Language), ovvero il linguaggio di markup .NET per la progettazione della UI in modalità dichiarativa, oppure attraverso il codice C#, come mostrato nella Figura 2.6.

Figura 2.6. Approccio di Xamarin.forms

A questo punto l'interfaccia grafica sarà costruita esclusivamente per mezzo di questa astrazione tramite i classici componenti (pagine, layout, viste, pulsanti, etichette, etc.) e, solo in fase di esecuzione, Xamarin.Forms effettuerà il render selezionando i corrispettivi elementi specifici del sistema su cui l'applicazione sta girando, così da ottenere un aspetto nativo. Il controllo Label di Xamarin.Forms, infatti, diventerà una TextView in Android, una UILabel in iOS e un TextBlock in Windows Phone. Il codice condiviso é contenuto in uno specifico progetto all'interno della solution e può essere di due tipi:

- *shared*: si tratta di file condivisi tra i progetti tramite il meccanismo dei link;
- *portable*: si tratta di una vera e propria portable class library, che produce in output una libreria DLL.

All'interno del progetto condiviso verrà, poi, scritto il codice in grado di funzionare con certezza su tutte le piattaforme, anche se occorre considerare il fatto che Xamarin.Forms riesce a mappare solo gli elementi che hanno una corrispondenza in tutte le piattaforme e, quindi, definisce controlli che hanno esclusivamente proprietà e comportamenti comuni a tutti, lasciando fuori gli aspetti specifici di ciascuna piattaforma. Proprio per questo motivo, poiché sia iOS che Android che Windows possiedono caselle di testo ed etichette, Xamarin.forms definisce controlli Entry e Label che, sicuramente, saranno utilizzabili dalle piattaforme, anche se, naturalmente, ognuna di esse gestirà in modo diverso entrambi gli elementi visuali, i quali presentano proprietà e comportamenti differenti. Di recente, tuttavia, é stata inserita la possibilità di utilizzare controlli nativi, i quali, naturalmente, saranno specifici per ogni piattaforma.

Funzionalità come la comunicazione Bluetooth, necessitano, comunque, di essere gestite in modo differente a seconda della piattaforma. Di questa problematica ci occuperemo in dettaglio nel prossimo capitolo.

In Xamarin.forms ci sono tre possibilità per fare questo:

- *Effects*: usato nel caso in cui si vogliano utilizzare controlli nativi, ma senza dover intervenire anche sul loro comportamento, ad esempio per la modifica estetica.
- *Custom Renderer*: usato per la personalizzazione di tutti i layout e gli elementi visuali in generale, usufruendo di controlli nativi in modo diretto attraverso l'implementazione di una classe per ogni progetto specifico, volta a descrivere proprietà tipiche di quella piattaforma.
- *Dependency injection*: riguarda l'accesso a funzionalità hardware e software (sensori, fotocamera, file system) specifiche della piattaforma, realizzato tramite la definizione di un'interfaccia comune per descrivere i membri di una classe e fornendo nei progetti specifici la relativa implementazione.

Per di più va posta attenzione al fatto che Xamarin.forms, essendo una piattaforma giovane e in evoluzione, non dispone di tutto ciò che é di solito possibile utilizzare nelle piattaforme specifiche o in altri contesti di sviluppo. Per tale ragione Xamarin.forms, attrverso NuGet (il package manager gratuito realizzato per le piattaforme Microsoft), dà la possibilità di scaricare numerosi plugin per Xamarin, ovvero librerie per l'accesso a funzionalità specifiche di piattaforma tramite API unificate che sono fruibili dai progetti condivisi.

Nella Figura 2.7 é possibile vedere la suddivisione della soluzione, nei due macro progetti:

- *Platform-Specific Application Projects*: si tratta dell'insieme dei progetti delle piattaforme che si desiderano supportare per l'applicazione; nella figura 2.7 tali progetti sono rappresentati dalle diverse piattaforme per le quali si desidera sviluppare l'applicazione.
- *Core Project*: é composta dagli elementi essenziali; si tratta, infatti, del cuore dell'applicazione, come l'accesso ai dati e servizi remoti e alla logica di business; nella Figura 2.7 tale progetto é costituito dallo Shared Code.

Occorre, inoltre, affermare che lo Shared Code é composto da un insieme di strati, ciascuno dei quali possiede un compito preciso all'interno dell'applicazione:

- *Data Access Layer*: costituito da un insieme di API che, una volta richiamate dal Business Layer attraverso delle interfacce, permetteranno l'accesso ai dati del database;
- *Business Layer*: contenente la logica e le operazioni principali per offrire l'interazione con l'utente.

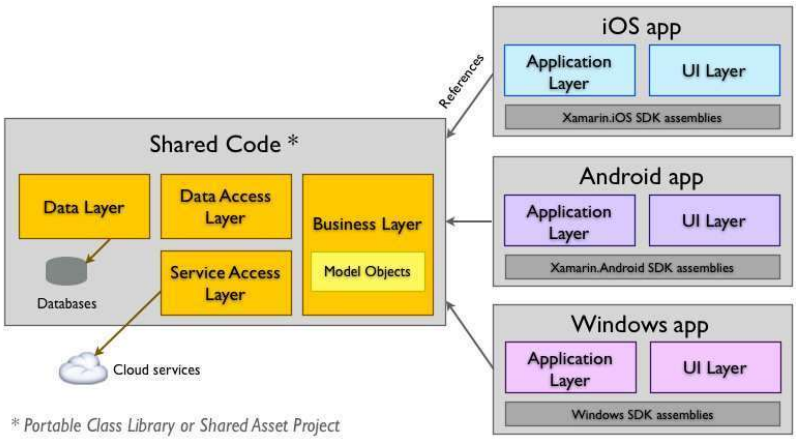

Figura 2.7. Architettura di un'applicazione multipiattaforma

Xamarin fornisce due diverse modalità di condivisione e riutilizzo del codice tra i progetti delle diverse piattaforme. Più precisamente, il Core Project permette una condivisione del codice attraverso due diversi approcci:

- *Shared Assets Project (SAP)*: dà la possibilità di condividere il codice attraverso l'intera soluzione, tramite le direttive del compilatore, al fine di includere differenti porzioni di codice specifiche per ciascuna piattaforma.
  I file inclusi nel progetto vengono, infatti, copiati all'interno dei progetti specifici delle piattaforme e compilati come parte di essi, come visibile nella Figura 2.8.
  Durante il processo di build non viene prodotta una DLL compilata, ma, più semplicemente, un assortimento di file; pertanto, una tipica soluzione a riguardo potrebbe contenere quattro progetti: uno Shared, uno specifico Android, uno specifico iOS, uno specifico Windows Phone. Va, anche, affermato che questo approccio supporta la compilazione condizionale tramite le direttive del compilatore al fine di gestire le differenze tra le varie piattaforme e includere il codice specifico solo in un sottoinsieme dei progetti.
  I progetti basati su questo approccio hanno, però, un limite di grande rilievo; i progetti Android e iOS, infatti, accedono ad una versione del .NET framework che differisce da quella usata per i progetti Windows. Di conseguenza occorre differenziare il codice per ogni piattaforma supportata in modo tale da riuscire

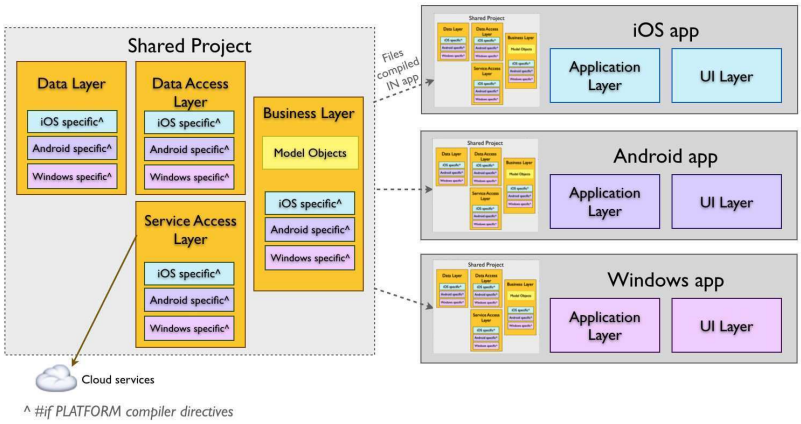

Figura 2.8. Approccio SAP

ad effettuare un'operazione d'interesse come, ad esempio, l'accesso al database. Una soluzione a questo problema viene visualizzata nel listato 2.1, dove C# dà la possibilità, attraverso `#if`, `#elif` e `#endif`, di dare direttive al compilatore.

```
#if__IOS__
// codice specifico per iOS
#elif__ANDROID__
// codice specifico per Andro id
#elif WINDOWS_PHONE_APP
// codice specifico per Windows Phone
#end if
```

Listato 2.1. gestione multipiattaforma SAP

- *Portable Class Libraries (PCLs)*: permette di usare un sottoinsieme di librerie offerte dal framework .NET per sfruttare lo stesso codice nei progetti che compongono l'intera soluzione. L'insieme di librerie da utilizzare é limitato dalle piattaforme che si desidera supportare per la propria applicazione.
  Le Portable Class Libraries sono delle librerie portabili ed, in quanto tali, possono essere consumate su più piattaforme a condizione che abbiano come target solo il sottoinsieme di API che risulta comune e supportato su tutte le piattaforme di destinazione. Le PCL producono una DLL come output, la quale può essere, poi, distribuita e utilizzata in altri progetti. Tuttavia le stesse non supportano la compilazione condizionale e, quindi, non possono contenere il codice specifico di una certa piattaforma proprio perché perderebbero la loro caratteristica di portabilità.
  Nella Figura 2.9 é possibile notare varie frecce, le quali mostrano come ogni specifica piattaforma referenzi la PLC e come sia possibile impiegare caratteristiche di una determinata piattaforma attraverso l'utilizzo del design pattern Dependency Injection (DI).
  Anche in questo caso, lo sviluppo dell'applicazione é diviso in più progetti distinti, uno per ogni specifica piattaforma supportata, più uno Portable che racchiude tutti gli elementi comuni dello sviluppo che, solo in fase di compilazione, vengono

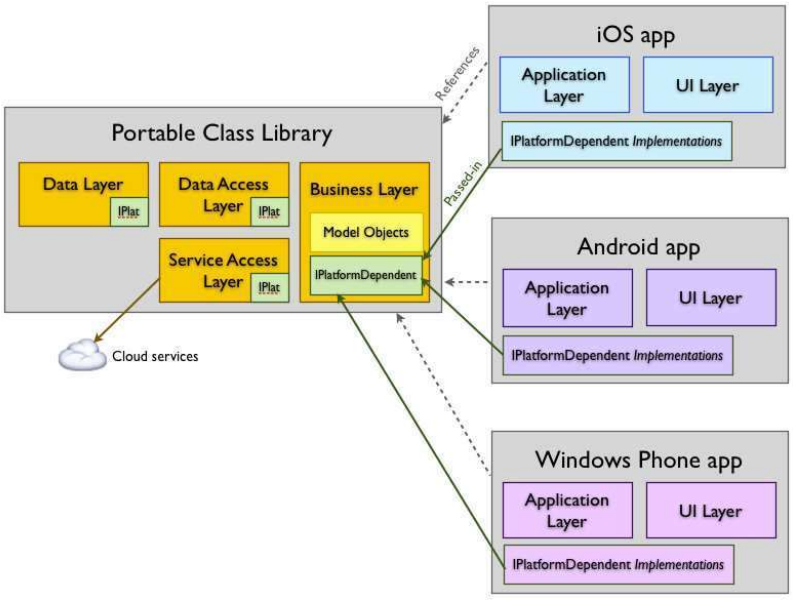

Figura 2.9. Approccio PCL

inseriti nei rispettivi progetti specifici.

Le componenti utilizzate per creare l'interfaccia utente di un'applicazione Xamarin.Forms possono essere suddivise in quattro gruppi:

- *Views*: con il termine "view" vengono definiti blocchi di costruzione delle interfacce utente, come pulsanti, etichette o caselle di testo (controlli dei widget); questi elementi sono, tipicamente, sottoclassi di View.
- *Cells*: con il termine "cell" si intende un elemento specializzato che viene utilizzato per descrivere l'aspetto dei singoli item di una tabella o di una lista. Le celle non sono degli elementi visivi, ma descrivono soltanto un template per la creazione di un elemento visivo e sono elementi progettati per essere aggiunti ai controlli ListView o TableView. La Figura 2.10 mostra la classificazione dei controlli in Xamarin.forms.
- *Layout*: con il termine "layout" si fa riferimento ad un sottotipo specializzato di View che fungono da contenitori per altri layout o viste; i layout, inoltre, vengono usati per organizzare i controlli dell'interfaccia utente (view e cell) in strutture logiche; infine, essi, di solito contengono la logica per impostare la posizione e la dimensione degli elementi figlio nelle applicazioni Xamarin.Forms. La Figura 2.11 mostra la classificazione dei layout in Xamarin.forms.
- *Pages*: con il termine "page" vengono identificati degli elementi visivi che occupano la maggior parte, o addirittura tutto, lo schermo e contengono un singolo figlio; essi, quindi, definiscono la schermata dell'applicazione. Una pagina rappresenta un View Controller in iOS o una Page in Windows Phone, mentre in Android può essere paragonata ad un'Activity, anche se, in realtà, le pagine

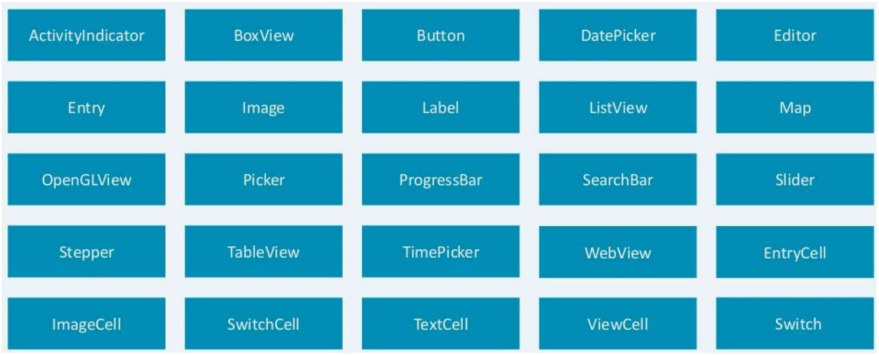

Figura 2.10. Controlli in Xamarin.forms

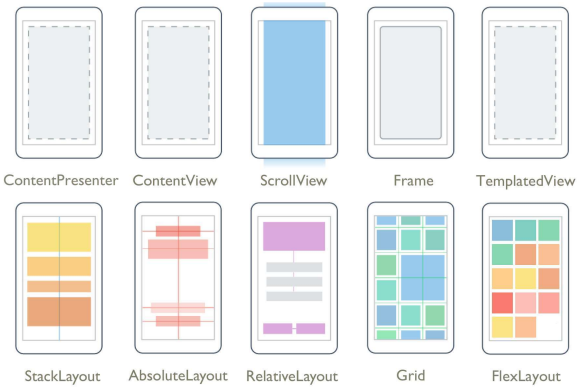

Figura 2.11. Layout in Xamarin.forms

in Xamarin.forms non sono vere e proprie Activity. La Figura 2.12 mostra la classificazione delle pagine in Xamarin.forms.

Figura 2.12. Pagine in Xamarin.forms

2.4 Strumenti per lo sviluppo

Xamarin Studio, nato nel Febbraio 2013, inizialmente dava la possibilità agli sviluppatori di programmare le proprie applicazioni per Android e Windows Phone dal PC tradizionale, e per Android e iOS se usato su Mac.

Oggi Xamarin Studio é obsoleto; difatti, in seguito all'acquisizione di Xamarin da parte di Microsoft, esso é stato sostituito con Visual Studio per Windows (Figura 2.13) e Visual Studio for Mac, per Mac OS.

Figura 2.13. Logo di Visual Studio

Xamarin viene facilmente integrato in Visual Studio attraverso l'aggiunta di un plugin; questo, a differenza di Xamarin Studio, offre la possibilità di sviluppo per tutti e tre i principali sistemi operativi (Android, iOS e Windows Phone).

Il completo sviluppo di un'applicazione Android può essere effettuato utilizzando Visual Studio, sia su Windows che su Mac, e sia su macchina fisica che virtuale. Invece, per le applicazioni iOS, la fase di compilazione necessita della presenza di una macchina fisica Mac con installata la versione aggiornata dell'IDE Xcode e dell'SDK iOS. Per il sistema operativo Linux non é previsto alcun supporto a Xamarin; tuttavia, si può usare il suo precursore Mono con l'appoggio dell'IDE MonoDevelop, che é una delle migliori alternative a Visual Studio per sviluppare in C# con Mono.

Una volta installato Visual Studio con Xamarin su Windows, per creare un progetto Xamarin.Forms per lo sviluppo di un'applicazione Cross-Platform, occorre seguire il percorso File > Nuovo > Progetto, come riportato nella Figura 2.14.

Dopo aver cliccato su "Progetto" si aprirà la finestra di dialogo "Nuovo progetto" (riportata nella Figura 2.15), nella quale, una volta recatesi su Cross-Platform, occorre selezionare "App per dispositivi mobili (Xamarin.Forms)" e definire il nome e il percorso nel quale verrà salvato il progetto.

Successivamente si aprirà la finestra di dialogo "New Cross Platform App", dalla quale è possibile scegliere il template (Blank App), la tecnologia UI (Xamarin.Forms) e la strategia di condivisione del codice da utilizzare.

Il numero di progetti generati varierà a seconda di quanti SDK sono installati. Nel caso in cui venissero scelti solo Android e iOS, come nel progetto oggetto di tesi, in "Esplora soluzioni" si vedrebbero tre progetti, come mostrato nella Figura 2.16.

Tra i file che compongono il progetto é possibile trovare:

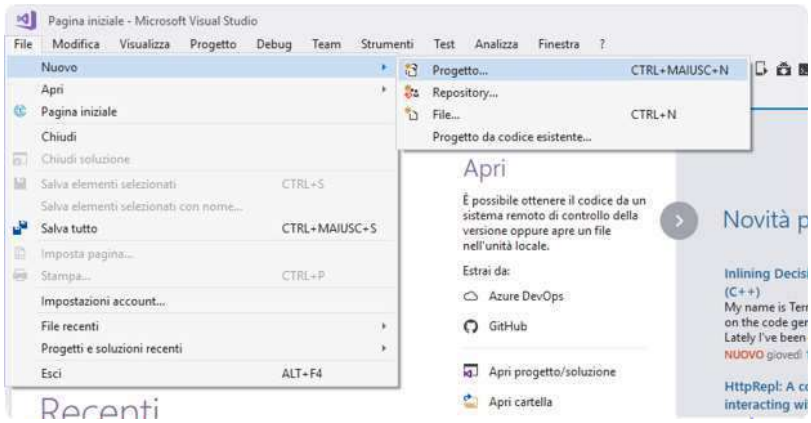

Figura 2.14. Creazione di nuovo progetto in Visual Studio 2017

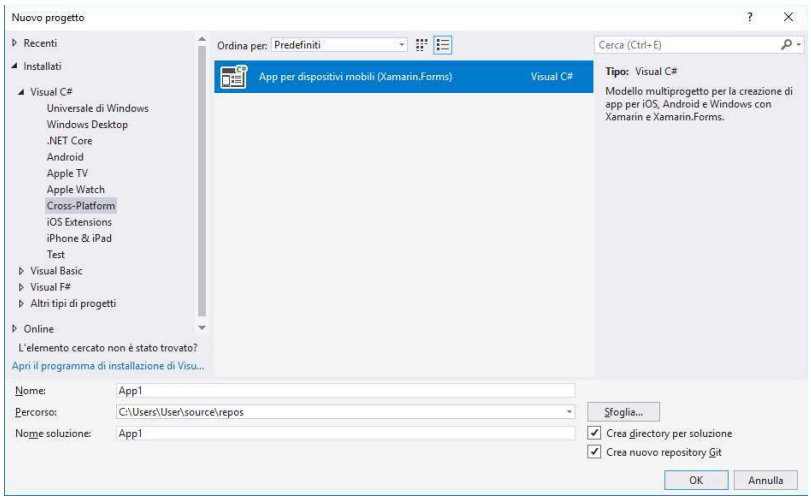

Figura 2.15. Finestra di dialogo "Nuovo progetto" in Visual Studio 2017

- `App.xaml`: rappresenta il codice di markup XAML per la classe `App`, il quale definisce un dizionario di risorse (stili, data template) fruibili dall'applicazione nel suo complesso.
- `App.xaml.cs`: il code-behind per la classe `App`, che si rende responsabile di instanziare la prima pagina mostrata dall'applicazione su ogni piattaforma e di gestire gli eventi del ciclo di vita dell'applicazione stessa.
- `MainPage.xaml`: il codice di markup XAML per la classe `MainPage`, il quale definisce l'interfaccia utente della pagina mostrata all'avvio dell'applicazione.
- `MainPage.xaml.cs`: il code-behind per la classe `MainPage`, che contiene la logica di business eseguita quando l'utente interagisce con la pagina.

Per lavorare con i pacchetti aggiornati all'ultima versione stabile, occorre cliccare su Strumenti > Gestione pacchetti NuGet > Gestisci pacchetti NuGet per la

Figura 2.16. Esplora soluzioni in Visual Studio 2017

soluzione...; dopodiché si deve selezionare Aggiornamenti e Installa. Va percorso lo stesso iter per installare o rimuovere nuovi pacchetti.

Per generare l'applicazione occorre selezionare Compilazione > Compila soluzione. A questo punto, se tutto é stato eseguito correttamente, verrà mostrato un messaggio di successo nella barra di stato di Visual Studio; al contrario, qualora vi siano degli errori, essi verranno segnalati in modo tale da poterli poi correggere e rieseguire la procedura.

Per testare l'applicazione occorre selezionare la piattaforma e l'emulatore; inoltre, è possibile utilizzare un dispositivo fisico per effettuare il debug, naturalmente dopo averlo predisposto in modalità sviluppatore. Nella Figura 2.17 viene mostrata la barra dalla quale é possibile selezionare il dispositivo o l'emulatore.

Per lo sviluppo dell'applicazione oggetto di tesi è stato utilizzato un dispositivo fisico per il debug, poiché un emulatore non offre la possibilità di effettuare connessioni Bluetooth.

Figura 2.17. Toolbar per il testing in Visual Studio 2017

3

La gestione del bluetooth nelle app

In questo capitolo verranno introdotti il Bluetooth e la modalità tramite cui è possibile implementare un'applicazione che utilizza questo protocollo.

