ALMA MATER STUDIORUM • UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in Informatica

Client VoIP su Windows Mobile in C#

Relatore:

Dott. Vittorio Ghini

Presentata da:

Andrea Tassinari

Sessione III

Anno Accademico 2010-2011

Sommario

Introduzione

Da ormai qualche tempo la comunicazione vocale tra persone lontane tra loro è diventata routine di tutti i giorni, e con telefoni, cellulari e internet oramai è difficile non poter più comunicare. Partendo dagli albori con i primi telefoni via cavo si è passati per i cellulari ed infine per il VoIP, l'ultima frontiera nel campo della telecomunicazione.

Il VoIP (Voice over IP) come spiegherò meglio più tardi, permette la comunicazione audio tra due o più utenti passando per una rete come internet.

Quello che però abbiamo visualizzato è stato: Una persona con il suo cellulare può chiamare liberamente un altro individuo munito di numero telefonico e allor stesso modo il medesimo individuo può fare la stessa cosa però con un client VoIP. Ma quando manca la connessione la conversazione si perde in entrambi i casi. Quindi, è possibile partire da uno dei due punti e trovare un modo per cui la rete diventi più affidabile, e non ci siano interruzioni di conversazione? Si è pensati a un modo per gestire una comunicazione VoIP usando non una singola via di trasmissione ma inserendo nel mezzo una applicazione proxy che gestisse più sistemi di comunicazione, come reti wireless, reti telefoniche, connessioni ad altri dispositivi che permettessero un accesso alla rete, ecc… Questo problema è già stato affrontato e risolto su alcune piattaforme, ma nessuno fin ora

aveva provato a farlo su un sistema Windows Mobile ed ecco lo scopo di questo progetto.

L’obbiettivo finale era chiaro, ma come è logico pensare i problemi da affrontare sono tanti e per farlo nel modo giusto prima bisogna avere chiaro quando si parla di Windows Mobile, di cosa si sta parlando, di quali strumenti si dispone e di come procedere.

Capitolo 1

VoIP

Con il termine VoIP (Voice over Internet Protocol) si denomina una tecnologia che permette di avere conversazioni telefoniche utilizzando reti basate su protocollo IP per il trasferimento dei dati, come ad esempio la rete Internet. Più nel dettaglio VoIP identifica tutti i protocolli che concorrono al conseguimento di tale obbiettivo e grazie a numerosi provider VoIP è oggi possibile effettuare chiamate anche verso la tradizionale rete telefonica.

Per prima cosa sarebbe utile guardare più nello specifico quali vantaggi comporta tutto questo. Prima del VoIP uno dei grossi problemi era l'obbligo di dover lasciare un minimo di banda riservata per le telefonate. Questo problema con VoIP viene abolito poiché c'è una distribuzione dinamica delle risorse di comunicazione e quindi una ottimizzazione notevole dei canali. Durante una comunicazione VoIP la trasmissione del segnale avviene attraverso l'invio e la ricezione di pacchetti, contenenti le informazioni vocali, codificati in forma digitale e questo solo quando è necessario.

Altri vantaggi derivanti dall'uso su larga scala di questa tecnologia si riscontrano in:

- Costi delle chiamate ridotti soprattutto sulle lunghe distanze
- Costi delle infrastrutture minori in quanto si necessita solo di una rete IP e null'altro
- Possibilità di nuove features, le cui implementazioni non necessiteranno di un cambiamento di Hardware

Prima abbiamo detto che per la conversazione in VoIP basta una rete basata su protocollo IP, e con questo puntualizziamo subito che non necessariamente la rete in questione deve essere internet, difatti possono essere usati anche altri mezzi di trasmissione come una rete LAN.

Detto questo possiamo quindi dedurre correttamente che l'uso di questa tecnologia, per molti aspetti, riduce sensibilmente i costi per gli utenti.

Con la tecnologia VoIP si vengono a creare ulteriori vantaggi quali:

- L'eliminazione della distinzione tra chiamate locali e a lunga distanza
- La possibilità di avere su un unico collegamento più numeri telefonici associati
- Salvataggio delle conversazioni su dispositivi digitali quali i computer
- E la possibilità di poter anche rendere gratuite le chiamate tra utenti dello stesso fornitore.

Più nel dettaglio.