3.1 Generalità

Il Bluetooth (Figura 3.1) rappresenta un modo tramite cui i dispositivi possono comunicare in modalità wireless su brevi distanze. Le comunicazioni wireless sono state introdotte intorno alla fine del diciannovesimo secolo ed hanno preso forma nella radio, nei raggi infrarossi, nella televisione e, più recentemente, nell'IEEE 802.11 (in informatica e telecomunicazioni, con tale sigla si definisce un insieme di standard di trasmissione per le reti WLAN). Nonostante la connessione Bluetooth assomigli a quella Internet, ciò che contraddistingue il Bluetooth è la sua particolare attenzione alla breve distanza di comunicazione; questa differenza influenza notevolmente la modalità con cui ogni dispositivo inizialmente trova l'altro e stabilisce una connessione; dopo di che, tutto è praticamente uguale alla connessione Internet. La massima potenza prevista per un dispositivo non permette trasmissioni oltre i 100 metri; nella realtà, poiché questa tecnologia è particolarmente sfruttata in dispositivi portatili in cui il consumo energetico è un fattore cruciale, la potenza di trasmissione dei dispositivi non permette l'invio di dati oltre i 9/10 metri.

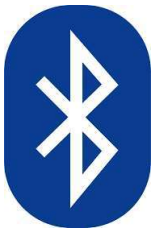

Figura 3.1. Icona del Bluetooth

Il problema nello stabilire una connessione tra dispositivi risiede nel fatto che occorre stabilire una connessione in uscita o in entrata poiché si deve poter distinguere il dispositivo che invia il primo pacchetto di dati per avviare le comunicazioni (client) e il dispositivo che riceve quel pacchetto (server). Nello specifico, si usano le parole "client" e "server" solo per distinguere il dispositivo che avvia una connessione rispetto all'altro, e senza che ciò comporti alcuna relazione con il modello client-server della programmazione network.

Prima di stabilire una connessione e trasferire dati, i dispositivi che l'attivano devono scegliere un dispositivo di destinazione e un protocollo di trasporto; i dispositivi che stabiliscono una connessione in entrata devono scegliere un protocollo di trasporto e poi "ascoltare" prima di accettare una connessione e trasferire dati.

La Figura 3.2 illustra i concetti di base relativi alle connessioni e il modo in cui quest'ultimi vengono implementati in Internet e nel Bluetooth. Si noti che per la connessione solo le prime due fasi (la scelta di un dispositivo di destinazione, di un protocollo di trasporto e di un numero di porta) sono diverse tra Bluetooth e Internet, mentre per il resto si mantiene tutto identico. I processi per accettare una connessione in ingresso sono ancora più simili tra loro, con la differenza principale dell'aggiunta del supporto Bluetooth per numeri di porta assegnati dinamicamente.

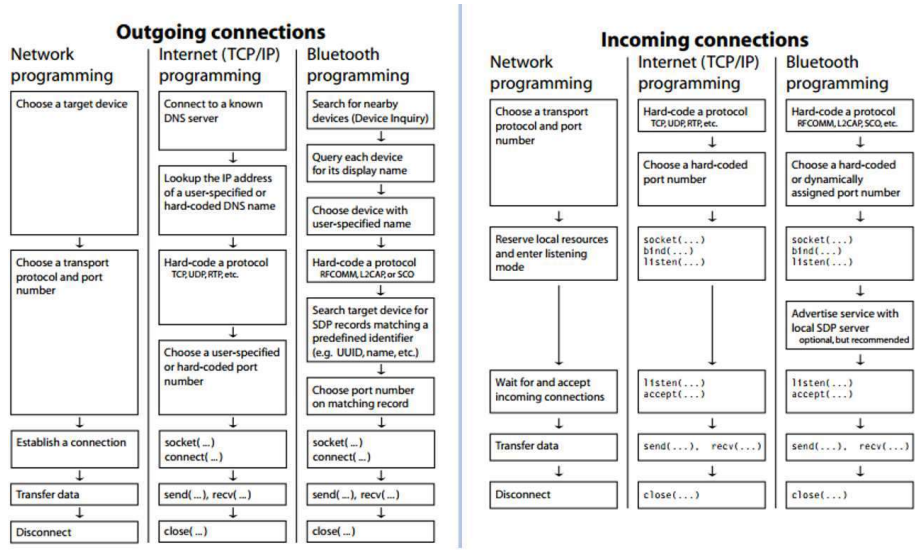

Figura 3.2. Connessione internet e Bluetooth a confronto

Su ogni chip Bluetooth prodotto viene impresso un indirizzo a 48 bit univoco globale, denominato indirizzo Bluetooth, o indirizzo del dispositivo, le cui caratteristiche sono analoghe a quelle del Machine Address Code (MAC) per l'Ethernet. In realtà, entrambi gli indirizzi sono gestiti dalla IEEE e vengono assegnati al momento della produzione; inoltre sono destinati ad essere unici e rimanere tali per tutta la durata del chip.

Un dispositivo Bluetooth, per poter comunicare con un altro, deve necessariamente determinare l'indirizzo Bluetooth dell'altro, il quale viene utilizzato in tutti gli strati del processo di comunicazione Bluetooth, dai protocolli radio di basso livello ai protocolli applicativi di livello superiore. Al contrario, i dispositivi di rete TCP/IP, che utilizzano Ethernet, scartano l'indirizzo MAC a 48 bit a strati superiori del processo di comunicazione e lo commutano mediante indirizzi IP. Il principio rimane, comunque, lo stesso poiché l'indirizzo identificativo univoco del dispositivo di destinazione deve essere noto per riuscire a comunicare con esso.

Il dispositivo client non può conoscere fin da subito gli indirizzi di destinazione, ma nel protocollo TCP/IP utilizzato da Internet l'utente sa o fornisce un nome host, come ad esempio `www.univpm .it`, il quale deve poi essere tradotto in un indirizzo IP fisico. Nella connessione Bluetooth l'utente sarà, in genere, identificato da un nome facile da usare, come, ad esempio, "My Phone", e il client tradurrà questo nome in un opportuno indirizzo numerico, cercando i dispositivi Bluetooth nelle vicinanze e controllando il nome di ogni dispositivo.

Gli esseri umani non interagiscono facilmente con i numeri a 48 bit, come per esempio 0x000EED3D1829; di conseguenza i dispositivi Bluetooth presentano un nome facile da usare. Questo nome viene solitamente mostrato all'utente in sostituzione dell'indirizzo Bluetooth per identificare un dispositivo, ma alla fine è l'indirizzo Bluetooth che viene utilizzato nella comunicazione reale. Per molte macchine, come ad esempio smartphone e computer desktop, tale nome è configurabile e l'utente può scegliere una parola o una frase arbitraria; tuttavia, il fatto che non v'é alcun obbligo per l'utente di scegliere un nome univoco può causare confusione, nel caso molti dispositivi adiacenti possiedano lo stesso nome.

I nomi dei dispositivi Bluetooth sono simili ai nomi DNS in quanto entrambi corrispondono ad identificatori di facile riconoscimento per gli esseri umani, i quali, alla fine, vengono tradotti in identificatori adatti alle macchine. I nomi dei dispositivi Bluetooth si differenziano dai nomi DNS in quanto questi ultimi, sono unici e vengono registrati in un database centrale (non ci possono essere più `www.univpm.it`); mentre i nomi dei dispositivi Bluetooth sono più o meno arbitrari e spesso duplicati; inoltre, vengono registrati solo sul dispositivo stesso. Nel TCP/IP, utilizza un DNS, il quale viene tradotto in un indirizzo IP in modo tale da poter contattare un server, emettere una query e attendere una risposta. Nel Bluetooth il processo di ricerca è invertito: in primo luogo un dispositivo cerca i dispositivi Bluetooth nelle vicinanze e compila un elenco dei loro indirizzi; poi, contatta individualmente ciascun dispositivo nelle vicinanze attraverso la query per mezzo del nome Bluetooth.

Un altro aspetto molto importante riguarda l'individuazione dei dispositivi, ossia il processo di ricerca e di identificazione di dispositivi Bluetooth nelle vicinanze. Per capire che cosa c'è nelle vicinanze, i dispositivi trasmettono un messaggio "discovery" e poi attendono le risposte dei dispositivi limitrofi. In particolare, ogni risposta è costituita dall'indirizzo del dispositivo che risponde e da un intero che identifica la classe del dispositivo stesso (ad esempio, un telefono cellulare, un PC desktop, un auricolare, etc).

Il processo di ricerca dei dispositivi Bluetooth vicini può richiedere molto tempo, ovvero, dato un telefono cellulare Bluetooth e un computer portatile Bluetooth disposti uno accanto all'altro su una scrivania, di solito occorrono in media 5 secondi prima che il telefono rilevi la presenza del portatile; durante il processo di rileva-

mento dei dispositivi, lo smartphone cambia frequenze più di mille volte al secondo, e ci sono solo 79 possibili frequenze su cui può trasmettere. Uno degli obbiettivi delle nuove versioni Bluetooth è proprio quello di ridurre questi tempi di ricerca.

Per problemi di privacy e di alimentazione, tutti i dispositivi Bluetooth hanno due opzioni capaci di determinare se il dispositivo risponde o meno alle richieste di individuazione degli altri dispositivi e ai tentativi di connessione, le quali sono configurabili, almeno in parte, nella maggior parte dei dispositivi. Nel dettaglio, l'opzione "inquiry scan" controlla il primo e l'opzione "page scan" controlla l'ultimo. Tuttavia, un dispositivo risulta in modalità rilevabile solo nel momento in cui l'inquiry scan è attivo.

Nel Bluetooth, sono previsti due tipi di canali per la trasmissione delle informazioni, ovvero:

- *i canali sincroni*, che permettono trasmissione simmetrica della voce nelle due direzioni con una velocità di 64 kb/s;
- *i canali asincroni*, che prevedono due diverse modalità di funzionamento; in particolare, nel caso di trasmissione asincrona non simmetrica è prevista una velocità di trasferimento massima di 723.2 kb/s in una direzione e 57.6 kb/s nell'altra; nel caso di trasmissione asincrona simmetrica la massima velocità di trasferimento è di 433.9 kb/s.

3.2 Scelta di un protocollo di trasporto

I protocolli Bluetooth si possono dividere in due grandi categorie:

- *protocolli di trasporto*; sono in realtà protocolli sviluppati appositamente per la tecnologia Bluetooth ed intervengono in tutti gli scambi di dati tra due dispositivi;
- *protocolli middleware*; comprendono, invece, sia protocolli specifici della tecnologia Bluetooth, che altri adottati per fornire supporto ad applicazioni già esistenti.

La Figura 3.3 mostra in dettaglio i vari protocolli dello stack. Il motivo dei diversi protocolli di trasporto risiede nel fatto che le diverse applicazioni hanno esigenze differenti.

I due fattori che distinguono i protocolli sono:

- *le garanzie*;
- *la semantica.*

La garanzia di un dato protocollo è definita dal grado di difficoltà tramite cui si può recapitare un pacchetto inviato. In particolare, un protocollo affidabile, come il TCP, ha l'idea "deliver-or-die", cioè o garantisce che tutti i pacchetti inviati vengono consegnati o muore (termina la connessione). Un protocollo best-effort, come l'UDP, tenta di recapitare i pacchetti e, in caso negativo, non ne indica la causa, cioè non dà alcuna garanzia dell'effettiva consegna dei dati né tantomeno dei livelli di qualità o priorità garantiti.

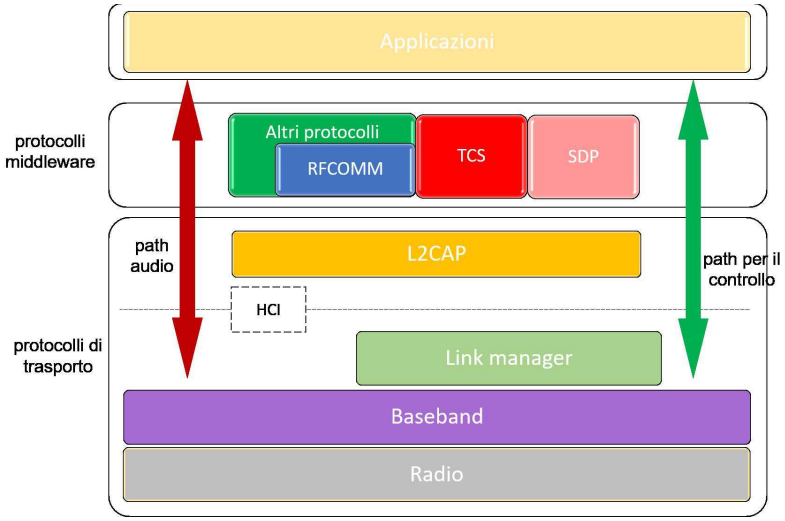

Figura 3.3. Panoramica dei protocolli di comunicazione

La semantica di un protocollo può essere sia packet-based che streams-based. Un protocollo basato su pacchetti, come l'UDP, invia i dati in singoli datagrammi di lunghezza massima prefissata, mentre un protocollo streams-based, come il TCP, invia i dati senza preoccuparsi di dove finisce un pacchetto e inizia quello successivo. Questa scelta è più una questione di preferenza rispetto a qualsiasi specifica esigenza perché un protocollo basato su pacchetti può essere utilizzato come un protocollo basato su flussi, e viceversa.

Il protocollo radio definisce le caratteristiche tecniche del sistema di trasmissione radio dei dispositivi Bluetooth; il suo modulo opera nella banda ISM (Industrial Scientific Medicine band) a 2.4GHz ed utilizza la tecnica del salto di frequenza per evitare interferenze tra dispositivi che iniziano contemporaneamente una trasmissione. Questa tecnica consiste nel cambiare periodicamente, in modo pseudo-casuale, la frequenza di trasmissione, scegliendo il canale su cui trasmettere tra uno dei 79 disponibili, a partire dalla frequenza di 2.402 MHz. Le 79 possibili frequenze, previste dallo standard, sono (2.402+k) MHz per k = 0, 1, 2,..., 78. Limitazioni sul range di frequenze sono presenti in varie nazioni; così, speciali algoritmi per il salto di frequenza sono progettati per i dispositivi a loro destinati. Per esempio, in Francia, i canali disponibili sono 23 e le relative frequenze sono (2.454+k) MHz con k=0, ..., 22. La sequenza di frequency hopping è unica per una rete Bluetooth ed è determinata dall'indirizzo del dispositivo Master della rete.

Il protocollo baseband definisce le procedure chiavi che rendono possibile la comunicazione tra i vari dispositivi di una rete Bluetooth, chiamata piconet. In particolare, il protocollo è responsabile della sincronizzazione delle unità Bluetooth, della selezione della frequenza di hopping, della correzione degli errori, del controllo del flusso e della sicurezza delle trasmissioni. Il protocollo definisce, inoltre, come le piconet vanno costruite, come sono strutturati i link e come possono essere trasmesse risorse che andranno, poi, condivise dai vari dispositivi.

Una piconet è una rete wireless che permette lo scambio di informazioni tra i dispositivi che la compongono. Una tale rete è costruita dinamicamente, senza alcun intervento esterno, man mano che nuovi dispositivi vengono aggiunti ad essa. Una piconet contiene almeno un dispositivo master e al più sette dispositivi slave, con cui il master comunica attivamente. I dispositivi sono sempre prodotti in modo da poter operare sia da master che da slave; la loro esatta modalità di funzionamento è decisa solo nel momento in cui la piconet è effettivamente costituita. Tra i vari compiti del dispositivo master troviamo:

- l'assegnamento agli slave, con cui comunica attivamente, di un indirizzo univoco all'interno della piconet; tale indirizzo viene detto active member address (AM_ADDR);
- l'istante di tempo in cui ciascuno slave presente nella rete può iniziare a trasmettere.

Oltre ai dispositivi slave, con il master possono essere registrati anche al più 255 dispositivi parked, cioé dispositivi che possono essere invitati, se necessario, a diventare attivi. Se un dispositivo Bluetooth non è associato ad una piconet, si dice che è in modalità stand-by.

Poiché le piconet usano radiofrequenze è possibile che, nel caso in cui esse siano tra loro vicine, uno stesso dispositivo possa essere contemporaneamente membro di due o più piconet; in tal caso si parla di scatternet (Figura 3.4).

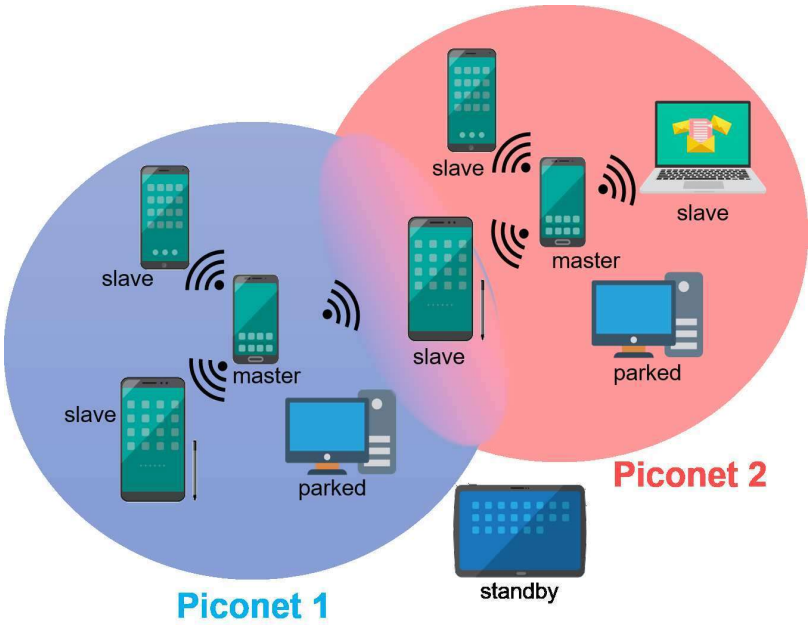

Figura 3.4. Due piconet che costituiscono uno scatternet

Il canale utilizzato in Bluetooth per le trasmissioni è definito come la sequenza pseudo-casuale dei salti di frequenza seguiti in modo sincrono dai membri della piconet. La sequenza dei salti è unica per una certa piconet ed è determinata dall'indirizzo Bluetooth (BD_ADDR) del dispositivo master e dal suo clock. Il canale è diviso in time slot dove l'inizio di ciascuno di essi coincide con un salto di frequenza. Il tasso nominale a cui si verificano i salti è di 1600 hops/sec, ovvero ogni slot ha una lunghezza di 0.625 ms. Gli slot sono numerati in funzione del clock del dispositivo master e il range della numerazione loro assegnata va da 0 a 227 -1.

Sia i dispositivi master che quelli slave possono trasmettere pacchetti solo in time slot. Per la trasmissione è utilizzato uno schema time-division duplex (TDD) dove master e slave si alternano nella trasmissione (Figura 3.5). Il master trasmette negli slot con numero pari, mentre gli slave in quelli con numero dispari.

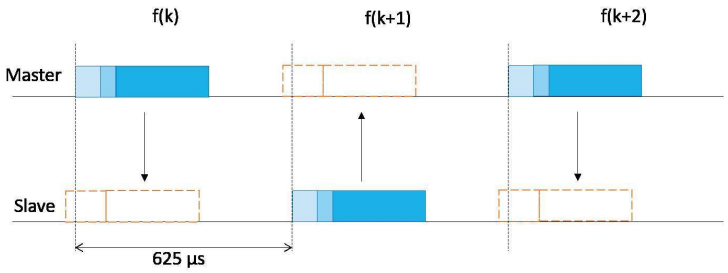

Figura 3.5. Schema time-division duplex

L'inizio della trasmissione di un pacchetto coincide sempre con l'inizio di uno slot e durante la trasmissione di un pacchetto non avvengono mai salti di frequenza.

Con il Bluetooth è possibile anche trasmettere pacchetti multislot, cioé pacchetti che possono occupare, viste le loro dimensioni, anche 5 slot, come mostrato in Figura 3.6. Durante la trasmissione di un tale tipo di pacchetto non avvengono salti di frequenza all'inizio di ciascuno slot, ma alla fine della trasmissione si parte con la frequenza che si sarebbe usata se fossero stati trasmessi solo pacchetti di dimensione pari ad uno slot.

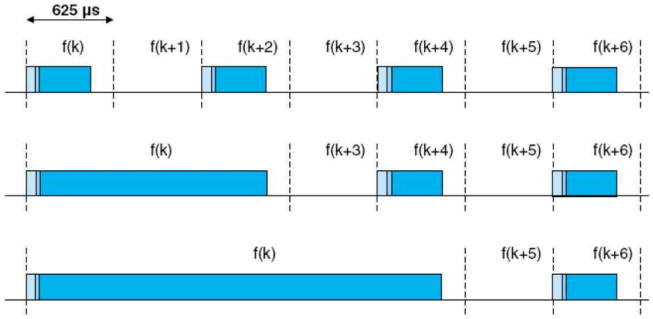

Figura 3.6. Pacchetti multislot

Un dispositivo slave può trasmettere solo dopo che ha ricevuto una trasmissione del dispositivo master. Nel caso di una scatternet, il nodo slave appartenente alle diverse piconet costituenti la scatternet può trasmettere solo in una piconet per volta.

Per mantenere la sincronizzazione dei salti di frequenza, i dispositivi slave sfruttano la conoscenza del clock del master e la lunghezza degli slot (0.625 ms). Da qui, in ogni momento, gli slave mantengono l'offset tra il proprio clock e quello del master.

3.2.1 RFCOMM

Il protocollo RFCOMM è un protocollo stream-based e fornisce un servizio affidabile con le stesse garanzie del TCP. Le specifiche Bluetooth affermano che esso è stato progettato per emulare porte seriali RS-232 (per rendere più facile per i produttori aggiungere funzionalità Bluetooth ai loro dispositivi con porta seriale esistente). Pertanto, si tratta di un protocollo di trasporto di uso generale che funziona bene per l'emulazione seriale di porte.

Dal punto di vista di un programmatore di rete, la scelta di numeri di porta rappresenta la maggiore differenza tra il TCP e l'RFCOMM. Infatti, il TCP, supporta fino a 65.535 porte aperte su una singola macchina, mentre l'RFCOMM ne supporta solo 30. Questo ha un impatto significativo nella modalità di scelta dei numeri di porta per le applicazioni server.

Un attributo distintivo di RFCOMM è che, a seconda dell'applicazione e della piattaforma di destinazione, talvolta costituisce la sola ed unica scelta possibile; alcuni ambienti, come, per esempio, le API Microsoft Windows XP Bluetooth e Nokia Series 60 Python, supportano solo il protocollo di trasporto RFCOMM, per l'appunto.

3.2.2 L2CAP

Il Logical Link Control and Adaption Protocol (L2CAP) è un protocollo basato su pacchetti che può essere configurato con diversi livelli di affidabilità. La dimensione massima del pacchetto di default è 672 byte, ma questo può essere aumentato fino a 65.535 byte dopo che la connessione è stata stabilita. L'L2CAP può essere confrontato con l'UDP, un protocollo basato su pacchetti best-effort. Ci sono, tuttavia, delle differenze tra i due; ad esempio, i casi di utilizzo dell'L2CAP sono molti di più rispetto a quelli dell'UDP in quanto, sebbene entrambi siano protocolli basati su pacchetti, L2CAP fa rispettare l'ordine di consegna. Per di più, nell'UDP, è possibile che, una volta trasmessi due pacchetti dallo stesso dispositivo, il secondo arrivi prima del primo a causa della peculiarità di Internet routing; ciò non avviene in L2CAP, dove i pacchetti vengono sempre consegnati nell'ordine con cui sono stati inviati. Un'altra differenza da sottolineare riguarda il fatto che l'UDP risulta limitato alle garanzie best-effort, mentre l'L2CAP può essere configurato per livelli di affidabilità variabile.

Tali differenze ci consentono di dire che L2CAP può essere usato sia dove viene utilizzato l'UDP per semplice comunicazione basata su pacchetti best-effort, sia per applicazioni in cui l'UDP non risulta adatto. In particolare, l'affidabilità

è raggiunta da un sistema di trasmissione/riconoscimento in cui possono o meno essere ritrasmessi pacchetti non riconosciuti. Ci sono tre possibili politiche di ritrasmissione:

- mai ritrasmettere (ma fare un best effort);
- sempre ritrasmettere fino al successo o al fallimento totale della connessione;
- abbandonare un pacchetto e passare agli altri in coda, qualora quest'ultimo non sia stato riconosciuto entro un certo limite di tempo specificato (0-1279 ms); ciò è utile soprattutto quando i dati devono essere trasmessi in modo tempestivo (e assume una best effort).

Queste abitudini, a prima vista, sembrano spesso utili. Tuttavia, ci sono alcuni problemi in cui si deve porre attenzione e che dovrebbero essere evitati.

Innanzitutto, regolando la semantica di consegna per una singola connessione ad un dispositivo remoto, tutte le connessioni L2CAP per quel dispositivo vengono influenzate, mentre le connessioni L2CAP ad altri dispositivi non sono interessate. In secondo luogo, L2CAP serve effettivamente come protocollo di trasporto per RFCOMM, per cui ogni connessione RFCOMM viene effettivamente incapsulata all'interno di una connessione L2CAP. Questo fatto, unito con il primo, indica che, modificando le garanzie di consegna per L2CAP, cambiano anche le garanzie di consegna per RFCOMM.

La preoccupazione principale riguarda il fatto che, cambiando l'abituale ritrasmissione dell'L2CAP, possano verificarsi delle conseguenze indesiderate e, nel caso in cui esse non siano accettabili, allora tale metodo di ritrasmissione può non essere molto utile.

3.2.3 ACL

Un altro protocollo è l'Asynchronous Connection-oriented Logical (ACL) protocol. Due dispositivi Bluetooth possono avere al massimo una sola connessione ACL tra di essi; tale connessione viene utilizzata per il trasporto di tutto il traffico L2CAP e RFCOMM. L'ACL è simile all'IP in quanto è un protocollo fondamentale che viene utilizzato raramente per trasportare direttamente i dati, ma è quasi sempre utilizzato per incapsulare pacchetti di protocollo di livello più alto.

3.2.4 SCO

L'ultimo protocollo di trasporto descritto è il Synchronous Connection-Oriented (SCO) logical transport protocol. Questo è un protocollo best-effort packet-based che viene utilizzato esclusivamente per trasmettere audio vocale di qualità e, più precisamente, qualsiasi audio con qualità di voce di 64 kb/s. La trasmissione di audio di qualità CD è inutile, perché la velocità di trasmissione non è abbastanza elevata per permetterlo, ma è comunque sufficiente per fare telefonate. Se si utilizza un auricolare Bluetooth, i dati vocali sono probabilmente trasmessi attraverso una connessione SCO, mentre se si utilizzano le cuffie Bluetooth per ascoltare il lettore MP3 Bluetooth l'audio viene probabilmente trasmesso su una connessione L2CAP.

I pacchetti SCO non sono affidabili e non vengono mai ritrasmessi, ma esiste una garanzia di qualità del servizio; difatti, è sempre garantito che una trasmissione

SCO abbia una velocità di trasmissione di 64 kb/s. Se altre applicazioni e connessioni su un dispositivo si contendono il tempo radio per trasmettere un file o per sincronizzare un calendario, alla connessione SCO verrà data una priorità. Al fine di assicurare tale garanzia a tutti i collegamenti SCO nessun dispositivo Bluetooth può avere più di tre connessioni SCO attive; inoltre due dispositivi Bluetooth possono avere al massimo un collegamento SCO tra di essi.

Le applicazioni che trasmettono dati, indipendentemente dal fatto che richiedano almeno una certa affidabilità sui pacchetti consegnati, devono quasi sempre utilizzare le connessioni L2CAP. SCO deve essere utilizzata solo per trasmettere audio vocale di qualità o audio di qualità ancora più elevata e mai per altri tipi di dati; questo protocollo è molto importante per il suo uso diffuso con auricolari Bluetooth.

In conclusione, è possibile affermare che ci sono quattro principali protocolli di trasporto definiti dalle specifiche Bluetooth. Di questi, RFCOMM è spesso la scelta migliore e, talvolta, l'unica scelta possibile; L2CAP è un protocollo di trasporto ampiamente utilizzato quando non è necessaria la natura di streaming di RFCOMM; ACL consente di trasportare traffico relativo a L2CAP e RFCOMM; SCO è un protocollo di trasporto altamente specializzato, che è stato progettato specificamente per l'audio e la voce di qualità.

Una connessione è rappresentabile tramite una scatola (come mostrato anche nella Figura 3.7), e la presenza di una scatola dentro un'altra scatola indica che i pacchetti per il collegamento interno sono incapsulati all'interno di pacchetti per il collegamento esterno.

Infine, due dispositivi Bluetooth possono avere al massimo un collegamento SCO e una connessione ACL tra di essi; invece, il numero di connessioni L2CAP e RFCOMM attivi è limitato soltanto dal numero di porte disponibili.

3.3 Porte

Una porta viene utilizzata per consentire a più applicazioni sullo stesso dispositivo di utilizzare contemporaneamente il medesimo protocollo di trasporto. Quasi tutti i protocolli di trasporto Internet di uso comune sono stati progettati con il concetto di numeri di porta, ed il Bluetooth non fa eccezione, nonostante si serva di una terminologia leggermente diversa. In L2CAP le porte sono chiamate Protocollo a Servizio Multiplexer (PSM) e il loro numero può assumere valori dispari compresi tra 1 e 32.767. In RFCOMM i canali 1-30 sono disponibili per l'uso. Molti protocolli di trasporto sono stati progettati pensando a specifiche applicazioni; di conseguenza, proprio all'inizio della progettazione del protocollo, alcune delle porte sono state messe da parte e riservate per tali applicazioni. Si prevede che le applicazioni personalizzate non utilizzino mai una qualsiasi porta nota, a meno che non stiano attuando un servizio standardizzato assegnato a quel numero di porta. Per esempio, nel protocollo TCP/IP, la porta 80 è riservata per il traffico Web, la porta 23 per la posta elettronica, e così via. Il Bluetooth non fa eccezione, e si riserva per l'L2CAP le porte 1-1023 per un utilizzo standardizzato.

Se il server è un'applicazione standardizzata che utilizza uno dei numeri di porta noti, allora la connessione avverrà su quella determinata porta; in caso contrario,

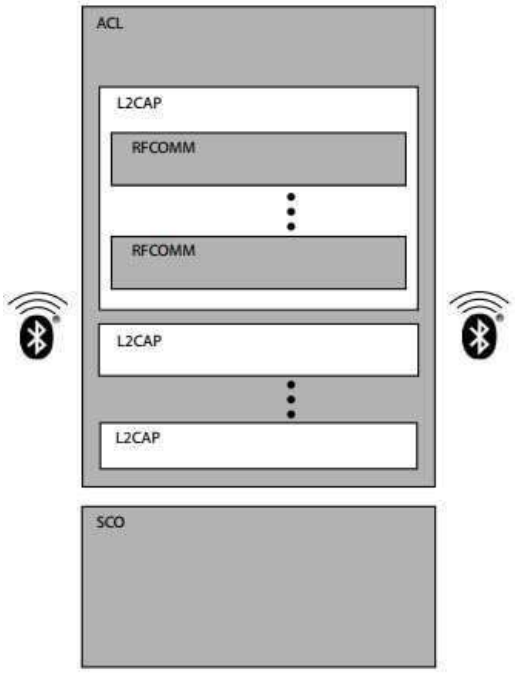

Figura 3.7. Protocolli Bluetooth

l'approccio tradizionale nella programmazione di Internet è quello di codificare un numero di porta in entrambe le applicazioni, client e server. Quando arriva il momento di stabilire la connessione, il server ascolta su una determinata porta, e il client si connette a quella porta. Il principale svantaggio di questo approccio è che non è possibile eseguire due applicazioni server che utilizzino entrambe lo stesso numero di porta; il Bluetooth cerca di evitare questo problema introducendo la SDP. La premessa fondamentale è che ogni dispositivo Bluetooth mantiene un server SDP in ascolto su un numero di porta ben noto: quando un'applicazione client remota si connette per la prima volta al dispositivo, fornisce una descrizione del servizio che sta cercando sul server SDP e il server SDP genera un elenco di tutti i servizi corrispondenti. Nelle Figura 3.8 viene illustra questa differenza.

La descrizione di un servizio che un'applicazione server registra con il proprio server SDP e che il server SDP trasmette ai client che richiedono l'informazione, viene indicata come un record di servizio detto, anche, record SDP. In teoria, il record di servizio consiste in un elenco di coppie attributo/valore, dove ogni attributo è un numero intero a 16 bit e ogni valore può essere un tipo di dati di base, come un numero intero o una stringa o un elenco di altri tipi di dati; pertanto, un record di servizio è essenzialmente un dizionario di varie voci che descrivono il servizio offerto. Accanto al numero di porta effettivo elencato per un servizio, probabilmente i due attributi più importanti in un record di servizio sono l'ID di servizio e l'Elenco ID delle classi di servizio, che sono quelli utilizzati più frequentemente da un client per

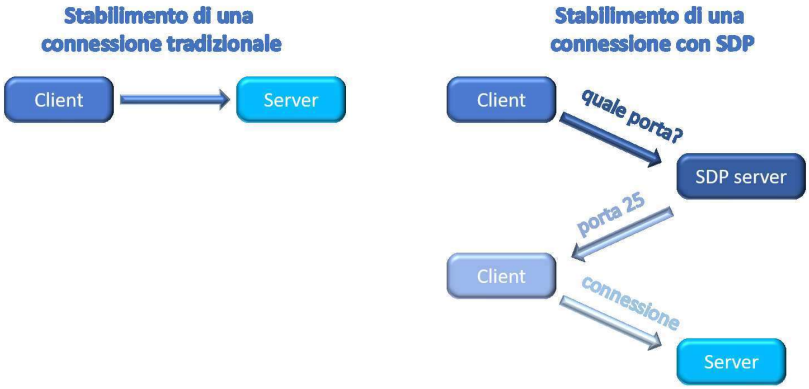

Figura 3.8. Confronto tra connessione tradizionale e con SDP

identificare effettivamente il servizio desiderato.

La struttura base di un record SDP è il data element. Un data element è costituito da un tipo, una dimensione e un valore. Il tipo di un data element può essere un tipo primitivo, come, ad esempio, numeri interi e stringhe, o una sequenza di data element. La dimensione indica quanto spazio (in byte) occupa il valore; infine, il valore è costituito dai dati effettivi da trasmettere.

3.4 Servizio ID

Il Bluetooth assegna ad ogni singolo servizio un identificatore univoco, e si assicura che il client conosca già l'ID univoco del servizio che sta cercando. In altre parole, anziché assegnare un numero di porta in fase di progettazione, gli sviluppatori assegnano un ID univoco in fase di progettazione. Il grande vantaggio derivante dall'utilizzo di un ID univoco al posto dei numeri di porta è che lo spazio degli ID univoci validi è progettato per essere molto più grande dello spazio dei numeri di porta validi, consentendo, così, agli sviluppatori di ridurre al minimo la possibilità di conflitti ID univoci.

L'Internet Engineering Task Force ha un metodo standard tramite cui gli sviluppatori possono creare autonomamente identificatori universali univoci a 128 bit (UUID). Questa è l'idea di base intorno a cui ruota l'SDP, e tale identificatore è chiamato Servizio ID. Nello specifico, uno sviluppatore sceglie questo UUID in fase di progettazione e quando viene eseguito il programma registra un record di servizio contenente il suo ID di servizio con il server SDP per quel dispositivo. Un'applicazione client che sta cercando di trovare un servizio specifico interrogherà il server SDP su ciascun dispositivo che trova al fine di vedere se esso offra servizi con lo stesso UUID. La notazione standard per un UUID è una serie di cifre separate dal trattino del modulo "XXXXXXXXXXXX-XXXXXXXX-XXXXXXXXXXXX", dove ogni "X" è una cifra esadecimale. Il primo segmento di otto cifre corrisponde ai bit 1-32 dell'UUID, mentre il segmento successivo di quattro cifre costituisce i bit 33-36, e così via.

3.5 Comunicare via socket

Un socket rappresenta il punto finale di un collegamento di comunicazione. Ci sono due modi generali per utilizzare un socket, ovvero come client o come server. La Figura 3.9 illustra i passaggi necessari per ottenere una connessione mediante socket. Il collegamento di un socket avviene attraverso la connessione di quest'ultimo ad un Bluetooth adapter e un numero di porta (bound); quando un socket del server è in modalità di ascolto (listening mode), i dispositivi remoti possono avviare la procedura di connessione; infine, dopo che un socket del server ha accettato una nuova connessione, genererà un nuovo socket connesso al dispositivo remoto.

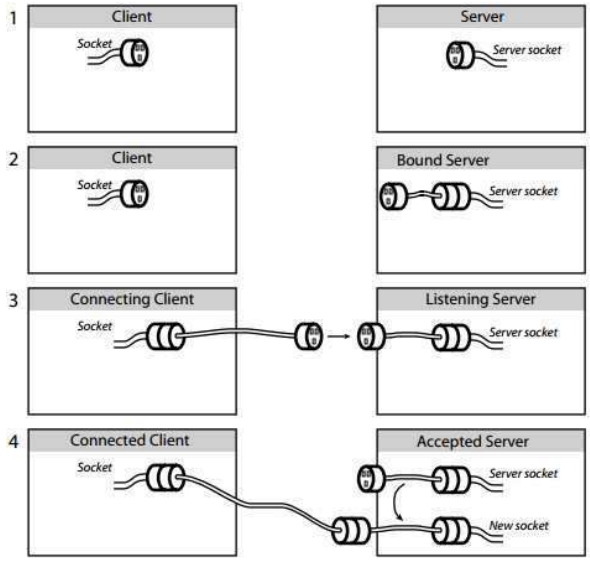

Figura 3.9. Utilizzo dei socket in una comunicazione client-server

Il primo step che un'applicazione deve compiere per avviare una comunicazione via socket consiste, proprio, nella creazione di un socket; questo avviene attraverso il comando `create()`, il quale, in genere, specifica il protocollo di trasporto da utilizzare.

I socket client sono abbastanza semplici, in quanto, una volta che il socket viene creato, è necessario soltanto un ulteriore passaggio per ottenere un socket connesso, e questo è possibile attraverso il comando `connect()` specificando l'indirizzo e la porta del dispositivo di destinazione. Il sistema operativo si occuperà, quindi, di tutti i dettagli di livello inferiore, riservando risorse sul Bluetooth adapter locale, ricercando il dispositivo remoto, formando una piconet e stabilendo una connessione. Solo una volta che il socket è collegato può essere utilizzato per il trasferimento dati.

Affinché il socket sia connesso, il socket server deve effettuare più passaggi rispetto al socket client. In primo luogo, l'applicazione deve collegare il socket alle risorse Bluetooth locali, specificando il Bluetooth adapter locale e il numero di porta

da utilizzare; in seguito, attraverso il comando `listen()`, viene posizionato il socket in modalità di ascolto. Questo indica al sistema operativo di accettare le richieste di connessione in entrata sul numero di Bluetooth adapter e la porta locale del socket con il quale è stato effettuato il bound. Infine, il comando `accept()` viene utilizzato per ottenere un socket collegato che può essere utilizzato per il trasferimento dei dati.

Una volta che un'applicazione Bluetooth ha un socket connesso, è possibile usarlo per la comunicazione attraverso i comandi di invio e ricezione dei dati, che sono, rispettivamente, `send()` e `receive()`. In seguito ad una `send()`, si riceve un messaggio di ritorno che indica la riuscita o meno dell'invio. Questo messaggio di ritorno significa solo che i byte sono stati spostati dallo spazio degli indirizzi dell'applicazione al buffer del sistema operativo; quindi il messaggio non è detto che abbia effettivamente lasciato il dispositivo.

Quando l'applicazione è terminata, per scollegare il socket, occorre usare il comando `close()`. La chiusura di un socket del server di ascolto chiude la porta e smette di accettare le connessioni in entrata.

3.6 Implementazione del Bluetooth nell'applicazione TrisBluetooth

Nonostante l'applicazione oggetto della presente tesi venga sviluppata attraverso `Xamarin.Forms`, con lo scopo di sviluppare un'applicazione ibrida, l'implementazione del Bluetooth è stata effettuata in modo nativo poichè, tra i dispositivi della Apple e quelli Android, vi è una notevole differenza nell'implementazione di questa tecnologia. Infatti, ci sono molti pacchetti NuGet destinati all'inplementazione del Bluetooth cross-platform ma riguardano tutti la tecnologia Bluetooth LE, quindi non sono adatti per l'applicazione che si vuole realizzare.

Nel Listato 3.1 viene mostrata l'interfaccia `IBluetooth.cs` la quale verrà, poi, implementata nativamente in ogni piattaforma.

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;

namespace TrisBluetooth
{
    public interface IBluetooth
    {
        void StartDiscovery();

        void Cancel();

        void CreateBond(string Mac);
    }
}
```

Listato 3.1. Interfaccia `IBluetooth.cs`

Nel Listato 3.2 è possibile visionare come l'interfaccia `IBluetooth.cs` è stata implementata per i dispositivi Android.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.Bluetooth;
using Java.Util;
using System.Threading.Tasks;
using Java.IO;
using TrisBluetooth.Droid;
using System.Threading;
using System.Collections.ObjectModel;
using Java.Lang;
using Java.Nio.Charset;

[assembly: Xamarin.Forms.Dependency(typeof(Bth))]
namespace TrisBluetooth.Droid
{
    public class Bth : IBluetooth
    {
        public static BluetoothAdapter mBluetoothAdapter;
        public static BluetoothDevice serverDevice;


        public Bth()
        {
            mBluetoothAdapter = BluetoothAdapter.DefaultAdapter;

        }

        public void Cancel()
        {
            System.Console.WriteLine("Ricerca disattivata");
            if (mBluetoothAdapter.IsDiscovering)
            {
                mBluetoothAdapter.CancelDiscovery();
            }
        }

        public void CreateBond(string Mac)
        {
            Cancel();
            foreach (BluetoothDevice device in MainActivity.devices)
            {
                if (device.Address.Equals(Mac))
                {
                    serverDevice = device;
                    device.CreateBond();
                    break;
                }
            }
        }

        public void StartDiscovery()
        {
            if (mBluetoothAdapter.IsDiscovering)
            {
                Cancel();
                System.Console.WriteLine("Ricerca attivata");
                mBluetoothAdapter.StartDiscovery();
            }
            else
            {
                System.Console.WriteLine("Ricerca attivata");
                mBluetoothAdapter.StartDiscovery();
            }

        }

    }
}
```

Listato 3.2. La classe `Bth.cs` che implementa in Android l'interfaccia `IBluetooth.cs`

Di seguito verrà analizzata in dettaglio la classe `Bth.cs` che implementa, in Android, l'interfaccia `IBluetooth.cs`:

- `Cancel()`: ha lo scopo di disattivare la ricerca dei dispositivi Bluetooth.
- `CreateBond(string Mac)`: riceve come argomento la stringa che identifica il MAC del dispositivo con il quale il primo giocatore vuole intraprendere una partita; successivamente, ricerca nella lista dei dispositivi limitrofi, il dispositi-

vo con il MAC, specificato e, una volta determinato quest'ultimo, procede alla connessione tra esso e il dispositivo del primo giocatore.
- `StartDiscovery()`: serve ad attivare la ricerca dei dispositivi Bluetooth limitrofi; se una ricerca era stata già avviata, verrà interrotta per avviarne una nuova.

`Bth.cs` verrà, poi, utilizzata, attraverso l'interfaccia `IBluetooth.cs` nel progetto cross-platform. Per mezzo di MessagingCenter verranno messe in "comunicazione" la parte nativa con quella cross platform dell'applicazione.

Per maggiori dettagli sull'invio e recezione dei messaggi attraverso il Bluetooth, la definizione del dispositivo client e server e altro, si rimanda al Capitolo 5 completamente dedicato all'implementazione dell'applicazione TrisBluetooth.

4

Analisi dei requisiti e progettazione

In questo capitolo verrà illustrato il mondo del Mobile Gaming, nonchè la sua importanza nel mercato delle applicazioni, successivamente, dopo aver analizzato i requisiti necessari per lo sviluppo dell'applicazione TrisBluetooth, verranno illustrati i diagrammi delle attività e dei casi d'uso e i mockup di livello 0 e di livello 1.

4.1 Mobile Gaming

Con il termine "Mobile Gaming" si intende tutto il mondo che ruota intorno al gioco mobile, ovvero alle applicazioni che offrono ai consumatori esperienze di gioco su telefono cellulare, smartphone, tablet, smartwatch, PDA, lettore multimediale portatile o calcolatrice grafica.

4.1.1 Importanza del Mobile Gaming

Nel 2019 molti articoli riguardanti il mondo del Mobile Gaming riportano l'aumento di utilizzo di questa tipologia di applicazioni nonchè i crescenti introiti ricavati da tale attività.