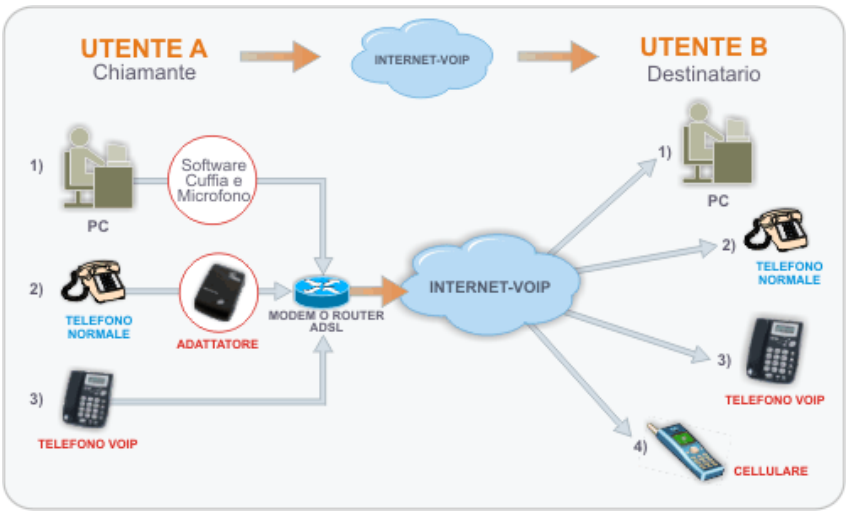


Figura 1 VoIP

Il funzionamento di questa tecnologia richiede la presenza e il funzionamento in parallelo di due protocolli di comunicazione. Uno per la trasmissione dei dati, nello specifico pacchetti voce sulla rete IP, e l'altro per il monitoraggio della conversazione, che si identifica in tante

operazioni, come identificare il chiamante, sincronizzare i client, codificare e ricostruire i messaggi audio.

Per il trasporto dei dati si è giunti a uno standard trovato nel protocollo RTP (Real-time Transport Protocol), mentre per il secondo protocollo non si è ancora arrivati a una standardizzazione.

Per la standardizzazione di questo protocollo al momento sono coinvolti tre enti internazionali di standardizzazione:

- ITU (International Telecommunication Union)
- IETF (Internet Engineering Task Force)
- ETSI (European Telecommunication Standard)

Al momento la gestione delle chiamate voce sulla rete IP è indirizzata verso due proposte, H.323 e SIP(Session Initiation Protocol).

Lo scontro viene alimentato dalle convinzioni dei sostenitori di entrambi i protocolli. H.323 è nato prima e quindi ha ottenuto il supporto di tutti i fornitori di apparati VoIP, mentre dall'altra parte i sostenitori di Sip si fregiano del minor consumo di risorse durante l'attivazione della chiamata e rendendo noto il dubbio che l'interoperabilità tra prodotti di diverse case produttrici non sia effettiva con l'utilizzo del primo protocollo.

Bisogna ricordare che la nascita dei due protocolli aveva differenti obbiettivi, difatti Sip è stato pensato e sviluppato come protocollo per la comunicazione real-time su IP e possiede delle funzioni base di controllo sulla chiamata, come ad esempio l'instaurazione e terminazione di sessione, operazioni di segnalazione, tono di chiamata, chiamata in attesa, identificazione chiamante, e cosi via. Mentre H.323 ha messo le fondamenta e la struttura per ospitare sessioni multimediali allo scopo di svolgere videoconferenze, comprendendo la definizione di formati di codifica a livello applicativo, definizione di protocolli per la segnalazione e controllo, gestione degli aspetti di sicurezza ed il tutto in riferimento ad architetture di rete locali.

1.1 H.323

H.323 e tutte le sue feature sono state inizialmente l'unica soluzione standard adottata dai produttori di telefonia su IP, in più questa tecnologia è supportata non solo da dispositivi di rete, come i router, ma anche da terminali utenti come IP Phone e applicativi PC. Pur essendo nata successivamente la proposta SIP, grazie prevalentemente alla sua ottima integrazione con gli altri protocolli della suite TCP/IP ed alla sua maggiore semplicità, sta riscontrando sempre più consensi. H.323, difatti, nasce in ambito telefonico dove le specifiche sono complete, precise e estremamente complicate. Altra cosa importante è che questo protocollo include al suo interno altri componenti precedentemente definiti dall'ITU e questo non può che aumentarne la complessità notevolmente.