Il Mobile Gaming ha registrato un incremento record nel secondo trimestre del 2019, con oltre 30,3 miliardi di download di app e quasi 22,6 miliardi di dollari di spesa dei consumatori su iOS e Google Play. I giochi rappresentavano il 35% dei download globali nel trimestre, ma quasi il 75% della spesa dei consumatori su iOS e Google Play. Nella Figura 4.1 è possibile visualizzare quanti download di applicazioni di gioco sono stati registrati in Google Play e iOS App Store nel secondo trimestre del 2019

Su Google Play, la spesa dei consumatori nei giochi è cresciuta del 20% annuo, mentre su iOS tale spesa è cresciuta di oltre il 15% annuo. Nella Figura 4.2 si può vedere quanto i consumatori hanno speso nelle applicazioni di gioco nei diversi store nel secondo semestre del 2019.

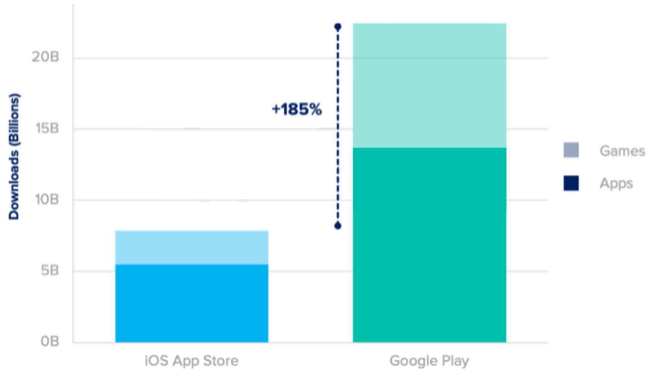

Figura 4.1. Download di applicazioni di gioco nel secondo trimestre del 2019

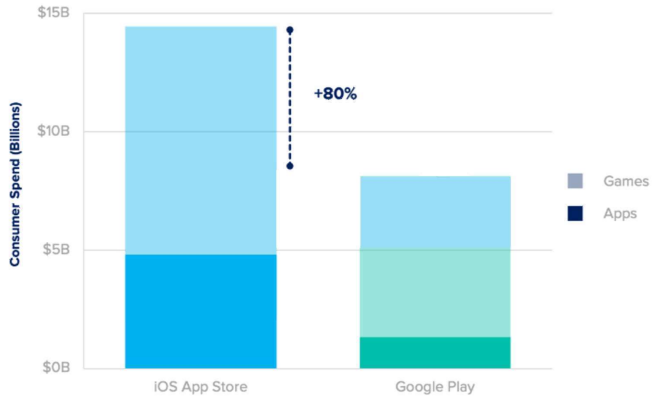

Figura 4.2. Spesa dei consumatori nelle app di gioco nel secondo trimestre del 2019

4.1.2 Connettività nel Mobile Gaming

La connettività nelle applicazioni di gioco rappresenta sempre di più una consuetudine; questo poichè si ha sempre più bisogno di connessioni tra dispositivi e con server per offrire esperienze di gioco sempre più avvincienti. Con il termine "connettività" non si indica solo quella Internet ma anche quelle attraverso altre tecnologie, come, ad esempio, il Bluetooth.

Le applicazioni che utilizzano il Bluetooth hanno lo scopo di essere utilizzate in ambienti in cui la connettività Internet è assente o non permessa, come, ad esempio, durante un viaggio aereo, ma anche per far risparmiare i consumatori in merito all'utilizzo dei dati mobili.

YouTube Go su Android ne è un esempio; infatti quest'applicazione offre una funzionalità molto importante sottoforma di un'opzione offline; gli utenti sono in grado di visualizzare l'anteprima dei video, nonchè di scegliere una risoluzione per il download o lo streaming, per risparmiare tempo quando connessi a dati o al Wi-Fi; inoltre, la condivisione è semplificata grazie alla possibilità di condividere un video tramite Bluetooth, senza dover essere necessariamente connessi ad Internet.

4.2 Requisiti

Per la progettazione dell'applicazione oggetto della presente tesi è opportuno definire con chiarezza quali sono i requisiti poichè questi costituiscono il mio punto di partenza. Tutti i requisiti descriveranno ciò che l'applicazione dovrà fare, ma non come dovrà farlo.

I requisiti dell'applicazione oggetto della presente tesi possono essere raggruppati nei seguenti macro insiemi:

- *autorizzazioni*;
- *ricerca*;
- *istruzioni di utilizzo*;
- *griglia di gioco*;
- *user Experience Design*.

Ciascuno di essi sarà illustrato in dettaglio nella prossime sottosezioni.

4.2.1 Autorizzazioni

Le applicazioni, per funzionare, devono interagire con il dispositivo per raccogliere alcune informazioni specifiche che consentono l'utilizzo di alcune funzioni. Se, ad esempio, si desidera inviare un messaggio su un contatto su WhatsApp, l'app avrà bisogno di accedere alla rubrica.

Quando si scarica una nuova applicazione dal Play Store, in alcuni casi viene chiesto l'accesso ad alcune autorizzazioni fin da subito perché necessarie per poter procedere con l'installazione del servizio. Non sempre, però, si presenta questa situazione perché alcune applicazioni si installano subito richiedendo alcune autorizzazioni specifiche successivamente, durante l'utilizzo.

Una delle novità più interessanti introdotte da Marshmallow è senza dubbio, la possibilità di selezionare i permessi singolarmente al momento dell'utilizzo di un'applicazione. I permessi delle app non devono più essere letti e gestiti tutti al momento del download dell'applicazione, come avveniva precedentemente, ma singolarmente, per ogni azione che li richiederà.

Google classifica alcune autorizzazioni come "pericolose"; tra queste si trovano anche le seguenti:

- *Calendario*: simile all'autorizzazione SMS, quella del calendario è divisa in accesso in lettura e in scrittura. Ovviamente, queste autorizzazioni hanno senso solo per le app calendario di terze parti.
- *Fotocamera*: le applicazioni collegate alla fotocamera usano queste autorizzazioni per aprire, chiudere e ordinare le immagini sul dispositivo. Teoricamente un'app dannosa potrebbe scattare foto autonomamente e diffonderle successivamente sul web ma, di norma, tutte le foto salvate sul dispositivo Android dovrebbero essere al sicuro.
- *Contatti*: come è facile intuire, con questa autorizzazione l'app potrà accedere alla lista dei contatti. Le app di messaggistica, quelle dedicate agli SMS ed i social network necessitano di questo permesso per poter funzionare, ma per altre app questa autorizzazione è totalmente inutile.

- *Sensori*: app dedicate allo sport o alla salute spesso richiedono la misurazione della frequenza cardiaca. Se il dispositivo dispone di un sensore dedicato, occorre sbloccarlo per poterlo sfruttare nell'applicazione.
- *Microfono*: questa autorizzazione consente la registrazione audio.
- *SMS*: questo permesso autorizza l'app ad inviare SMS. Anche in questo caso è d'obbligo considerare attentamente l'affidabilità dell'applicazione che richiede questo permesso, perchè, in caso di utilizzo non ortodosso, si potrebbe incorrere in servizi a pagamento indesiderati.
- *Telefono*: questa opzione consente all'app di comporre numeri telefonici senza input diretto da parte dell'utente. App come Skype o Facebook Messenger necessitano di questa opzione per svolgere la loro funzione telefonica, ma altre applicazioni fraudolente potrebbero utilizzare questa autorizzazione per comporre numeri a pagamento ad insaputa dell'utente. Anche alcune app dedicate alla musica necessitano, tuttavia, di questa autorizzazione per permettere la ricezione di una chiamata durante la riproduzione di un brano.
- *Memoria*: con questa autorizzazione le app hanno accesso allo spazio di archiviazione condiviso. Le sotto-autorizzazioni consentono l'accesso in lettura o in scrittura. Alcune app utilizzano questo permesso per scrivere i dati dell'utente nella memoria condivisa.
- *Localizzazione*: tramite questo permesso un'app potrà sempre risalire alla posizione geografica del dispositivo in modo preciso. Questo è fondamentale per i navigatori e le app che si basano sulla geolocalizzazione, ma sta diventando importante anche per i social network. In ogni caso, questa autorizzazione verrà usata spesso anche per pubblicità "mirate"e per promuovere servizi o prodotti accessibili nella zona in cui l'utente si trova.

È possibile modificare queste autorizzazioni in qualsiasi momento, semplicemente accedendo alle impostazioni del dispositivo. La disattivazione di alcuni permessi, però, può limitare il funzionamento dell'app stessa.

Per fare un esempio, sono diversi i permessi richiesti da WhatsApp. Questo non significa, però, che occorre accettarli tutti; per quanto riguarda le app di messaggistica, per esempio, è possibile disattivare l'accesso alla fotocamera.

A partire da Android 8.1, diverse autorizzazioni sono state classificate come "normali". Questo significa che le app le richiedono per essere installate e l'utente non può modificarle dopo l'installazione. Tali autorizzazioni sono le seguenti:

- stato della rete;
- direttiva notifiche;
- amministrazione Bluetooth;
- cambio dello stato della rete;
- blocco tastiera aperto;
- Internet;
- blocco dei processi in background;
- NFC;
- disabilitazione delle ottimizzazioni della batteria;
- cambio dello sfondo;
- utilizzo del sensore d'impronte digitali.

L'applicazione oggetto della presente tesi ha bisogno di due permessi da parte dell'utente; il primo riguarda, naturalmente, l'attivazione del Bluetooth mentre il secondo riguarda il permesso di localizzazione, necessario per usufruire della tecnologia Bluetooth.

La prima autorizzazione richiesta è quella relativa alla localizzazione che avviene all'avvio dell'applicazione, mentre la seconda, quella relativa al Bluetooth, viene richiesta una volta cliccato nel pulsante "Cerca".

4.2.2 Ricerca

Per avviare il gioco occorre connettere i dispositivi dei due giocatori e, per far questo, occorre avviare una ricerca e far comparire, per mezzo di una lista, i nomi dei dispositivi limitrofi che hanno il Bluetooth attivo. Questo permette di far selezionare all'utente il nome del dispositivo con il quale vuole connettersi per effettuare il bond ed, infine, instaurare la connessione.

4.2.3 Istruzioni di utilizzo

Una caratteristica essenziale che tutte le applicazioni devono possedere è la facilità di utilizzo. L'utente necessita di essere guidato in ogni passaggio dell'applicazione e questo può avvenire o per mezzo di una guida introduttiva, che illustra all'utente tutte le funzionalità dell'app, la quale appare solo al primo avvio, oppure per mezzo di guide, selezionabili su richiesta dall'utente.

L'applicazione TrisBluetooth utilizza quest'ultimo metodo, ovvero per mezzo di pulsanti posti in ogni schermata dell'applicazione, è possibile aprire un alert che mostra le istruzioni di utilizzo.

4.2.4 Griglia di gioco

TrisBluetooth, basandosi sul gioco di società Tris, necessita di una griglia di nove caselle da riempire con X o O a seconda di quale giocatore decide di occupare quella determinata cella.

Oltre alla griglia si ha bisogno di un tabellone che definisca il punteggio proprio e quello dell'avversario. Naturalmente il punteggio deve essere visibile per tutta la durata del gioco e non deve azzerarsi all'avvio di una rivincita.

Naturalmente, se una cella è occupata da un segno non potrà essere occupata nuovamente dallo stesso segno nè, tanto meno, dal segno opposto.

4.2.5 User Experience Design

Lo User Experience Design (UX Design) è lo sforzo intenzionale di un designer, o di un team di progettazione, per curare un'esperienza utente, sia online che offline; esso comprende molteplici caratteristiche, dalla semplice usabilità di un prodotto al livello di soddisfazione che gli utenti provano quando interagiscono con esso.

Anche se risale al 2000, la struttura di "The Elements of UX" di Jesse James Garrett è ancora utile per comprendere la disciplina dello UX; la sua illustrazione

chiara e concisa (Figura 4.3) mostra che lo UX include diversi livelli i quali vanno dal concepimento al completamento e, quindi, dall'astratto (base dell'iceberg) al concreto (apice dell'iceberg).

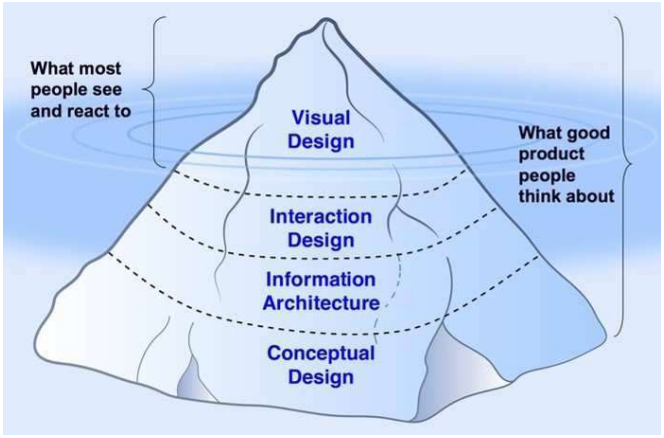

Figura 4.3. "The Elements of UX" di Jesse James Garrett

Il design dell'applicazione TrisBluetooth è stato progettato in modo da rendere la grafica vivace e giocosa, avendo l'applicazione TrisBluetooth uno scopo prettamente ludico.

Già dal logo dell'applicazione, mostrato nella Figura 4.4, si è voluto mettere in evidenza l'oggetto della stessa; infatti, si può notare la griglia caratteristica del gioco Tris ed, in alto a destra, si può visualizzare il logo del Bluetooth, il quale mette subito in evidenza la caratteristica principale dell'applicazione ovvero il fatto che il gioco avviene per mezzo di una connessione Bluetooth e non richiede, quindi, l'utilizzo di dati mobili o comunque di una connessione Internet.

Figura 4.4. Logo TrisBluetooth

Le altre pagine hanno uno sfondo che richiama il colore del logo e il font delle scritte ricorda una calligrafia manuale.

È possibile vedere il design dell'applicazione negli screenshot visibili, nel Capitolo dedicato al Manuale Utente (Capitolo 6).

4.3 Progettazione

La progettazione permette, attraverso diagrammi, di comprendere le funzionalità che l'applicazione deve possedere e come le diverse operazioni dell'utente vengono gestite.

4.3.1 Diagramma dei casi d'uso

Il diagramma dei casi d'uso si può, semplicemente, definire come una rappresentazione dell'interazione dell'utente con il sistema che mostra la relazione tra l'utente stesso e i differenti casi d'uso (use case) in cui egli è coinvolto. Il diagramma può identificare i diversi tipi di utenti di un sistema e i diversi casi d'uso ed essere spesso associati ad altri tipi di diagrammi.

In questi diagrammi si individuano i casi d'uso principali del sistema che vengono, poi, descritti in dettaglio tenendo conto delle sequenze di interazioni fra l'utente, che può essere un operatore umano o un sistema esterno, ed il sistema oggetto di analisi. Se si considerano i singoli caso d'uso, essi sono una successione di richieste o interazioni elementari compiute dall'utente al sistema insieme alle corrispettive risposte da parte di quell'ultimo.

In uno Use Case Diagram si possono individuare:

- *Gli attori*: questi hanno un comportamento o un ruolo e possono essere identificati da una persona, un altro sistema o un'organizzazione.
- *Lo scenario*: esso rappresenta una sequenza di azioni e interazioni tra gli attori e il sistema.
- *Use case*: questo denota una collezione di scenari di successi e fallimenti, descrivendo gli attori che usano il sistema per raggiungere un obiettivo.

Formalmente si può definire un caso d'uso come una sequenza di operazioni eseguita da un attore che interagisce con il sistema, permettendo di stabilire con precisione, quali sono i servizi richiesti dall'attore, i servizi forniti dal sistema, quali sono gli utenti che possono fare delle richieste e, quindi, ottenere dei servizi e, infine, quali sono le limitazioni nell'erogazione di servizi.

Un attore può essere di due tipi:

- *attivo*, quando inizia uno use case;
- *passivo*, quando partecipa a uno use case, ma non lo inizia

Il diagramma dei casi d'uso relativo all'applicazione TrisBluetooth (Figura 4.5) mostra le operazioni che possono essere effettuate dall'attore, in questo caso il giocatore.

Le operazioni che il giocatore può compiere sono:

- la visualizzazione delle info;

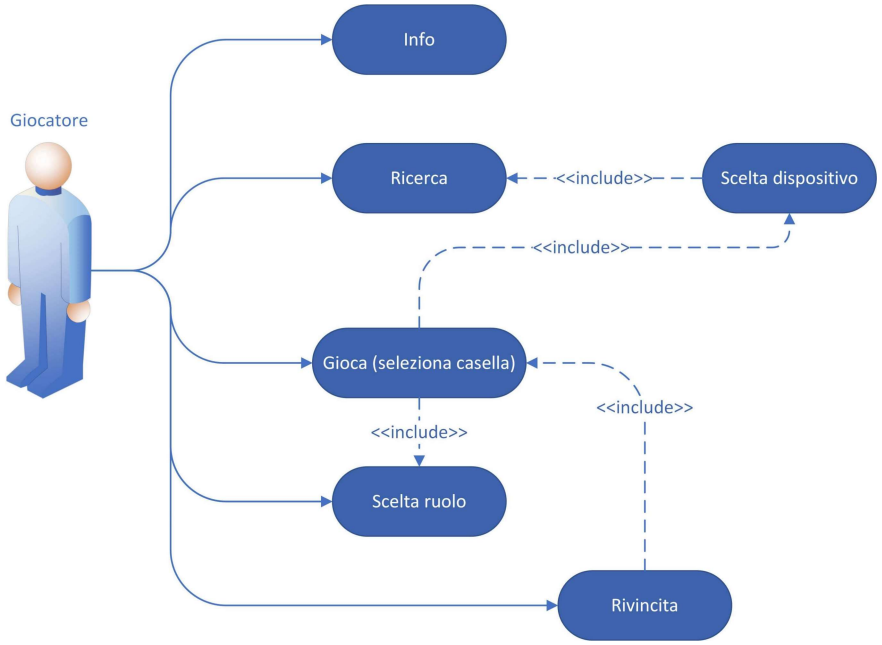

Figura 4.5. Diagramma dei casi d'uso dell'applicazione TrisBluetooth

- la ricerca dei dispositivi Bluetooth limitrofi;
- la scelta del ruolo (server o client) identificata con i simboli X o O;
- la selezione di una casella di gioco, che implica la scelta del dispositivo contro il quale giocare e la scelta del ruolo;
- la richiesta di una rivincita (questa operazione non è detto che si verifichi).

4.3.2 Diagramma delle attività

Nei diagrammi delle attività l'attenzione si sposta sulle attività o azioni che gli attori del caso d'uso compiono; non viene specificato di solito chi compie l'azione o come si compie, ossia come avviene l'interazione tra i soggetti coinvolti. Il diagramma indica le componenti dell'attività mostrando la successione logico-temporale delle azioni, la cronologia delle stesse e il loro svolgimento.

Di seguito verranno elencati gli elementi di base dei diagrammi delle attività:

- *Attività (Activity)*: operazione scomponibile all'interno di un sistema di stati.
- *Azione (Action)*: operazione non scomponibile che restituisce un risultato o un cambiamento del sistema.
- *Stato di azione (Action state)*: uno stato che indica l'esecuzione di un'azione (invoca un'operazione).
- *Stato di attività (Activity state)*: uno stato non atomico composto da più stati, che possono essere stati di attività e stati d'azione.

La Figura 4.6 mostra il diagramma delle attività dell'applicazione TrisBluetooth.

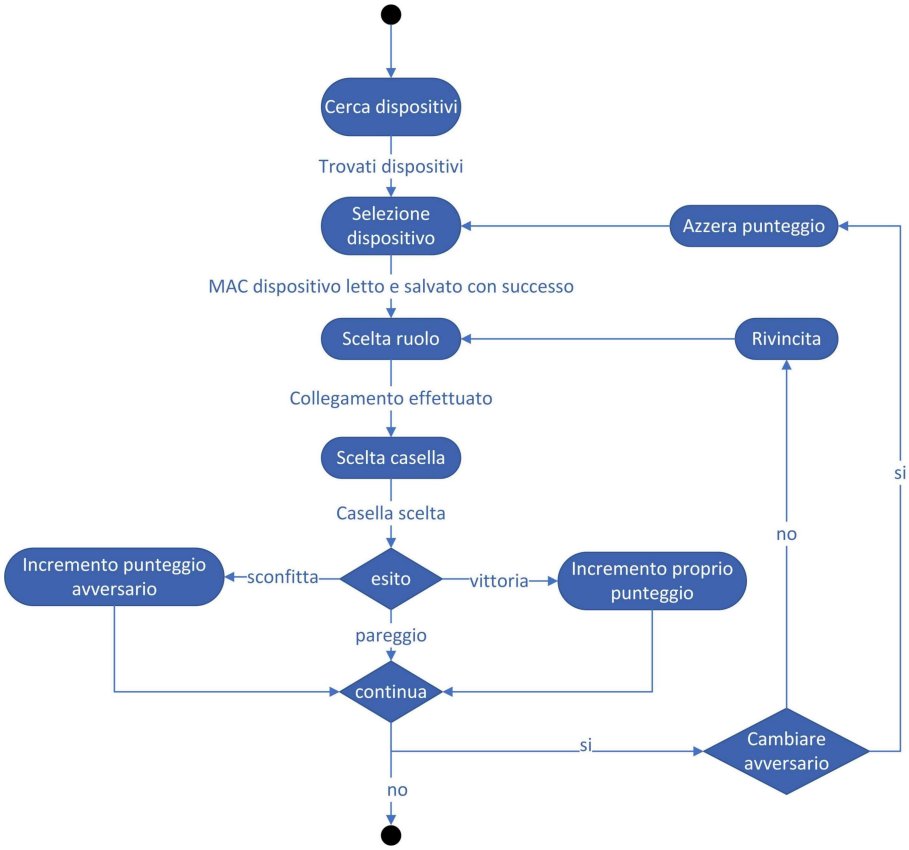

Figura 4.6. Diagramma delle attività dell'applicazione TrisBluetooth

Nel diagramma è possibile vedere cosa succede ad ogni operazione che l'utente svolge nell'applicazione, sia nel caso di riuscita che nel caso di fallimento.

4.3.3 Mockup

Per presentare un progetto al cliente, il quale non possiede, nella maggior parte dei casi, nessuna conoscenza in ambito di programmazione mobile, si realizzano dei mockup che forniscono un'idea di come dovrà essere l'applicazione.

I mockup hanno diversi livelli di dettaglio; i mockup di livello 0 forniscono una struttura generale dell'applicazione, mentre quelli di livello 1 presentano un maggiore dettaglio.

Di seguito verranno mostrati i mockup di livello 0 e 1 dell'applicazione TrisBluetooth.

All'avvio dell'applicazione l'utente si troverà la Splash Page; le Figure 4.7 e 4.8 mostrano, rispettivamente, il mockup di livello 0 e quello di livello 1 della splash page dell'applicazione TrisBluetooth.

Successivamente, l'utente dovrà accettare l'autorizzazione Bluetooth (Figure 4.9 e 4.10) per poter ottenere la lista dei dispositivi Bluetooth limitrofi, come mostrato nella Figure 4.12 e 4.13 che rappresentano, rispettivamente, i mockup di livello 0 e di livello 1 delle lista dei dispositivi Bluetooth limitrofi della pagina Gestione Bluetooth. Per comprendere il funzionamento l'utente può visualizzare le informazioni della pagina Gestione Bluetooth (come mostrato in Figura 4.11).

In seguito, verrà visualizzata la pagina di gioco dell'applicazione (Figure 4.14 e 4.15) e le rispettive istruzioni (Figura 4.16).

Infine l'ultimo dettaglio da mostrare è la pagina di gioco dell'applicazione durante una partita; i mockup di livello 0 e di livello 1 sono, rispettivamente, nella Figure 4.17 e 4.18.

A fine partita i giocatori possono scegliere se richiedere la rivincita, come mostrato nelle Figure 4.19 e 4.20.

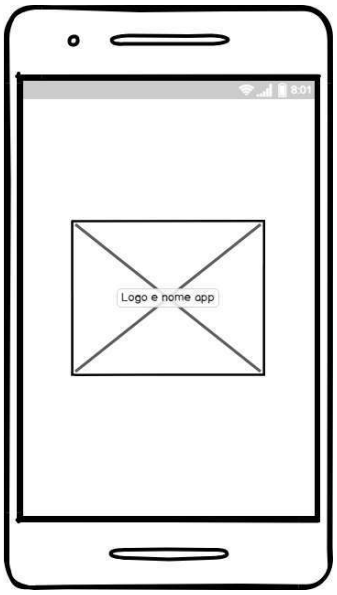

Figura 4.7. Mockup di livello 0 della Splash page dell'applicazione TrisBluetooth

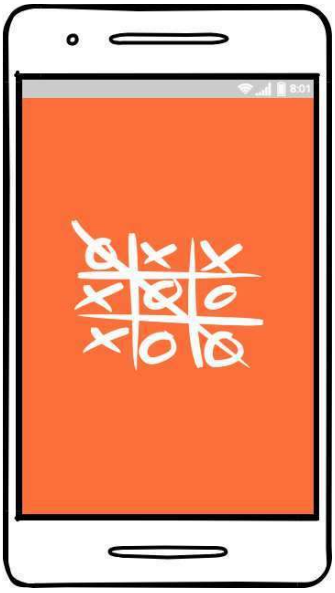

Figura 4.8. Mockup di livello 1 della Splash page dell'applicazione TrisBluetooth

Figura 4.9. Mockup di livello 0 della richiesta di autorizzazione Bluetooth

Figura 4.10. Mockup di livello 1 della richiesta di autorizzazione Bluetooth

Figura 4.11. Mockup di livello 1 delle informazioni della pagina di Gestione Bluetooth

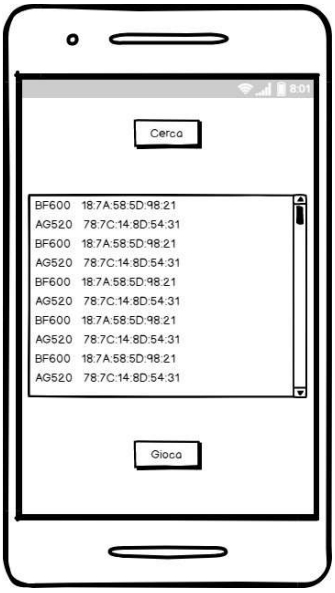

Figura 4.12. Mockup di livello 0 delle lista dei dispositivi Bluetooth limitrofi della pagina di Gestione Bluetooth dell'applicazione TrisBluetooth

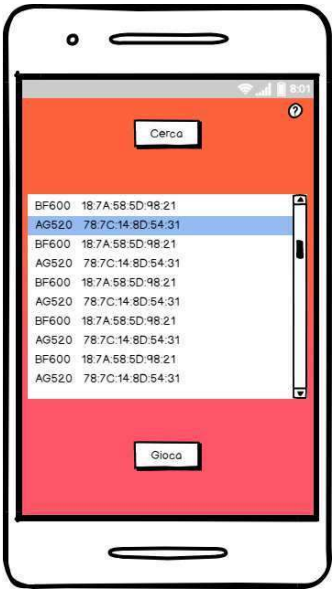

Figura 4.13. Mockup di livello 1 delle lista dei dispositivi Bluetooth limitrofi della pagina di Gestione Bluetooth

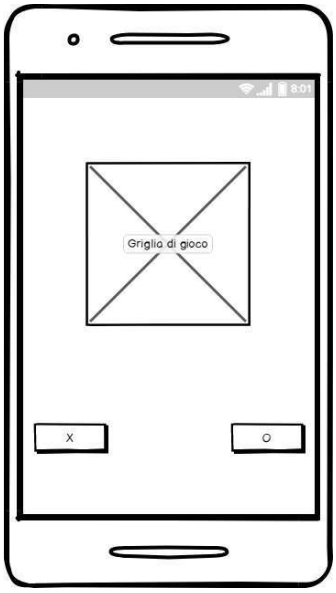

Figura 4.14. Mockup di livello 0 della pagina di gioco

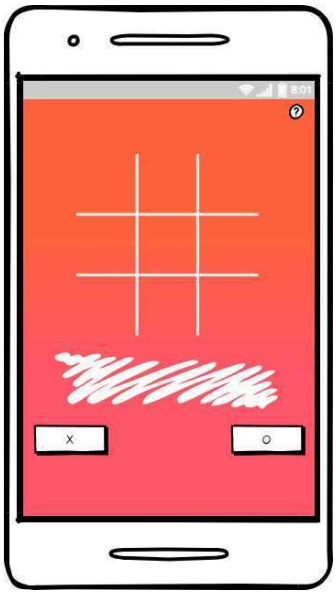

Figura 4.15. Mockup di livello 1 della pagina di gioco

Figura 4.16. Mockup di livello 1 delle informazioni della pagina di gioco

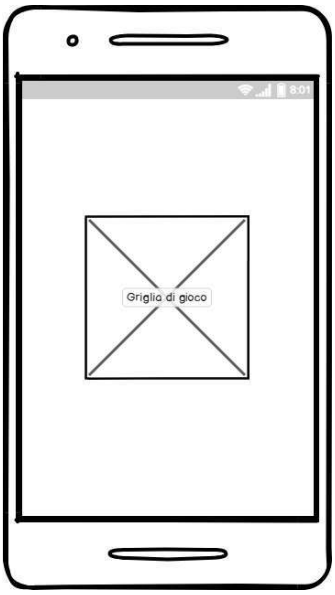

Figura 4.17. Mockup di livello 0 della pagina di gioco durante una partita

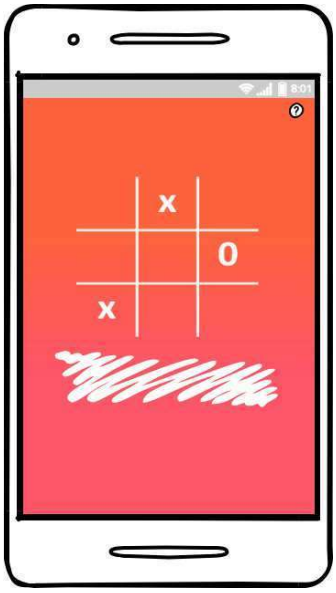

Figura 4.18. Mockup di livello 1 della pagina di gioco durante una partita

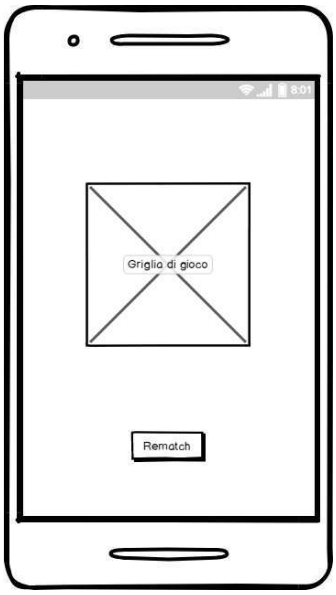

Figura 4.19. Mockup di livello 0 della pagina di gioco a fine partita

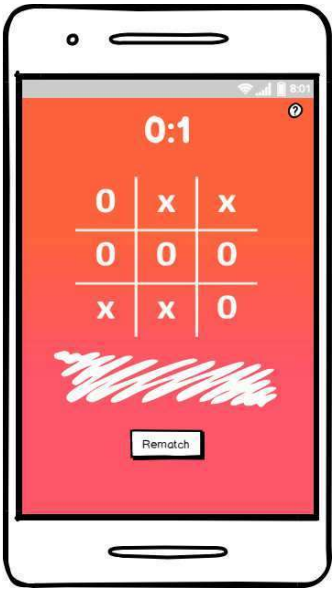

Figura 4.20. Mockup di livello 1 della pagina di gioco a fine partita

5

Implementazione

In questo capitolo verrà illustrata l'implementazione di tutto il progetto, analizzando ogni file che lo compone.

5.1 Implementazione di `SplashPage.xaml.cs`

Nel Listato 5.1 è possibile visualizzare come è stato implementato il file `SplashPage.xaml.cs`. Come sarà in seguito visibile nel Listato 5.20, la Splash Page è la prima pagina che apparirà all'avvio dell'applicazione TrisBluetooth.

```
using Android.OS;
using FFImageLoading.Svg.Forms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;


namespace TrisBluetooth
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class SplashPage : ContentPage
    {


        public SplashPage()
        {

            InitializeComponent();

            Handler h = new Handler();
            Action start = () =>
              {
                  var page = new MainPage();
                  Navigation.PushAsync(page);
              };
            h.PostDelayed(start, 13000);

        }

    }
}
```

Listato 5.1. Il codice relativo a `SplashPage.xaml.cs`

Una volta invocato il costruttore `SplashPage()`, si nota immediatamente la chiamata a `InitializeComponent()`, il quale carica il file XAML relativo alla Splash Page.

L'handler, essendo il miglior modo per comunicare tra background e UI thread, viene usato in modo tale da far si che l'immagine che raffigura un'animazione del logo dell'applicazione venga mostrata per una durata di 13 secondi per poi effettuare il passaggio alla pagina di Gestione Bluetooth.

La classe `NavigationPage` offre un'esperienza di navigazione gerarchica in cui l'utente è in grado di scorrere le pagine avanti e indietro in base alle esigenze. La classe implementa la navigazione come stack LIFO (Last-In, First-Out) di oggetti Pagina. Per passare da una pagina all'altra, un'applicazione esegue il push di una nuova pagina nello stack di navigazione, in cui diventa la pagina attiva, come illustrato nella Figura 5.1.

Figura 5.1. Pushing di una nuova pagina dallo stack di navigazione

Per tornare alla pagina precedente, l'applicazione preleva la pagina corrente dallo stack di navigazione e la nuova pagina in primo piano diventa la pagina attiva, come illustrato nella Figura 5.2.

Figura 5.2. Popping di una nuova pagina dallo stack di navigazione

Proprio per quanto appena riportato, attraverso `Navigation.PushAsync(page)`, dalla Splash Page si passa alla Main Page; infatti viene invocato `MainPage()`.

5.1.1 Implementazione di `SplashPage.xaml`

Naturalmente al file `SplashPage.xaml` è collegato il relativo file XAML (`SplashPage.xaml`), il quale ha come unico componente un elemento di tipo ffimageloading; tale file viene mostrato nel Listato 5.2.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="TrisBluetooth.SplashPage"
            xmlns:ffimageloading="clr-namespace:FFImageLoading.Svg.Forms;assembly=FFImageLoading.Svg.Forms"
            BackgroundImage="backgroundbumper"
            NavigationPage.HasNavigationBar="false">

    <ffimageloading:SvgCachedImage HorizontalOptions="Fill" VerticalOptions="Fill" x:Name=‘‘imageWav’’
    Source="logoBumper.gif" Margin="0" />
</ContentPage>
```

Listato 5.2. Il codice relativo a `SplashPage.xaml`

Per visualizzare l'animazione del logo dell'applicazione TrisBluetooth si è deciso di usare un'immagine di tipo GIF, la quale, per essere visualizzata, ha bisogno del pacchetto NuGet `Xamarin.FFImageLoading.Svg.Forms`.

L'imagine ha la caratteristica di avere gli allineamenti `HorizontalOptions= "Fill"` e `VerticalOptions="Fill"`. In altre parole, per l'allineamento orizzontale l'immagine riempie la larghezza del layout e per l'allineamento verticale essa riempie l'altezza del layout; infine con `Source="logoBumper.gif"` si indica a quale immagine si fa riferimento.

Per usare una singola immagine in tutte le applicazioni occorre che il file di immagine abbia lo stesso nome nelle diverse piattaforme; nel caso di Android l'immagine va caricata nella cartella `Resources/drawable`, mentre, per i dispositivi iOS, occorre caricare il file immagine nella cartella `Asset`.

5.2 Implementazione di `MainActivity.cs`

Il Listato 5.3, specifico per la piattaforma Android, definisce tramite `Label = "TrisBlu etooth"` e `Icon = "@mipmap/icona"`, rispettivamente, il nome e l'icona dell'applicazione.

```
using System;

using Android.App;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using Android.Bluetooth;
using Android.Support.V4.App;
using Android;
using Android.Support.V4.Content;
using Android.Content;
using Xamarin.Forms;
using System.Collections;
using System.Threading;
using Java.Util;
using System.IO;
using Java.IO;
using System.Text;
using Java.Nio.Charset;
using FFImageLoading.Forms.Platform;

namespace TrisBluetooth.Droid
{
    [Activity(Label = "TrisBluetooth", Icon = "@mipmap/icona", Theme = "@style/MainTheme", MainLauncher = true,
    ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
    public class MainActivity : global:: Xamarin.Forms.Platform.Android.FormsAppCompatActivity
    {
        // thread per gestire lo scambio di messaggi
        private static ConnectedThread mConnectedThread;

        // ruolo gestito dal dispositivo nella connessione bluetooth (true client, false server)
        private static bool ruolo = false;

        // Broadcast receiver per quando si scoprono nuovi dispositivi
        private BluetoothReceiver receiver = new BluetoothReceiver();
        private class BluetoothReceiver : BroadcastReceiver
```

```
{

    public override void OnReceive(Context context, Intent intent)
    {
        string action = intent.Action;

        // Quando il rilevamento trova un dispositivo, invia un messaggio contenente la descrizione del
        //dispositivo
        if (action == BluetoothDevice.ActionFound)
        {
            System.Console.WriteLine(action);
            // Ottiene l'oggetto BluetoothDevice
            BluetoothDevice device = (BluetoothDevice)intent.GetParcelableExtra(BluetoothDevice.ExtraDevice);
            System.Console.WriteLine("Trovato: " + device.Name + " " + device.Address);
            devices.Add(device);
            String[] Description = { device.Name, device.Address };
            MessagingCenter.Send<Object, String[]>(this, "SaveDevices", Description);
        } else if(action == BluetoothDevice.ActionAclConnected)
        {
        } else if(action == BluetoothDevice.ActionAclDisconnected)
        {
            System.Console.WriteLine(action);
            reset();
        }

    }
}
```

Listato 5.3. Definizione del metodo `OnReceive()` del file `MainActivity.cs`

Il metodo `OnReceive` fa in modo che durante la ricerca dei dispositivi limitrofi, quando il rilevamento trova un dispositivo, invia un messaggio contenente la descrizione dello stesso attraverso `MessagingCenter.Send<Object, String[]>(this, "SaveDevices", Description)` alla parte cross platform.

Anche nei successivi listati viene fatto uso del `MessagingCenter`. Quest'ultimo è un modello di messaggistica in cui i server di pubblicazione inviano messaggi senza conoscere i ricevitori, noti come sottoscrittori. In modo analogo, i sottoscrittori sono in ascolto di messaggi specifici, senza conoscere i server di pubblicazione.

La classe `MessagingCenter` di Xamarin.Forms implementa il modello di pubblica zione-sottoscrizione, consentendo la comunicazione basata su messaggi tra componenti che non è semplice collegare per mezzo di riferimenti a oggetti e tipi. Questo meccanismo consente ai server di pubblicazione e ai sottoscrittori di comunicare senza avere un riferimento reciproco, contribuendo a ridurre le dipendenze tra di essi. La classe `MessagingCenter` fornisce una funzionalità di pubblicazione-sottoscrizione multicast. Ciò significa che possono essere presenti più server di pubblicazione, che pubblicano un singolo messaggio, e più sottoscrittori in ascolto dello stesso messaggio, come mostrato in Figura 5.3.

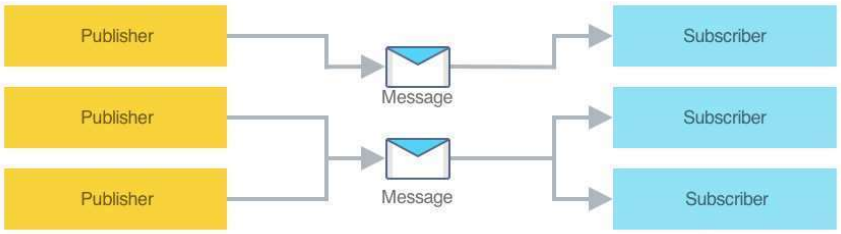

Figura 5.3. Funzionamento del `MessagingCenter`

I server di pubblicazione inviano i messaggi usando il metodo `MessagingCenter`

.Send, mentre i sottoscrittori sono in ascolto dei messaggi usando il metodo MessagingCenter .Subscribe. I sottoscrittori possono, inoltre, annullare la sottoscrizione dei messaggi, se necessario, con il metodo MessagingCenter.Unsubscribe.

Nell'applicazione oggetto della presente tesi il MessagingCenter è stato usato per inviare messaggi tra la parte nativa (Android) e quella cross-platform.

Nel Listato 5.4 va posta particolare attenzione a Window.SetStatusBarColor(Android.Graphics.Color.Rgb(255, 100, 60)); questa istruzione fa in modo che la barra di stato in alto abbia lo stesso colore (arancione) dello sfondo delle pagine dell'applicazione.

```
        //Array list che contiene i Device trovati
        public static System.Collections.ArrayList devices = new System.Collections.ArrayList();

        // Thread per gestire la connessione
        private static AcceptThread server;
        private static ConnectThread client;

        // UUID con cui fare la connessione
        private static readonly UUID MY_UUID =
    UUID.FromString("8ce255c0-200a-11e0-ac64-0800200c9a66");

        protected override void OnCreate(Bundle savedInstanceState)
        {
            CachedImageRenderer.Init(true);
            base.OnCreate(savedInstanceState);
            global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
            LoadApplication(new App());

            Window.SetStatusBarColor(Android.Graphics.Color.Rgb(255, 100, 60));

            //Questa funzione controlla se il dispositivo ha i permessi necessari
            checkPermission();

            //Questa funzione registra i Broadcast Receiver necessari
            registerReceivers();

            //Questa funzione registra le caselle per lo scambio di messaggi fra la parte android e cross-platform
            setMailBoxes();

        }
```

Listato 5.4. Definizione del metodo OnCreate() del file MainActivity.cs

Il Listato 5.5 mostra il metodo OnDestroy() che fa in modo che prima che la pagina venga "distrutta", venga stampato il messaggio "OnDestroy called" e venga attivato il metodo reset() mostrato nel Listato 5.6.

```
        protected override void OnDestroy()
        {
            base.OnDestroy();
            System.Console.WriteLine("OnDestroy called");
            reset();
        }
```

Listato 5.5. Definizione del metodo OnDestroy() del file MainActivity.cs

Il metodo reset() stampa il messaggio "Reset Chiamata" per poi far si che, se il thread per gestire lo scambio di messaggi non è vuoto, venga svuotato; lo stesso accade per il client e per il server.

```
        private static void reset()
        {
            System.Console.WriteLine("Reset Chiamata");

            if (mConnectedThread != null)
            {
                mConnectedThread.Cancel();
                mConnectedThread = null;
            }
            if (client != null)
            {
```

```
            client.Cancel();
            client = null;
        }
        if (server != null)
        {
            server.Cancel();
            server = null;
        }
    }
```

Listato 5.6. Definizione del metodo `reset()` del file `MainActivity.cs`

Il metodo `checkPermission()`, riportato nel Listato 5.7, controlla se l'utente ha concesso i permessi relativi alla posizione; qualora tali permessi non fossero stati concessi provvederà a richiederli.

```
    private void checkPermission()
    {
        const int locationPermissionsRequestCode = 1000;

        var locationPermissions = new[]
        {
            Manifest.Permission.AccessCoarseLocation,
            Manifest.Permission.AccessFineLocation
        };

        // Controlla se l'app e' autorizzata ad accedere a una posizione approssimativa
        var coarseLocationPermissionGranted =
            ContextCompat.CheckSelfPermission(this, Manifest.Permission.AccessCoarseLocation);

        // Controlla se l'app e' autorizzata ad accedere alla posizione corretta
        var fineLocationPermissionGranted =
            ContextCompat.CheckSelfPermission(this, Manifest.Permission.AccessFineLocation);

        // Se non si ha una delle due autorizzazioni dell'utente vengono richieste
        if (coarseLocationPermissionGranted == Permission.Denied ||
            fineLocationPermissionGranted == Permission.Denied)
        {
            ActivityCompat.RequestPermissions(this, locationPermissions, locationPermissionsRequestCode);
        }
    }
```

Listato 5.7. Definizione del metodo `checkPermission()` del file `MainActivity.cs`

Il metodo `registerReceivers()`, mostrato nel Listato 5.8, fa in modo che, quando un dispositivo Bluetooth viene trovato, questo venga registrato.