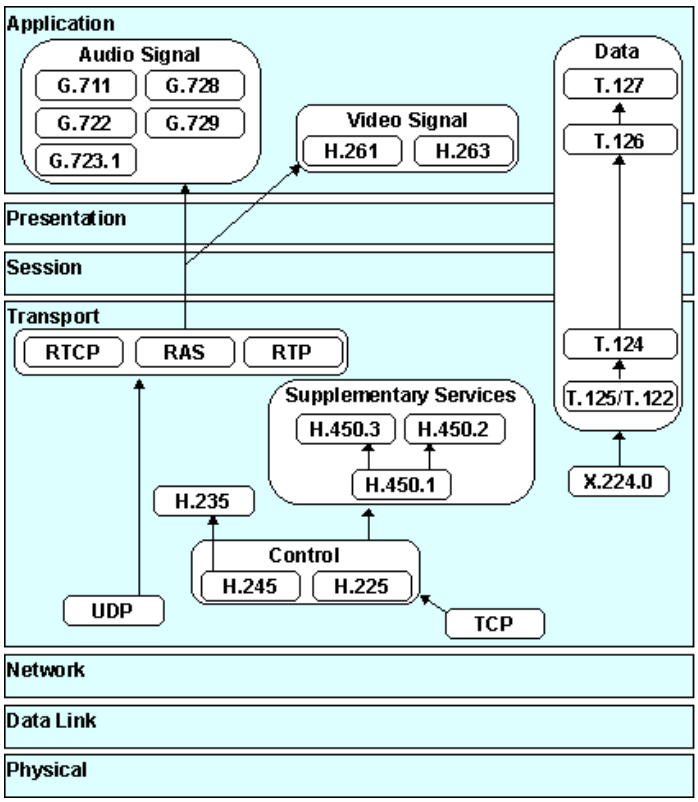


Figura 2 Funzionamento H.323

In questo piccolo schema si può notare quante componenti partecipano allo standard H.323.

1.2 Dallo standard al mercato

Nonostante la diffusione dello standard H.323, il mercato dei dispositivi per l'utenza finale, del software e dei gateway per la connessione alla rete PSTN (Public Switched Telephone Network) si è ormai orientato verso le soluzioni basate su SIP. Quasi la totalità dei prodotti in commercio oggi giorno supporta entrambi gli standard, ma è da evidenziare come alcuni enti di standardizzazione come 3GPP e UMTS abbiano incluso SIP nelle loro specifiche.

La cosa strana è che nonostante tutto questo sistema di standard, alcuni produttori di software abbiano deciso di andare controcorrente e non attenersi agli schemi, come per esempio Skype, la quale con oltre 100 milioni di utenti è riuscita a sfruttare il protocollo e a creare una rete chiusa che permette chiamate solo al suo interno o a numeri della PSTN e grazie al suo protocollo proprietario che usa nel proprio software ha potuto allargare la sua attività di gestione di partnerships con i produttori di hardware e sviluppatori di software.

Altri protocolli usati per la codifica della segnalazione della conversazione sono:

- Skinny Client Control Protocol, prodotto dalla Cisco
- Megaco e MAGCP
- MiNET di casa Mitel
- Inter Astersk Xchange
- XMPP usato da Google Talk

1.3 RTP

Abbiamo parlato finora dei protocolli di "segnalazione" di una conversazione, ma sarebbe meglio fare chiarezza anche sul protocollo RTP (Real-time Transport Protocol).

In telecomunicazione l'RTP è un protocollo del livello applicazioni usato per i servizi di comunicazione in tempo reale su Internet. Questo protocollo permette la distribuzione di servizi che necessitano di trasferimento in tempo reale come l'interattività audio e video. Di questi servizi fanno parte

anche l’identificazione del playload type, la numerazione sequenziale, la marcazione temporale e la monitorazione.

Questo protocollo è stato sviluppato da un gruppo interno all’IETF e pubblicato nel 1996.

Inizialmente il protocollo doveva essere solo di multicast, ma venne più spesso impiegato nelle applicazioni unicast(per multicast si intende distribuzione in linea a più ricevitori, mentre con unicast si intende a singolo destinatario).

Questo protocollo basato su UDP (User Datagram Protocol) viene usato con RTCP (RTP Control Protocol) che monitora la quantità del servizio e trasporta le informazioni inerenti ai partecipanti di una sessione. RTCP è sufficiente per le sessioni “loosely controlled” in cui non c’è un reale controllo dei partecipanti e set-up della sessione e non è nemmeno necessario che tutti i requisiti di controllo siano soddisfatti. Fondamentalmente per questo motivo il protocollo RTP può essere coadiuvato da un protocollo per la gestione delle sessioni come i due esplicati prima: SIP e H.323.

1.4 Vantaggi Svantaggi Problemi Soluzioni

Tornando con lo sguardo sull’argomento iniziale, VoIP, una delle questioni importanti è la possibilità di intercettazioni delle conversazioni. Difatti con VoIP ci sono dei vantaggi significativi per la tutela della privacy. In molte applicazioni VoIP, il traffico vocale, viaggia attraverso una rete la quale non offre una struttura fissa dove i pacchetti viaggiano e quindi il rischio di intercettazioni nel canale di comunicazione diventa minimo. Tuttavia questo non è sempre vero, e per motivi a volte tecnici e a volte dati dalle pressioni di organi di polizia sui gestori dei sistemi VoIP, la comunicazione viene vincolata e quindi diventa possibile l’intercettazione. Nonostante ciò una sessione VoIP può essere instaurata utilizzando canali sicuri, che con sistemi di crittografia a chiavi asimmetriche e firme digitali non permettono a terzi di comprendere il contenuto della conversazione anche se questi riuscissero a intercettare tutti i pacchetti che la compongono (ovviamente è da ricordare che l’impossibilita matematica non esiste visto che per forza di cose una chiave che apre un codice cifrato esiste sempre).