```
    private void registerReceivers()
    {
        // Quando viene trovato un dispositivo viene registrato
        receiver = new BluetoothReceiver();
        var filter = new IntentFilter(BluetoothDevice.ActionFound);
        RegisterReceiver(receiver, filter);

        // Registra i broadcasts quando la discovery termina
        filter = new IntentFilter(BluetoothAdapter.ActionDiscoveryFinished);
        RegisterReceiver(receiver, filter);
    }
```

Listato 5.8. Definizione del metodo `registerReceivers()` del file `MainActivity.cs`

Il metodo `setMailBoxes()` mostrato nel Listato 5.9, usa il `MessagingCenter` descritta precedentemente.

Per quanto riguarda "Request", possono accadere diversi casi, sulla base dell'argomento. In particolare, gli argomenti possibili sono i seguenti:

- *Server*: imposta il ruolo (un booleano che è `true` se è client e `false` se è server) a `false`, stampa il messaggio "Richiesta di Server" e avvia `AcceptThread()`.
- *Client*: imposta il ruolo a `true`, stampa il messaggio "Richiesta di Client" e avvia `ConnectThread()`.

- *Discovery*: stampa il messaggio "Richiesta di Discoverability" e avvia `Abilita Discoverabilty()`.
- *Reset*: stampa il messaggio "Richiesta di Reset" e avvia `reset()`.

Per quanto riguarda "IncomingMessage", l'argomento può essere suddiviso in due parti; la prima è la posizione cliccata (`arg[0]`) mentre la seconda rappresenta l'esito (`arg[1]`).

```
private void setMailBoxes()
{

    MessagingCenter.Subscribe<Object, String>(this, "Request",
        (sender, arg) =>
        {
            String request = arg;

            if (request.Equals("Server"))
            {
                ruolo = false;
                System.Console.WriteLine("Richiesta di Server");
                server = new AcceptThread();
                server.Start();
            }
            else if (request.Equals("Client"))
            {
                ruolo = true;
                System.Console.WriteLine("Richiesta di Client");
                client = new ConnectThread(Bth.serverDevice, MY_UUID);
                client.Start();
            } else if(request.Equals("Discovery"))
            {
                System.Console.WriteLine("Richiesta di Discoverability");
                AbilitaDiscoverabilty();
            } else if (request.Equals("Reset"))
            {
                System.Console.WriteLine("Richiesta di Reset");
                reset();
            }
        });

    MessagingCenter.Subscribe<Object, string>(this, "IncomingMessage",
        (sender, arg) =>
        {
            System.Console.WriteLine("posizione cliccata: " + arg[0]);
            System.Console.WriteLine("esito: " + arg[1]);
            byte[] bytes = Encoding.Unicode.GetBytes(arg);
            mConnectedThread.Write(bytes);
        });

}
```

Listato 5.9. Definizione del metodo `setMailBoxes()` del file `MainActivity.cs`

Il Listato 5.10 mostra il metodo `AbilitaDiscoverabilty()` che fa in modo di abilitare la ricerca di dispositivi Bluetooth limitrofi, se tale funzionalità non è già abilitata. La durata della ricerca è di 30 secondi e stampa il messaggio "Discoverability Attivata".

```
//abilita la discoverability, se bt e' spento lo accende
private void AbilitaDiscoverabilty()
{
    Intent discoverableIntent =
           new Intent(BluetoothAdapter.ActionRequestDiscoverable);
    discoverableIntent.PutExtra(BluetoothAdapter.ExtraDiscoverableDuration, 30);
    StartActivity(discoverableIntent);
    System.Console.WriteLine("Discoverability Attivata");
}
```

Listato 5.10. Definizione del metodo `AbilitaDiscoverabilty()` del file `MainActivity.cs`

Il Listato 5.11 mostra la classe `ConnectThread` per gestire il client. Tale classe, attraverso il metodo `Run()` fa in modo che si disattivi la ricerca di altri dispositivi per non rallentare la connessione con il dispositivo remoto attraverso il socket; ne-

cessariamente, prova a connettersi come client e, se non ce la fa, genera un'eccezione. Infine, il metodo `Cancel()` chiude il socket del client e fa terminare il thread.

```
//Classe per Gestire il Client
private class ConnectThread : Java.Lang.Thread
{
    private readonly BluetoothSocket mmSocket;
    private readonly BluetoothDevice mmDevice;


    public ConnectThread(BluetoothDevice device, UUID myUuid)
    {
        // Utilizza un oggetto temporaneo che verra' successivamente assegnato a mmSocket
        // perche' mmSocket e' final.
        BluetoothSocket tmp = null;
        mmDevice = device;

        try
        {
            // Ottiene un BluetoothSocket per la connessione con il dispositivo Bluetooth indicato.
            // MY_UUID e' la stringa UUID dell'app, utilizzata anche nel codice del server.
            tmp = device.CreateRfcommSocketToServiceRecord(myUuid);
        }
        catch (Java.IO.IOException e)
        {
            System.Console.WriteLine("Errore: " + e);
        }
        mmSocket = tmp;
    }

    override
    public void Run()
    {
        // Annulla il rilevamento perche' altrimenti rallenta la connessione.
        Bth.mBluetoothAdapter.CancelDiscovery();

        try
        {
            // Connessione con il dispositivo remoto attraverso il socket.
            // Questa chiamata si blocca fino a quando non riesce o genera un'eccezione.
            System.Console.WriteLine("Tentativo di Conessione come Client");
            mmSocket.Connect();
        }
        catch (Java.IO.IOException connectException)
        {
            System.Console.WriteLine("Errore: " + connectException);
            //Impossibile connetersi; chiudere il socket e tornare.
            try
            {
                mmSocket.Close();
            }
            catch (Java.IO.IOException closeException)
            {
                System.Console.WriteLine("Errore: " + closeException);
            }
            return;
        }

        System.Console.WriteLine("Connesso come Client");



        // Il tentativo di connessione e' riuscito. Eseguire il lavoro associato alla connessione in
        //un thread separato.
        mConnectedThread = new ConnectedThread(mmSocket);
        mConnectedThread.Start();
    }

    // Chiude il socket del client e fa terminare il thread.
    public void Cancel()
    {
        try
        {
            mmSocket.Close();
        }
        catch (Java.IO.IOException e)
        {
            System.Console.WriteLine("Errore: " + e);
        }
    }
}
```

Listato 5.11. Definizione della classe `ConnectThread` del file `MainActivity.cs`

Il Listato 5.12 riporta la classe `AcceptThread` che ha il compito di gestire il server. Questa classe, attraverso il metodo `Run()`, fa in modo che disattivi la ricerca

di altri dispositivi per non rallentare la connessione con il dispositivo remoto attraverso il socket; necessariamente prova a connettersi come server e, se non ce la fa, genera un'eccezione. Infine, il metodo `Cancel()` chiude il socket del server e fa terminare il thread.

```
//classe per gestire il Server
private class AcceptThread : Java.Lang.Thread
{
    private readonly BluetoothServerSocket mmServerSocket;


    public AcceptThread()
    {

        BluetoothServerSocket tmp = null;
        try
        {
            tmp = Bth.mBluetoothAdapter.ListenUsingRfcommWithServiceRecord("NAME", MY_UUID);

        }
        catch (Java.IO.IOException e)
        {
            System.Console.WriteLine("Errore: " + e);
        }
        mmServerSocket = tmp;
    }

    override
    public void Run()
    {
        BluetoothSocket socket = null;
        // Continua ad ascoltare fino a quando si verifica un'eccezione o viene restituito un socket.
        while (true)
        {
            try
            {
                System.Console.WriteLine("Tentativo di Conessione come Server");

                socket = mmServerSocket.Accept();

            }
            catch (Java.IO.IOException e)
            {
                System.Console.WriteLine("Errore: " + e);
                break;
            }
            System.Console.WriteLine("Connesso come Server");

            if (socket != null)
            {
                mConnectedThread = new ConnectedThread(socket);
                mConnectedThread.Start();
                break;
            }
        }
    }

    // Chiude il socket di connessione e fa terminare il thread.
    public void Cancel()
    {
        try
        {
            mmServerSocket.Close();
        }
        catch (Java.IO.IOException e)
        {
            System.Console.WriteLine("Errore: " + e);
        }
    }

}
```

Listato 5.12. Definizione della classe `AcceptThread` del file `MainActivity.cs`

La classe `ConnectedThread`, riportata nel Listato 5.13, gestisce i flussi in input e output; infatti, attraverso il metodo `Run()`, legge il flusso in ingresso (`InputStream`).

Il metodo `Write()`, invece, chiama la `Main Activity` per inviare dati al dispositivo remoto.

Infine, il metodo `Cancel()` chiama questo metodo dalla Main Activity per chiudere la connessione.

```
//questa classe gestisce lo scambio di messaggi
private class ConnectedThread: Java.Lang.Thread
{
    private Handler handler;

    private readonly BluetoothSocket mmSocket;
    private readonly InputStream mmInStream;
    private readonly OutputStream mmOutStream;
    private byte[] mmBuffer;

    public ConnectedThread(BluetoothSocket socket)
    {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;


        // Ottiene i flussi di input e output; usando oggetti temp perche' i flussi membri sono final.
        try
        {
            tmpIn = ((Android.Runtime.InputStreamInvoker)socket.InputStream).BaseInputStream;
        }
        catch (System.IO.IOException e)
        {
            System.Console.WriteLine("Errore: " + e);
        }
        try
        {
            tmpOut = ((Android.Runtime.OutputStreamInvoker)socket.OutputStream).BaseOutputStream;
        }
        catch (Java.IO.IOException e)
        {
            System.Console.WriteLine("Errore: " + e);
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    override
    public void Run()
    {
        mmBuffer = new byte[1024];
        int numBytes; // i bytes che ritornano dopo il read()

        // Continua ad ascoltare InputStream fino a quando si verifica un'eccezione.
        while (true)
        {
            try
            {
                // Legge dall'InputStream.
                numBytes = mmInStream.Read(mmBuffer);
                //invia i bytes ottenuti all'UI activity.
                System.Console.WriteLine(numBytes);
                String message = Encoding.Unicode.GetString(mmBuffer, 0, numBytes);
                System.Console.WriteLine("Messaggio: " + message);
                try {
                    MessagingCenter.Send<Object, String>(this, "OutgoingMessage", message);

                } catch (Exception e)
                {
                    System.Console.WriteLine("Errore: " + e);
                }

            }
            catch (Java.IO.IOException e)
            {
                System.Console.WriteLine("Errore: " + e);
                break;
            }
        }

    }

    // Chiama la Main Activity per inviare dati al dispositivo remoto.
    public void Write(byte[] bytes)
    {
        string text = System.Text.Encoding.Default.GetString(bytes);
        try
        {
            mmOutStream.Write(bytes);
        }
        catch (Java.IO.IOException e)
        {
            System.Console.WriteLine("Errore: " + e);
        }

    }
        // Chiama questo metodo dalla Main Activity per chiudere la connessione.
    public void Cancel()
    {
        try
        {
```

```
                mmSocket.Close();
            }
            catch (Java.IO.IOException e)
            {
            System.Console.WriteLine("Errore: " + e);
            }
        }
    }

}
}
```

Listato 5.13. Definizione della classe `ConnectedThread` del file `MainActivity.cs`

5.3 Implementazione di `MainPage.xaml.cs`

Il Listato 5.14 riporta il costruttore del file `MainPage.xaml.cs`. Tale costruttore, oltre a caricare il file XAML mediante `InitializeComponent()`, riceve, per mezzo della `MessagingCenter`, in "SaveDevices", il MAC e il nome dei dispositivi Bluetooth limitrofi rilevati, e in "OutgoingMessage" il messaggio ricevuto dall'altro dispositivo che rinvierà alla pagina `Gioca.xaml.cs`.

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms.Xaml;
using Xamarin.Forms;

namespace TrisBluetooth
{
    public partial class MainPage : ContentPage
    {

        public ObservableCollection<ListViewTemplate> data = new ObservableCollection<ListViewTemplate>();
        Button play;
        Image playImage;
        public Gioca page = new Gioca();

        public MainPage()
        {
            InitializeComponent();
            MainListView.ItemsSource = data;

            MessagingCenter.Subscribe<Object, String[]>(this, "SaveDevices",
            (sender, arg) =>
            {
                string[] description = arg;
                string name = description[0];
                string mac = description[1];

                data.Add(new ListViewTemplate(name, mac));
                MainListView.ItemsSource = data;
            });

            MessagingCenter.Subscribe<Object, String>(this, "OutgoingMessage",
            (sender, arg) =>
            {
                page.messageReceived(arg);
            });

            play = FindByName("gioca") as Button;
            playImage = FindByName("giocaImage") as Image;
        }
```

Listato 5.14. Definizione del costruttore del file `MainPage.xaml.cs`

Il metodo `Cerca()`, Listato 5.15, invia, attraverso il `MessagingCenter`, "Discovery" alla `MainActivity.cs` e avvia la ricerca dei dispositivi Bluetooth limitrofi visibili.

```
async void Cerca(object sender, EventArgs args)
{
    MessagingCenter.Send<Object, String>(this, "Request", "Discovery");
    DependencyService.Get<IBluetooth>().StartDiscovery();
}
```

Listato 5.15. Definizione del metodo `Cerca()` del file `MainPage.xaml.cs`

Il metodo `MainListView_ItemTapped()`, Listato 5.16), fa in modo che, una volta selezionato il dispositivo con il quale ci si vuole connettere, si possa effettuare il bond, attraverso il MAC, e si renda visibile e cliccabile il pulsante play.

```
async private void MainListView_ItemTapped(object sender, ItemTappedEventArgs e)
{
    var Selected = e.Item as ListViewTemplate;
    DependencyService.Get<IBluetooth>().CreateBond(Selected.Mac);
    play.IsVisible = true;
    playImage.IsVisible = true;
}
```

Listato 5.16. Definizione del metodo `MainListView_ItemTapped()` del file `MainPage.xaml.cs`

Il Listato 5.17 definisce il metodo associato al pulsante "Play", attraverso il quale si passa alla pagina `Gioca.xaml.cs`.

```
private void Gioca(object sender, EventArgs e)
{
    Navigation.PushAsync(page);
}
```

Listato 5.17. Definizione del metodo `Gioca()` del file `MainPage.xaml.cs`

`Info()`, Listato 5.18, è il metodo grazie al quale è possibile visualizzare l'alert con le istruzioni della pagina `MainPage.xaml.cs`.

```
public void Info(object sender, EventArgs e)
{
    Application.Current.MainPage.DisplayAlert("Istruzioni", "1) Cliccare il pulsante Cerca e attendere
    la ricerca dei dispositivi bluetooth \n2) Selezionare il device del tuo avversario \n3) Cliccare
    Gioca", "OK");
}
    }
}
```

Listato 5.18. Definizione del metodo `Info()` del file `MainPage.xaml.cs`

5.3.1 Implementazione di `MainPage.xaml`

Il Listato 5.19 definisce il file `MainPage.xaml` collegato alla pagina `MainPage`.
La pagina è stata impostata come una tabella (`<Grid>`) dove:

- *La cella in alto a destra* è occupata da un pulsante che, una volta cliccato, attiva il metodo `Info()`, nonché da un'immagine identificata con "info" sovrapposta al pulsante.
- *La cella centrale al centro* è occupata da un pulsante che, cliccato, attiva il metodo `Cerca()`, nonché da un'immagine identificata con "button" sovrapposta al pulsante.

- Al centro della pagina è inserita la lista dei dispositivi rilevati dalla ricerca.
- *La cella in fondo al centro* è occupata da un pulsante che, cliccato, attiva il metodo `Gioca()`, nonché da un'immagine identificata con "button", sovrapposta al pulsante, che diventa visibile e cliccabile solo quando l'utente seleziona il dispositivo che vuole sfidare.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:TrisBluetooth"
             x:Class="TrisBluetooth.MainPage"
             NavigationPage.HasNavigationBar="false"
             BackgroundImage="background3">

    <StackLayout Margin="10">
        <Grid>
            <Image Source="info"
              Aspect="AspectFit"
              VerticalOptions="CenterAndExpand"
              HorizontalOptions="End"
              WidthRequest="40"
               HeightRequest="40"/>

            <Button HorizontalOptions="End"
                VerticalOptions="CenterAndExpand"
                BackgroundColor="Transparent"
                WidthRequest="40"
                HeightRequest="40"
                Clicked="Info"/>
        </Grid>

        <Grid>


            <Image Source="button"
              Aspect="AspectFit"
              VerticalOptions="CenterAndExpand"
              HorizontalOptions="Center"
              WidthRequest="110"
               HeightRequest="50"/>

            <Button VerticalOptions="CenterAndExpand"
                TextColor="Red"
                Text="Cerca"
                FontAttributes="Bold"
                HorizontalOptions="Center"
                BackgroundColor="Transparent"
                FontFamily="IndieFlower.ttf#IndieFlower"
                WidthRequest="110"
                HeightRequest="50"
                Clicked="Cerca"/>
        </Grid>

              <ListView x:Name="MainListView"
             HasUnevenRows="True"
             ItemTapped="MainListView_ItemTapped">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout Orientation="Vertical">
                            <Grid>
                                <Label Text="{Binding Name}"
                                       TextColor="White"
                                       VerticalOptions="CenterAndExpand"
                                       HorizontalOptions="Start"
                                       WidthRequest="110"
                                       FontFamily="IndieFlower.ttf#IndieFlower"
                                       HeightRequest="50"/>
                                <Label Text="{Binding Mac}"
                                      TextColor="White"
                                      VerticalOptions="CenterAndExpand"
                                      FontFamily="IndieFlower.ttf#IndieFlower"
                                      HorizontalOptions="End"/>
                            </Grid>
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>

        <Grid>
            <Image x:Name="giocaImage"
              Source="button"
              Aspect="AspectFit"
              VerticalOptions="CenterAndExpand"
              HorizontalOptions="Center"
              WidthRequest="110"
```

```
                HeightRequest="50"
                IsVisible="False"/>

            <Button x:Name="gioca"
                Text="Gioca"
                FontFamily="IndieFlower.ttf#IndieFlower"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center"
                TextColor="Red"
                BackgroundColor="Transparent"
                WidthRequest="110"
                HeightRequest="50"
                Clicked="Gioca"
                IsVisible="False"/>
        </Grid>
    </StackLayout>

</ContentPage>
```

Listato 5.19. Il codice relativo al file `MainPage.xaml`

5.4 Implementazione di `App.xaml.cs`

Nel Listato 5.20 viene presentato il file `App.xaml.cs`; dall'esame del listato è possibile vedere come è stato implementato il costruttore `App()`. Dopo aver caricato il file `App.xaml` per mezzo di `InitializeComponent()`, si passa alla Splash Page.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[assembly: XamlCompilation(XamlCompilationOptions.Compile)]
namespace TrisBluetooth
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            var navigationPage = new NavigationPage(new SplashPage());
            MainPage = navigationPage;

        }

        protected override void OnStart()
        {
            // operazioni all'avvio dell'app
        }

        protected override void OnSleep()
        {
            // operazioni quando l'app "dorme"
        }

        protected override void OnResume()
        {
            var navigationPage = new NavigationPage(new MainPage());
            MainPage = navigationPage;
        }
    }
}
```

Listato 5.20. Il codice relativo al file `App.xaml.cs`

5.4.1 Implementazione di `App.xaml`

Il file `App.xaml` (Listato 5.21), come già riportato nel Capitolo 2, definisce un dizionario di risorse; infatti, attraverso `Setter`, viene definito come FontFamily un Font importato "IndieFlower", disponibile gratuitamente da GoogleFonts.

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TrisBluetooth.App">
    <Application.Resources>
        <Style TargetType="Label">
            <Setter Property="FontFamily" Value="IndieFlower.ttf#IndieFlower"/>
        </Style>
    </Application.Resources>
</Application>
```

Listato 5.21. Il codice relativo al file App.xaml

5.5 Implementazione di Gioca.xaml.cs

Il Listato 5.22 mostra il costruttore `Gioca()` che, come sempre, carica il relativo file XAML (`Gioca.xaml`) attraverso `InitializeComponent ()` e collega i relativi pulsanti e le corrispettive immagini al file XAML per mezzo di `FindByName()`.

```
using Android.OS;
using FFImageLoading.Svg.Forms;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace TrisBluetooth
{
	public partial class Gioca : ContentPage
	{

	public static bool[] cliccati = new bool[9];
	public static bool[] occupati = new bool[9];

	public static Button[] cellButton = new Button[9];
	public static Image[] cellImageX = new Image[9];
	public static Image[] cellImageO = new Image[9];

	int mioPunteggio = 0;
	int suoPunteggio = 0;
	//master sara' true solo se il giocatore e' client
	public static bool master = false;

	public static Button riv;
	public static Button x;
	public static Button o;
	public static Image xImage;
	public static Image oImage;
	public static Image rivincitaImage;
	public static Label punteggio;
	public static Label display;

	public static SvgCachedImage vinto;
	public static SvgCachedImage pareggio;
	public static SvgCachedImage perso;

	public Gioca()
		{
			InitializeComponent ();

	    riv = FindByName("Revenge") as Button;
	    x = FindByName("X") as Button;
	    o = FindByName("O") as Button;
	    xImage = FindByName("XImage") as Image;
	    oImage = FindByName("OImage") as Image;
	    rivincitaImage = FindByName("RivincitaImage") as Image;
	    punteggio = FindByName("Punteggio") as Label;
	    display = FindByName("Display") as Label;


	    punteggio.Text = mioPunteggio.ToString() + " : " + suoPunteggio.ToString();
	    display.Text = "Scegli il tuo simbolo";

	    //assegna agli array le immagini e i bottoni della griglia
```

```
        cellButton[0] = FindByName("cella0") as Button;
        cellImageX[0] = FindByName("XImg0") as Image;
        cellImage0[0] = FindByName("OImg0") as Image;
        cellButton[0].Clicked += (sender, args) => { ButtonClicked(0); };


        cellButton[1] = FindByName("cella1") as Button;
        cellImageX[1] = FindByName("XImg1") as Image;
        cellImage0[1] = FindByName("OImg1") as Image;
        cellButton[1].Clicked += (sender, args) => { ButtonClicked(1); };


        cellButton[2] = FindByName("cella2") as Button;
        cellImageX[2] = FindByName("XImg2") as Image;
        cellImage0[2] = FindByName("OImg2") as Image;
        cellButton[2].Clicked += (sender, args) => { ButtonClicked(2); };


        cellButton[3] = FindByName("cella3") as Button;
        cellImageX[3] = FindByName("XImg3") as Image;
        cellImage0[3] = FindByName("OImg3") as Image;
        cellButton[3].Clicked += (sender, args) => { ButtonClicked(3); };


        cellButton[4] = FindByName("cella4") as Button;
        cellImageX[4] = FindByName("XImg4") as Image;
        cellImage0[4] = FindByName("OImg4") as Image;
        cellButton[4].Clicked += (sender, args) => { ButtonClicked(4); };


        cellButton[5] = FindByName("cella5") as Button;
        cellImageX[5] = FindByName("XImg5") as Image;
        cellImage0[5] = FindByName("OImg5") as Image;
        cellButton[5].Clicked += (sender, args) => { ButtonClicked(5); };


        cellButton[6] = FindByName("cella6") as Button;
        cellImageX[6] = FindByName("XImg6") as Image;
        cellImage0[6] = FindByName("OImg6") as Image;
        cellButton[6].Clicked += (sender, args) => { ButtonClicked(6); };


        cellButton[7] = FindByName("cella7") as Button;
        cellImageX[7] = FindByName("XImg7") as Image;
        cellImage0[7] = FindByName("OImg7") as Image;
        cellButton[7].Clicked += (sender, args) => { ButtonClicked(7); };


        cellButton[8] = FindByName("cella8") as Button;
        cellImageX[8] = FindByName("XImg8") as Image;
        cellImage0[8] = FindByName("OImg8") as Image;
        cellButton[8].Clicked += (sender, args) => { ButtonClicked(8); };

        vinto = FindByName("HaiVinto") as SvgCachedImage;
        perso = FindByName("HaiPerso") as SvgCachedImage;
        pareggio = FindByName("HaiPareggiato") as SvgCachedImage;

        vinto.IsVisible = false;
        perso.IsVisible = false;
        pareggio.IsVisible = false;

        riv.IsVisible = false;

        reset();
    }
```

Listato 5.22. Il codice relativo al costruttore del file `Gioca.xaml.cs`

Il metodo `Info()`, mostrato nel Listato 5.23, fa in modo che, una volta invocato, appaia l'alert con le istruzioni della pagina `Gioca`.

```
    // alert per istruzioni gioco
    public void Info(object sender, EventArgs e)
    {
        Application.Current.MainPage.DisplayAlert("Istruzioni", "1)Decidere chi tra te e i tuo avversario
        inizia a giocare, \n2)Colui che inizia sara' la X e l'altro giocatore il O, \n3)Vince il giocatore
        che riesce a disporre tre dei propri simboli in linea retta orizzontale, verticale o diagonale
        \n4)Per continuare il gioco cliccate entrambi Rivincita, in ogni caso il punteggio continuera' ad
        essere presente nella parte altra dello schermo", "OK");
    }
```

Listato 5.23. Il codice relativo al metodo `Info()` del file `Gioca.xaml.cs`

Il Listato 5.24 riporta il metodo `Client()`, il quale invia, attraverso il `Messaging Center`, l'opzione "Client" alla `MainActivity.cs`, rendendo, così, cliccabili e visibili i pulsanti e le relative immagini di X e O, successivamente, esso fa mostrare nella Label il testo "Tocca a te" e rende cliccabili tutte le celle della griglia di gioco.

```
public void Client(object sender, EventArgs args)
{
    MessagingCenter.Send<Object, String>(this, "Request", "Client");
    xoisEnable(false);
    display.Text = "Tocca a te";
    ButtonEnable(true);
    master = true;
}
```

Listato 5.24. Il codice relativo al metodo `Client()` del file `Gioca.xaml.cs`

Il Listato 5.25, riporta il metodo `Server()`; questo invia, attraverso il `Messaging Center`, l'opzione "Server" alla `MainActivity.cs`, rendendo, così, non cliccabili e non visibili i pulsanti e le relative immagini di X e O, fa mostrare nella Label il testo "Attendi..." e rende non cliccabili le celle della griglia di gioco.

```
public void Server(object sender, EventArgs args)
{
    MessagingCenter.Send<Object, String>(this, "Request", "Server");
    xoisEnable(false);
    display.Text = "Attendi...";
    master = false;
}
```

Listato 5.25. Il codice relativo al metodo `Server()` del file `Gioca.xaml.cs`

Il metodo `Rivincita()` (Listato 5.26) invia, attraverso il `MessagingCenter`, l'opzione "Reset" alla `MainActivity.cs` e rende visibili e cliccabili i pulsanti e le relative immagini di X e O.

```
public void Rivincita(object sender, EventArgs args)
{
    MessagingCenter.Send<Object, String>(this, "Request", "Reset");
    reset();
    xoisEnable(true);
}
```

Listato 5.26. Il codice relativo al metodo `Rivincita()` del file `Gioca.xaml.cs`

Il Listato 5.27 riporta il metodo `ButtonClicked()` che viene invocato ogni volta che viene cliccata una cella della griglia di gioco. Il metodo rende tutte le celle della griglia di gioco non cliccabili, fa mostrare sul display della pagina "Attendi..." e imposta a "true" la posizione relativa al pulsante cliccato degli array `cliccati` e `occupati`. L'array `cliccati` definisce le celle che sono state cliccate dal giocatore con `true` e l'array `occupati` identifica con `true` le celle della griglia di gioco cliccate dal giocatore e dall'avversario. Infine, a seconda se il giocatore è client o server, viene reso visibile nella cella cliccata l'immagine X o O.

Se, una volta invocato il metodo `victory()`, questo restituisce `true`, l'esito sarà impostato ad "1", nel display apparirà la dicitura "Hai Vinto!", il "mioPunteggio" sarà incrementato di uno e viene resa visibile l'animazione relativa alla vittoria per 7 secondi.

Se, invece, una volta invocato il metodo `draw()`, questo restituisce `true`, l'esito sarà impostato a "2", nel display apparirà la dicitura "Pareggio!"e verrà resa visibile l'animazione relativa al pareggio per 7 secondi.

Se nessuno dei due casi precedenti si verifica, verranno inviati all'avversario la posizione cliccata e l'esito della partita ("0" se perso,"1" se vinto e "2" se pareggiato), attraverso il `MessagingCenter`.

```
public void ButtonClicked(int position)
{
    ButtonEnable(false);
    display.Text = "Attendi...";
    cliccati[position] = true;
    occupati[position] = true;
    if (master) cellImageX[position].IsVisible = true;
    else cellImageO[position].IsVisible = true;

    int esito = 0;

    if (victory())
    {
        esito = 1;
        display.Text = "Hai Vinto!";

        mioPunteggio++;
        punteggio.Text = mioPunteggio.ToString() + " : " + suoPunteggio.ToString();

        vinto.IsVisible = true;

        Handler h = new Handler();
        Action start = () =>
        {
            vinto.IsVisible = false;
        };
        h.PostDelayed(start, 7000);

        riv.IsEnabled = true;
        riv.IsVisible = true;
        rivincitaImage.IsVisible = true;
    } else if (draw())
    {
        esito = 2;
        display.Text = "Pareggio!";
        pareggio.IsVisible = true;

        Handler h = new Handler();
        Action start = () =>
        {
            pareggio.IsVisible = false;
        };
        h.PostDelayed(start, 7000);

        riv.IsEnabled = true;
        riv.IsVisible = true;
        rivincitaImage.IsVisible = true;
    }

    MessagingCenter.Send<Object, string>(this, "IncomingMessage", position + esito.ToString());
}
```

Listato 5.27. Il codice del metodo `ButtonClicked()` del file `Gioca.xaml.cs`

Il Listato 5.28 mostra il metodo `reset()`, il quale, ogni volta che viene invocato, fa iniziare una nuova partita. Esso, quindi, rende tutte le celle della griglia di gioco non visibili e non cliccabili e imposta tutti gli elementi degli array `cliccati` e `occupati` a `false` e, naturalmente, rende non cliccabile e non visibile il pulsante rivincita e la relativa immagine sovrapposta, poiché, quando viene richiesta una nuova partita, significa che già tale pulsante è stato cliccato oppure i due giocatori stanno per attivare una nuova partita per la prima volta.

Il display mostra "Scegli il tuo simbolo"; infatti i pulsanti X e O e le relative immagini vengono resi visibili e cliccabili.

```
//invocata ogni volta che si inizia la partita
public void reset()
{
    master = false;
```

```
        for (int i = 0; i < 9; i++)
        {
            cellButton[i].IsEnabled = false;
            cellImageX[i].IsVisible = false;
            cellImageO[i].IsVisible = false;
            cliccati[i] = false;
            occupati[i] = false;
        }
        riv.IsEnabled = false;
        riv.IsVisible = false;
        rivincitaImage.IsVisible = false;
        xoisEnable(true);

        display.Text = "Scegli il tuo simbolo";
    }
```

Listato 5.28. Il codice relativo al metodo `reset()` del file `Gioca.xaml.cs`

Il metodo `ButtonEnable()`, Listato 5.29, riceve come argomento un booleano; se questo è pari a true allora tutte le celle della griglia di gioco vengono rese cliccabili, mentre se è false tutti i pulsanti della griglia di gioco sono resi non cliccabili.

```
    //cambia isEnable dei bottoni della griglia
    public void ButtonEnable(bool enable)
    {
        for (int i = 0; i < 9; i++)
        {
           cellButton[i].IsEnabled = enable;
        }
    }
```

Listato 5.29. Il codice relativo al metodo `ButtonEnable()` del file `Gioca.xaml.cs`

Il metodo `xoisEnable()`, Listato 5.30, riceve come argomento un booleano; se è `true` rende visibili e cliccabili i pulsanti X e O e le relative immagini, mentre se è `false` rende non visibili e non cliccabili i pulsanti X e O e le relative immagini.

```
    //cambia enable per i bottoni Server e Client
    public void xoisEnable (bool enable)
    {
        x.IsEnabled = enable;
        x.IsVisible = enable;
        o.IsEnabled = enable;
        o.IsVisible = enable;
        xImage.IsVisible = enable;
        oImage.IsVisible = enable;
    }
```

Listato 5.30. Il codice relativo al metodo `xoisEnable()` del file `Gioca.xaml.cs`

Il Listato 5.31 mostra il metodo `ButtonClick()`, il quale, a seconda del booleano che riceve come argomento, rende cliccabile o meno tutti i pulsanti della griglia di gioco che non sono occupati (ovvero non sono `true` nell'array `occupati`).

```
    //rende enable solo i bottoni che non sono occupati
    public void ButtonClick(bool enable)
    {
        for (int i = 0; i < 9; i++)
        {
            if(!occupati[i]) cellButton[i].IsEnabled = enable;
        }
    }
```

Listato 5.31. Il codice relativo al metodo `ButtonClick()` del file `Gioca.xaml.cs`

Il metodo che verifica se il giocatore ha vinto è `victory()`, Listato 5.32, il quale controlla tutte le possibili combinazioni che implicano la vittoria e, se almeno una di queste si verifica, restituisce `true` altrimenti `false`.

```
//funzione che verifica se il giocatore ha vinto
private Boolean victory()
{
    if (cliccati[0] && cliccati[1] && cliccati[2]) return true;
    if (cliccati[3] && cliccati[4] && cliccati[5]) return true;
    if (cliccati[6] && cliccati[7] && cliccati[8]) return true;
    if (cliccati[0] && cliccati[3] && cliccati[6]) return true;
    if (cliccati[1] && cliccati[4] && cliccati[7]) return true;
    if (cliccati[2] && cliccati[5] && cliccati[8]) return true;
    if (cliccati[0] && cliccati[4] && cliccati[8]) return true;
    if (cliccati[2] && cliccati[4] && cliccati[6]) return true;

    return false;
}
```

Listato 5.32. Il codice relativo al metodo `victory()` del file `Gioca.xaml.cs`

Il metodo `draw()`, mostrato nel Listato 5.33, controlla se il giocatore ha pareggiato e, come il metodo che controlla la vittoria, restituisce `true` o `false` a seconda se il giocatore ha pareggiato oppure no.

```
//funzione che verifica se c'e' pareggio
private Boolean draw()
{
    for (int i = 0; i < 9; i++)
    {
        if (!cliccati[i] & !occupati[i]) return false;
    }
    return true;
}
```

Listato 5.33. Il codice relativo al metodo `draw()` del file `Gioca.xaml.cs`

Il Listato 5.34 mostra il metodo `messageReceived()` il quale una volta ricevuto il messaggio (stringa) da parte dell'avversario, lo converte, attraverso `Int32.TryParse()`, in un intero e controlla che valore ha l'intero relativo all'esito. Più specificatamente, i valori possibili sono i seguenti:

- *0*: il display mostrerà "Tocca a te" e invocherà il metodo `ButtonClick(true)`.
- *1*: il display mostrerà "Hai Perso!" e renderà il pulsante "Rivincita" e la relativa immagine visibili e cliccabili, verrà incrementato il punteggio dell'avversario di uno e verrà resa visibile l'animazione della sconfitta per 7 secondi.
- *2*: il display mostrerà "Pareggio!" e renderà il pulsante "Rivincita" e la relativa immagine visibili e cliccabili, e verrà resa visibile l'animazione relativa al pareggio per 7 secondi.

```
public void messageReceived(String arg)
{
    string positionString = arg[0].ToString();
    string esitoString = arg[1].ToString();

    Int32.TryParse(positionString, out int position);
    Int32.TryParse(esitoString, out int esito);

    Device.BeginInvokeOnMainThread(() => {
        if (master) cellImageO[position].IsVisible = true;
        else cellImageX[position].IsVisible = true;
        occupati[position] = true;

        switch (esito)
        {
            case 0:
                {
                    display.Text = "Tocca a te";
                    ButtonClick(true);
                }
                break;
            case 1:
                {
                    display.Text = "Hai Perso!";
```

```
                riv.IsEnabled = true;
                riv.IsVisible = true;
                rivincitaImage.IsVisible = true;
                suoPunteggio++;
                punteggio.Text = mioPunteggio.ToString() + " : " + suoPunteggio.ToString();
                perso.IsVisible = true;

                Handler h = new Handler();
                Action start = () =>
                {
                    perso.IsVisible = false;
                };
                h.PostDelayed(start, 7000);
            }
            break;
        case 2:
            {
                display.Text = "Pareggio!";
                riv.IsEnabled = true;
                riv.IsVisible = true;
                rivincitaImage.IsVisible = true;

                pareggio.IsVisible = true;

                Handler h = new Handler();
                Action start = () =>
                {
                    pareggio.IsVisible = false;
                };
                h.PostDelayed(start, 7000);
            }
            break;
        }

    });

}
```

Listato 5.34. Il codice relativo al metodo `messageReceived()` del file `Gioca.xaml.cs`

Il metodo `OnDisappearing()`, Listato 5.35, permette di invocare il metodo `reset()` quando la pagina `Gioca` scompare.

```
    protected override void OnDisappearing()
    {
        reset();
    }
}
}
```

Listato 5.35. Il codice relativo al metodo `OnDisappearing()` del file `Gioca.xaml.cs`

5.5.1 Implementazione di `Gioca.xaml`

Il Listato 5.36 è il file XAML relativo al file `Gioca.xaml.cs`.

Anche in questo caso viene usato `<Grid>`. In alto a destra viene inserito un pulsante che, una volta cliccato, invoca il metodo `Info()` che ha sovrapposta un'immagine. In alto al centro viene inserito il punteggio.

Al centro è possibile vedere la tabella costituita da nove celle avente ciascuna un pulsante e due immagini sovrapposti, una X e un O, anche se realmente solo una delle due per ogni cella sarà resa visibile durante una partita.

Sono posizionate sopra la griglia di gioco, inizialmente invisibili, le animazioni relative alla vittoria, alla sconfitta e al pareggio.

È possibile notare, inoltre, un display, anch'esso centrato, che guiderà l'utente durante la partita.

In fondo alla pagina si trovano tre pulsanti, con le relative tre immagini sovrapposte; è possibile visionare la presenza di un pulsante per la X, uno per l'O e uno per la rivincita.

Come già visibile negli altri file XAML, i bottoni hanno un'immagine sovrapposta; ciò è dovuto al fatto che non è possibile impostare un'immagine come background di un pulsante. Va posta particolare attenzione al fatto che le immagini vanno inserite sotto i pulsanti, e non sopra, poiché, altrimenti, questo renderebbe non cliccabile il pulsante sottostante. Naturalmente il pulsante deve avere uno sfondo trasparente per permettere la visualizzazione corretta dell'immagine sottostante.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:ffimageloading="clr-namespace:FFImageLoading.Svg.Forms;assembly=FFImageLoading.Svg.Forms"
             NavigationPage.HasNavigationBar="false"
             x:Class="TrisBluetooth.Gioca"
             BackgroundImage="background3">
    <StackLayout Margin="10">
        <Grid>
            <Image Source="info"
              Aspect="AspectFit"
              VerticalOptions="Start"
              HorizontalOptions="End"
              WidthRequest="30"
               HeightRequest="30"/>

            <Label x:Name="Punteggio"
                TextColor="White"
                XAlign="Center"
                VerticalOptions="EndAndExpand"
                HorizontalOptions="Center"
                FontSize="80"
                FontFamily="IndieFlower.ttf#IndieFlower"/>

            <Button HorizontalOptions="End"
               VerticalOptions="Start"
               BackgroundColor="Transparent"
               WidthRequest="30"
               HeightRequest="30"
               Clicked="Info"/>
        </Grid>

        <Grid HorizontalOptions="Center" VerticalOptions="EndAndExpand" >

            <Image Source="griglia" Aspect="AspectFit" HeightRequest="300" WidthRequest="300"/>
            <Grid BackgroundColor="Transparent">
                <Grid.RowDefinitions>
                    <RowDefinition Height="100" />
                    <RowDefinition Height="100" />
                    <RowDefinition Height="100" />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="100" />
                    <ColumnDefinition Width="100" />
                    <ColumnDefinition Width="100" />
                </Grid.ColumnDefinitions >
                <Button x:Name="cella0" Grid.Row="0" Grid.Column="0" BackgroundColor="Transparent" />
                <Button x:Name="cella1" Grid.Row="0" Grid.Column="1" BackgroundColor="Transparent" />
                <Button x:Name="cella2" Grid.Row="0" Grid.Column="2" BackgroundColor="Transparent" />
                <Button x:Name="cella3" Grid.Row="1" Grid.Column="0" BackgroundColor="Transparent" />
                <Button x:Name="cella4" Grid.Row="1" Grid.Column="1" BackgroundColor="Transparent" />
                <Button x:Name="cella5" Grid.Row="1" Grid.Column="2" BackgroundColor="Transparent" />
                <Button x:Name="cella6" Grid.Row="2" Grid.Column="0" BackgroundColor="Transparent" />
                <Button x:Name="cella7" Grid.Row="2" Grid.Column="1" BackgroundColor="Transparent" />
                <Button x:Name="cella8" Grid.Row="2" Grid.Column="2" BackgroundColor="Transparent" />
                <Image x:Name="OImg0" Source="o" Grid.Row="0" Grid.Column="0" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg1" Source="o" Grid.Row="0" Grid.Column="1" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg2" Source="o" Grid.Row="0" Grid.Column="2" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg3" Source="o" Grid.Row="1" Grid.Column="0" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg4" Source="o" Grid.Row="1" Grid.Column="1" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg5" Source="o" Grid.Row="1" Grid.Column="2" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg6" Source="o" Grid.Row="2" Grid.Column="0" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg7" Source="o" Grid.Row="2" Grid.Column="1" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="OImg8" Source="o" Grid.Row="2" Grid.Column="2" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg0" Source="x" Grid.Row="0" Grid.Column="0" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg1" Source="x" Grid.Row="0" Grid.Column="1" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg2" Source="x" Grid.Row="0" Grid.Column="2" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg3" Source="x" Grid.Row="1" Grid.Column="0" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg4" Source="x" Grid.Row="1" Grid.Column="1" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg5" Source="x" Grid.Row="1" Grid.Column="2" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg6" Source="x" Grid.Row="2" Grid.Column="0" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg7" Source="x" Grid.Row="2" Grid.Column="1" Aspect="AspectFit" IsVisible="False"/>
                <Image x:Name="XImg8" Source="x" Grid.Row="2" Grid.Column="2" Aspect="AspectFit" IsVisible="False"/>

            </Grid>

        <ffimageloading:SvgCachedImage HorizontalOptions="Fill" VerticalOptions="Fill"
        x:Name="HaiVinto" Source="HaiVinto.gif" Margin="0"/>
        <ffimageloading:SvgCachedImage HorizontalOptions="Fill" VerticalOptions="Fill"
        x:Name="HaiPerso" Source="HaiPerso.gif" Margin="0"/>
```

```
        <ffimageloading:SvgCachedImage HorizontalOptions="Fill" VerticalOptions="Fill"
        x:Name="HaiPareggiato" Source="HaiPareggiato.gif" Margin="0"/>

            </Grid>
            <Grid HorizontalOptions="CenterAndExpand" VerticalOptions="End">
                <Label x:Name="Display" TextColor="White" XAlign="Center" FontFamily="IndieFlower.ttf#IndieFlower"
                FontSize="20"/>
            </Grid>
            <Grid HorizontalOptions="CenterAndExpand" VerticalOptions="EndAndExpand" BackgroundColor="Transparent">

                <Grid.RowDefinitions>
                    <RowDefinition Height="80"/>
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="80" />
                    <ColumnDefinition Width="140" />
                    <ColumnDefinition Width="80" />
                </Grid.ColumnDefinitions>

                <Image x:Name="XImage" Source="cerchio" Grid.Row="0" Grid.Column="0" Aspect="AspectFit" />
                <Button x:Name="X" Text="X" FontFamily="IndieFlower.ttf#IndieFlower" Grid.Row="0" Grid.Column="0"
                TextColor="White" BackgroundColor="Transparent" Clicked="Client"/>
                <Image x:Name="RivincitaImage" Grid.Row="0" Grid.Column="1" Aspect="AspectFit" Source="button"/>
                <Button x:Name="Revenge" Text="Rivincita" FontFamily="IndieFlower.ttf#IndieFlower" Grid.Row="0"
                Grid.Column="1" BackgroundColor="Transparent" Clicked="Rivincita" TextColor="Red"/>
                <Image x:Name="OImage" Source="cerchio" Grid.Row="0" Grid.Column="2" Aspect="AspectFit"/>
                <Button x:Name="O" Text="O" FontFamily="IndieFlower.ttf#IndieFlower" Grid.Row="0" Grid.Column="2"
                TextColor="White" BackgroundColor="Transparent" Clicked="Server"/>
            </Grid>
        </StackLayout>

    </ContentPage>
```

Listato 5.36. Il codice relativo al file `Gioca.xaml`

6

Manuale utente

In questo capitolo verranno illustrati gli screenshot del progetto, analizzando tutte le pagine che compongono l'applicazione TrisBluetooth in modo da facilitare l'utilizzo all'utente.

6.1 Avvio dell'applicazione

Una volta cliccata l'icona di TrisBluetooth nella schermata home (Figura 6.1) o dal menù dello smartphone, verrà avviata l'applicazione.

All'avvio apparirà direttamente la richiesta di autorizzazione per la posizione, come mostrato in Figura 6.2.

Una volta accettata l'autorizzazione si visualizzerà l'animazione nella Splash Page (Figura 6.3), ovvero uno zoom in dissolvenza del logo e della scritta "TrisBluetooth".

Va notato il fatto che, come già visibile nella Figura 6.2, l'utente può non accettare l'autorizzazione relativa alla posizione, ma questo comporterà l'impossibilità di effettuare la successiva ricerca dei dispositivi Bluetooth limitrofi.

Dopo circa 13 secondi, l'applicazione passerà automaticamente alla Main Page (Figura 6.4).

6.2 Ricerca dei dispositivi Bluetooth limitrofi

Una volta avviata la Main Page, l'utente può visualizzare le istruzioni per la ricerca del dispositivo dell'avversario (Figura 6.5), cliccando nel pulsante in alto a destra raffigurante un punto interrogativo.

Per avviare la ricerca dei dispositivi occorre cliccare il pulsante "Cerca". Una volta cliccato, verrà richiesta l'autorizzazione per l'accensione del Bluetooth, come mostrato nella Figura 6.6.

Una volta accettata l'autorizzazione, si avvierà la ricerca dei dispositivi Bluetooth limitrofi visibili e verrà attivata la visibilità del dispositivo per alcuni secondi, in modo che anche il dispositivo dell'avversario possa rilevare lo smartphone dell'utente.

Figura 6.1. Schermata home con l'icona dell'applicazione TrisBluetooth

Durante la ricerca cominceranno a comparire i dispositivi rilevati con il nome e il MAC, come in Figura 6.7. Come già riportato nel Capitolo 3, se l'avversario ha lo smartphone con lo stesso nome di un altro dispositivo Bluetooth limitrofo, occorrerà o conoscere il MAC dell'avversario o far modificare il nome allo smartphone dell'avversario.

Una volta selezionato il dispositivo avversario, questo verrà evidenziato e apparirà il pulsante "Gioca" (Figura 6.8).

6.3 Avvio di una partita

Appena cliccato il pulsante "Gioca", si passa alla pagina di gioco mostrata in Figura 6.9, la quale presenta la griglia di gioco ed il punteggio impostato a 0 per entrambi i giocatori.

L'utente può consultare le istruzioni di gioco (Figura 6.10) cliccando il pulsante in alto a destra nella schermata di gioco raffigurante un punto interrogativo.

Prima di avviare la partita occorre selezionare il segno, X o O, con il quale affronterà la partita; naturalmente, entrambi i giocatori non possono selezioanare lo stesso segno.

Se il giocatore seleziona il simbolo X avrà il privilegio di iniziare; se, invece, sceglie il simbolo O dovrà attendere la mossa dell'avversario.

A guidare i giocatori c'è il display, il quale, attraverso i messaggi "Tocca a te" (Figura 6.11) o "Attendi..." (Figura 6.12), alternerà i turni di gioco.

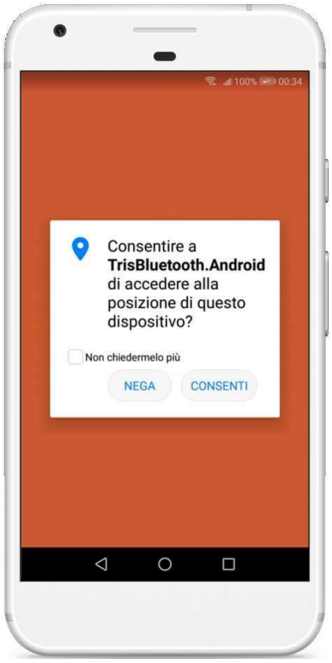

Figura 6.2. Autorizzazione all'acquisizione della posizione da parte dell'applicazione TrisBluetooth

Quando uno dei due giocatori vince, nel dispositivo vedrà apparire l'animazione della scritta "HAI VINTO", come mostrato nella Figura 6.13, e il proprio punteggio verrà incrementato di uno.

Nel dispositivo del perdente apparirà l'animazione della scritta "HAI PERSO", come evidenziato nella Figura 6.14, e il punteggio dell'avversario verrà incrementato di uno.

Nel caso di pareggio apparirà ad entrambi i giocatori l'animazione della scritta "HAI PAREGGIATO", come mostrato nella Figura 6.15.

Tutte e tre le animazioni hanno una durata di 7 secondi.

Alla fine della partita è possibile iniziarne una nuova e, per farlo, entrambi i giocatori devono cliccare il pulsante "Rivincita" (Figura 6.16) e ripercorrere l'iter dalla selezione del simbolo X o O.

Figura 6.3. Animazione dell'icona dell'applicazione TrisBluetooth nella Splash Page

Figura 6.4. Main Page dell'applicazione TrisBluetooth

Figura 6.5. Istruzioni per la ricerca del dispositivo dell'avversario

Figura 6.6. Autorizzazione all'utilizzo Bluetooth da parte dell'applicazione TrisBluetooth

Figura 6.7. Lista dei dispositivi rilevati

Figura 6.8. Selezione del dispositivo avversario in seguito alla ricerca dei dispositivi Bluetooth limitrofi

Figura 6.9. Pagina di gioco

Figura 6.10. Istruzioni di gioco

Figura 6.11. Il display afferma che il dispositivo ha il turno di gioco

Figura 6.12. Il display afferma che il dispositivo attende la mossa dell'avversario

Figura 6.13. Scritta animata in seguito alla vittoria

Figura 6.14. Scritta animata in seguito alla sconfitta

Figura 6.15. Scritta animata in seguito al pareggio

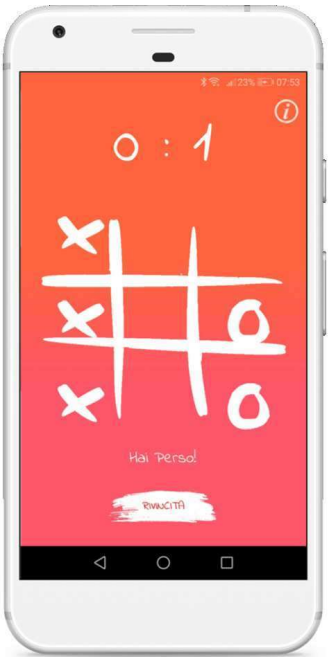

Figura 6.16. Pulsante per l'avvio della rivincita a fine partita

7

Discussione in merito al lavoro svolto

In questo capitolo verrà ripercorsa tutta l'attività di tesi, evidenziando le competenze maturate nel tempo. Nella seconda parte del capitolo verrà presentata l'analisi SWOT del sistema realizzato.

7.1 Competenze maturate durante la realizzazione dell'applicazione TrisBluetooth

Lavorare per la realizzazione dell'applicazione TrisBluetooth ha dato la possibilità di espandere il più possibile la conoscenza posseduta anche in ambiti non prettamente inerenti alla programmazione mobile.

Prendere parte a questo lavoro ha significato, infatti, dedicarsi allo sviluppo di un'applicazione "in toto", poiché si è passati dalla progettazione su carta alla realizzazione.

Per effettuare la progettazione e documentare il lavoro svolto è stato, inoltre, necessario approfondire la progettazione tramite UML, in particolar modo il diagramma dei casi d'uso e il diagramma delle attività.

Inoltre si sono anche approfondite l'analisi dei requisiti e la produzione dei mockup per poter, così, presentare il progetto ai clienti in modo da consentire loro di comprenderlo facilmente e di proporre eventuali modifiche o variazioni.

Dal punto di vista tecnico, lavorare alla realizzazione di questo progetto ha comportato affrontare molte sfide; infatti, oltre allo studio della programmazione mobile, si è dovuta approfondire la connessione Bluetooth.

Per prima cosa è stato approfondito il Bluetooth dal punto di vista teorico; in particolare, è stato visto come nasce questa tecnologia e perché, per poi comprendere come operano le connessioni Bluetooth con i diversi protocolli e quali di questi scegliere nel nostro caso.

In un secondo tempo è risultato indispensabile capire come implementare quanto appreso teoricamente al fine di realizzare un'applicazione Android mediante Xamarin.

Lavorare con i pacchetti NuGet ha comportato lo studio della relativa documentazione per comprendere fino in fondo le potenzialità del pacchetto e focalizzarle alle necessità implementative dell'applicazione TrisBluetooth.

Per rispondere alle esigenze ed ai requisiti di natura grafica, è stato necessario approfondire il linguaggio XAML, ma anche consultare progetti affini.

Importanza focale, invece, ha ricoperto il lavoro di gruppo. Infatti, è stato possibile apprendere nuove conoscenze e capacità, come lavorare in team migliorando il proprio "know how".

7.2 Analisi SWOT

L'analisi SWOT (Figura 7.1) ricopre una fase fondamentale nello sviluppo di un software.

Figura 7.1. Schematizzazione dell'analisi SWOT

L'analisi SWOT (o matrice SWOT) è una tecnica di pianificazione strategica utilizzata per aiutare una persona, o un'organizzazione, a identificare punti di forza, punti di debolezza, opportunità e minacce relative alla concorrenza aziendale o alla pianificazione di un progetto. Questa pianificazione ha lo scopo di specificare gli obiettivi dell'impresa commerciale o del progetto e identificare i fattori interni ed esterni che sono favorevoli e sfavorevoli al raggiungimento di tali obiettivi. Gli utenti di un'analisi SWOT spesso pongono e rispondono a domande per generare informazioni significative per ciascuna categoria, al fine di rendere utile lo strumento e identificarne il vantaggio competitivo.

Tale tecnica è attribuita ad Albert Humphrey, il quale ha guidato un progetto di ricerca all'Università di Stanford fra gli anni '60 e '70 utilizzando i dati forniti dalla Fortune 500.

I punti di forza e di debolezza sono fattori interni, visto che sono spesso correlati internamente, mentre opportunità e minacce, sono fattori esterni, poiché si concentrano comunemente sull'ambiente esterno. Il termine "SWOT" è un acronimo per i quattro parametri che vengono esaminati dalla tecnica, ovvero:

- *Punti di forza (**S**trengths)*: caratteristiche del lavoro o del progetto che conferiscono un vantaggio sugli altri.

- *Punti di debolezza (**W**eaknesses)*: caratteristiche del business che collocano le attività o il progetto in condizioni di svantaggio rispetto agli altri.
- *Opportunità (**O**pportunities)*: elementi nell'ambiente che l'azienda o il progetto potrebbero sfruttare a proprio vantaggio.
- *Minacce (**T**hreats)*: elementi nell'ambiente che potrebbero causare problemi all'azienda o al progetto.

Il grado di corrispondenza dell'ambiente interno del progetto con l'ambiente esterno viene espresso dal concetto di adattamento strategico. L'analisi SWOT è importante perché può supportare le fasi successive della pianificazione per raggiungere l'obiettivo; quindi, inizialmente, andrebbe considerato se l'obiettivo è raggiungibile, dati i risultati (almeno preliminari) dell'analisi SWOT. Se l'obiettivo non è raggiungibile, occorre selezionare un obiettivo diverso e ripetere il processo.

7.2.1 Punti di forza

I punti di forza sono le caratteristiche e gli elementi che possono portare il progetto al successo, inducendo gli utenti a prediligerlo ad un progetto concorrente. I punti di forza individuati per il progetto in esame sono i seguenti:

- *Non necessità della connessione Internet*, poiché l'applicazione implementa la connessione tra dispositivi attraverso il Bluetooth.
- *Grafica semplice ed efficace*, poiché l'applicazione riporta uno stile a tinta unita con semplici ed essenziali elementi di gioco (griglia, simboli di gioco, etc).
- *Implementazione per dispositivi Android*, poiché, come già riportato nel Capitolo 4, la maggior parte dei dispositivi mobili attualmente in uso adottano il sistema operativo Android e, quindi, si ha un più ampio bacino di utenti a cui è destinato il progetto rispetto al caso in cui quest'ultimo fosse stato implementato per altre piattaforme.

7.2.2 Punti di debolezza

Un punto di debolezza è un elemento che può rappresentare uno svantaggio nei confronti di un prodotto concorrente e che, quindi, può interferire con il successo dell'applicazione, facendo in modo che un'utente preferisca un altro progetto concorrente. Dall'analisi effettuata sull'applicazione TrisBluetooth sono emerse le seguenti criticità:

- *Elevata concorrenza*, poiché negli app store ci sono molte applicazioni similari.
- *Limitata distanza permessa tra i giocatori*, poiché gli utenti possono essere distanti al massimo 10 metri, come già riportato nel Capitolo 3.
- *Dimensione elevata*, poiché, avendo l'applicazione una dimensione di circa 60MB, occupa una quantità di memoria eccessiva rispetto ad altre applicazioni similari, le quali occupano dimensioni inferiori anche dell'80%.

7.2.3 Opportunità

I vantaggi che possono derivare dall'esterno (come, ad esempio, normative favorevoli, cambiamenti socio-economici, sviluppo di nuove tecnologie) che potrebbero avere risvolti positivi sul progetto sono, essenzialmente, i seguenti:

- *Elevate potenzialità di guadagno e download*: visto che, come riportato nel Capitolo 4, il mobile gaming è un settore in forte sviluppo e crescita.
- Estensione, ad altre applicazioni, della metodologia di lavoro adottate in questa tesi.

7.2.4 Minacce

Una minaccia è un fattore esterno che potrebbe risultare limitante per il prodotto. Nel caso in questione le possibili minacce al nostro progetto sono le seguenti:

- *Problemi di sicurezza*, poiché è possibile che, durante la connessione tra due dispositivi, ve ne sia un terzo che riesca ad entrare nella connessione e, quindi, intercettare i messaggi scambiati.
- *Sempre più GB a disposizione per gli utenti per la loro navigazione*: questo potrebbe comportare la predilezione nell'utilizzo di connessioni Internet a discapito di quelle Bluetooth, essendo, le prime, più semplici da implementare, più sicure nell'utilizzo e con un minore consumo di energia (batteria).

8

Conclusioni e uno sguardo al futuro

In questa tesi è stata presentata l'applicazione TrisBluetooth per i dispositivi Android, implementata attraverso il framework Xamarin, con l'obbiettivo di permettere di giocare tra smartphone anche dove la connessione Internet non è consentita o non disponibile.

Di seguito verranno ripercorse le principali tappe che hanno portato alla realizzazione del progetto TrisBluetooth.

La prima fase è consistita nell'effettuare una ricerca, sia nel settore delle tecnologie degli smartphone, sia in quello delle applicazioni simili esistenti, in modo da recepire le informazioni utili per una corretta progettazione del lavoro.

La seconda fase, forse la più onerosa, è stata effettuata attraverso uno studio di fattibilità, nel quale sono stati raccolti ed analizzati i requisiti tecnici del sistema, focalizzandosi su quelli relativi alla connessione Bluetooth tra smartphone.

La fase successiva ha richiesto uno sforzo creativo; infatti, sono stati elaborati i mockup di livello 0 e i mockup di livello 1 in modo da rappresentare il layout di ogni pagina dell'applicazione, cercando di renderlo semplice, intuitivo, ma anche elegante e con colori vivaci, tipici delle applicazioni di gioco.

La quarta fase è stata completamente dedicata allo studio delle connessioni Bluetooth, concentrandosi, soprattutto, nelle connessioni tra dispositivi mobili.

La fase successiva ha portato alla progettazione dell'applicazione; infatti sono stati progettati le classi e i file che compongono il sistema.

La sesta fase è consistita nell'implementazione dell'applicazione precedentemente progettata definendo i file XAML tenendo conto dei mockup realizzati in precedenza.

In conclusione, è stato stilato un manuale utente, avvalendosi di alcuni screenshot rappresentativi dell'applicazione finale.

Come già riportato nel Capitolo 4, il mobile gaming è un settore in continuo sviluppo ed in continua evoluzione, che comporta, per questo motivo, continue idee e continui aggiornamenti dei prodotti offerti.

L'implementazione del progetto oggetto della presente tesi potrebbe essere la base per l'implementazione di altre applicazioni che facciano uso della connessione Bluetooth, non obbligatoriamente giochi di società.

Ricopre un fattore di importante rilievo il fatto che l'applicazione riporti una possibile implementazione Bluetooth cross-platform. I progetti a supporto di al-

tri utenti, e quindi consultabili da altri programmatori, nell'implementazione delle connessioni Bluetooth non sono molti.

Va, sicuramente, posta particolare attenzione alla possibilità di far autenticare gli utenti, permettendo, così, di salvare i risultati delle varie partite nonchè di effettuare statistiche relative alle partite vinte e a quelle perse, nonché di stilare classifiche di gioco.

Un altro possibile sviluppo potrebbe riguardare l'implementazione di un menù che dia la possibilità di cambiare lingua, nonchè di modificare impostazioni grafiche, ma anche, una volta inseriti dei suoni, modificare volume e altre impostazioni audio.

La dimensione dell'applicazione è una caratteristica importante per offrire un prodotto di qualità agli utenti; per tale motivo andrebbe ottimizzata l'implementazione grafica in modo da ridurre la dimensione dell'applicazione, non inficiando molto sull'esperienza utente.

Un ultimo miglioramento è relativo allo User Experience Design; infatti, ci si potrebbe prefiggere di estendere i layout per supportare le svariate risoluzioni dei dispositivi Android.

Per realizzare i miglioramenti menzionati in precedenza ci sono, chiaramente, una serie di competenze che vanno acquisite.

Sicuramente, per implementare l'autenticazione dei giocatori, occorrerà la creazione di un database e, quindi, lo studio e la comprensione di un DBMS, come MySQL, e di un linguaggio di programmazione generale, come PHP.

Per effettuare statistiche di gioco e classifiche sarà, inoltre, necessario approfondire tecniche di grafica.

Per inserire un gingle di sottofondo occorre idearlo attraverso uno studio del suono, e poi realizzarlo attraverso software dedicati.

Un'altra possibile competenza da sviluppare riguarda la progettazione dell'applicazione TrisBluetooth per altre piattaforme, come iOS o Windows Phone.

Riferimenti bibliografici

1. Analisi SWOT. https://it.wikipedia.org/wiki/Analisi_SWOT, 2019.
2. autorizzazioni Android. https://www.androidpit.it/app-android-autorizzazioni, 2019.
3. connettività Bluetooth. https://stackoverflow.com/questions/40395162/xamarin-forms-how-to-develop-an-application-with-bluetooth-connectivity-in-xama, 2019.
4. Cordova. https://cordova.apache.org, 2019.
5. DependencyService. https://docs.microsoft.com/it-it/xamarin/xamarin-forms/app-fundamentals/dependency-service/introduction, 2019.
6. Ionic. https://ionicframework.com/docs/intro, 2019.
7. listview. https://docs.microsoft.com/it-it/xamarin/xamarin-forms/user-interface/listview/data-and-databinding, 2019.
8. MessagingCenter. https://docs.microsoft.com/it-it/xamarin/xamarin-forms/app-fundamentals/messaging-center, 2019.
9. Mobile Gaming. https://www.appannie.com/en/insights/market-data/q2-2019-mobile-games-market-index/, 2019.
10. navigation. https://docs.microsoft.com/it-it/xamarin/xamarin-forms/app-fundamentals/navigation/hierarchical, 2019.
11. Phonegap. https://phonegap.com/about/, 2019.
12. tableview. https://docs.microsoft.com/it-it/xamarin/xamarin-forms/user-interface/tableview, 2019.
13. J. Anderson. *Appcelerator Titanium Up and Running*. O'Reilly Media, 2013.
14. M. Bonifazi, F. Collini, A. Martellucci, and S. Sanna. *Android. Programmazione avanzata*. Edizioni LSWR, 2015.
15. Fu Cheng. *Build Mobile Apps with Ionic 4 and Firebase Hybrid Mobile App Development*. Apress, 2018.
16. D. Cope. *Appcelerator Titanium Application Development by Example Beginner's Guide*. Packt Publishing, 2013.
17. E. Frontoni and A. Mancini. *Programmazione ad oggetti per l'ingegneria informatica*. Mc Graw Hill, 2019.
18. D. Hermes. *Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals*. Apress, 2015.
19. P. F. Johnson. *Cross-platform UI Development with Xamarin.Forms*. Packt Publishing, 2015.
20. J. Kneen. *Appcelerator Titanium Smartphone App Development Cookbook 2nd Edition Over 100 recipes to help you develop cross-platform, native applications in JavaScript*. Packt Publishing, 2015.

21. R. Morrow. *Bluetooth: Operation and Use*. McGraw-Hill Education, 2002.
22. Nishanth Nair and Ragini Kumbhat Bhandari. *Building Mobile Applications Using Kendo UI Mobile and ASP.NET Web API*. Packt Publishing, 2013.
23. S. Olson, J. Hunter, and B. Horgen. *Professional Cross-Platform Mobile Development in C#*. Wrox, 2012.
24. M. Reynolds. *Xamarin Essentials*. Packt Publishing, 2014.
25. K. Shotts. *Mastering PhoneGap Mobile Application Development*. Packt Publishing Limited, 2016.
26. Ed Snider. *Mastering Xamarin.Forms*. Packt Publishing, 2018.
27. Syngress. *Bluetooth Application Developer's Guide*. Elsevier, 2002.

Ringraziamenti

Ogni meta necessita di un percorso per essere raggiunta; questo percorso per me è iniziato anni fa e, seppur tortuoso e impervio, è ormai giunto al termine; d'altronde:

"Per aspera sic itur ad astra"

È stata un'esperienza ricca di sacrifici e ansie ma soprattutto di gioie e soddisfazioni.

In conclusione alla presente tesi intendo rivolgere un ringraziamento a tutti coloro che, durante questo percorso, mi sono stati vicini e sono stati presenti nella mia vita.

In primis ai miei genitori Sauro e Fabiola che mi hanno reso quello che sono oggi e mi hanno donato la possibilità di realizzare tutto questo.

Alle mie sorelle Federica e Silvia, le quali, vivendo un'esperienza simile alla mia, mi hanno sopportato e supportato nei momenti di sconforto e smarrimento.

Ai miei colleghi che hanno condiviso con me questo percorso arricchendolo di risate; in modo particolare a Damiano, compagno di molti progetti.

Ai miei amici che mi sono stati sempre di sostegno.

A Luca, sempre disponibile ad aiutarmi.

Al professore Domenico Ursino, un esempio ed un modello morale, il quale mi ha fatto appassionare allo sviluppo mobile e mi ha dato la possibilità di realizzare la presente tesi e la relativa applicazione Xamarin.

E infine a tutti coloro che mi hanno incoraggiato e mi hanno dato supporto seppur in minima parte, poiché anche i piccoli passi contribuiscono al raggiungimento della meta.

Grazie di cuore!

Francesco