Altri problemi e questioni aperti.

Le reti IP non dispongono di algoritmi e meccanismi che garantiscano l'ordine di ricezione identico a quello di invio, tantomeno una garanzia relativa alla qualità del servizio. Questo genera i due problemi con la quale chi sviluppa applicativi per VoIP combatte fin da subito: la lentezza e la perdita di parte dei dati.

Il problema di fondo del VoIP è la corretta ricostruzione dello streaming audio, assicurando che quest'ultimo rispetti la coerenza temporale. Altro problema è mantenere un tempo di latenza dei pacchetti sufficientemente basso affinché l'utente non debba aspettare troppo tempo prima di ricevere le risposte durante la conversazione.

A tali problemi di ritardo e soprattutto di variabilità di ritardo e la conseguente necessaria computazione per il riordinamento e la ricostruzione viene incontro MPLS, un protocollo che instaura connessioni nella rete di trasporto grazie alla commutazione per etichetta e all'utilizzo di meccanismi di equalizzazione del ritardo tramite ai buffer di payload.

Alcuni punti chiave da osservare per migliorare la comunicazione sono:

- Alcuni router possono essere preimpostati per riuscire a distinguere i pacchetti VoIP dal resto e quindi dare a questi una maggiore priorità di trasmissione qual ora ci sia una congestione.

- È possibile memorizzare in buffer i pacchetti per rendere la trasmissione meno vulnerabile alla perdita di informazioni, ma con un conseguente aumento della latenza come per esempio la trasmissione satellitare.

- Una gestione della rete che mantenga una larghezza di banda sufficientemente ampia agevolerebbe di molto le conversazioni, ma è difficile applicare questa operazione qual ora il mezzo di comunicazione sia internet.

- La privacy potrebbe essere compromessa qual ora ci siano attacchi del tipo "man in the middle", e questo viene reso ancora meno riscontrabile qual ora il codice del client VoIP sia proprietario e non open source.

1.5 SIP

Passiamo ora a vedere più nel dettaglio la tecnologia SIP poiché è questa che è stata usata nel progetto.

Session Initiation Protocol (SIP) è un protocollo basato su IP definito dalla RFC 3261. Attraverso questo protocollo è possibile trasferire dati di diverso tipo (audio, video, messaggistica testuale…). Inoltre favorisce un'architettura modulare e scalabile.

Per l'avvio e la terminazione della connessione SIP supporta cinque features:

- Localizzazione dell'utente: è in grado di determinare gli end system usati
- Disponibilità dell'utente: si è in grado di comprendere se l'utente è disponibile per la comunicazione
- Capacità dell'utente: riesce a stimare una media e i parametri usati
- Impostazione della sessione: avviso, instaurazione dei parametri di una sessione su chiamate
- Gestione della sessione: trasferimento e terminazione di una sessione, modifica dei parametri della sessione e invocazione dei servizi.

Detto questo guardiamo ora più nel tecnico cosa offre e come si comportano i vari meccanismi all'interno.

Al contrario del protocollo UDP, con la quale è nato, che usava di default la porta 5060, recenti versioni di questo standard permettono anche l'uso di TCP e TLS.

Il metodo usato per la comunicazione è text-based, derivato dall'http. Per l'instaurazione della connessione avviene un three-way handshaking simile al protocollo TCP. Grazie a questo tipo di impostazione l'uso del protocollo trova compimento sia in apparati client-server che in connessioni peer to peer. Altre caratteristiche da sottolineare sono la facilità di estendibilità e programmabilità, e l'indipendenza dal protocollo di trasporto. Nelle comunicazioni SIP i messaggi sono o richieste o risposte. Le transazioni sono una sequenza di richieste e una o più risposte. Tali transazioni posso

essere identificate tramite il transaction-ID, che come una chiave primaria specifica la sorgente, la destinazione e il numero di sequenza

Per l'aspetto di mobilità SIP è dialog-oriented: un dialogo è una relazione presente tra entità parietiche che si scambiano richieste e risposte in un contesto comune.

Le entità usate nel protocollo sono:

- SIP User Agent: è un end-point e può fungere da client o da server; i due ruoli sono dinamici, nel senso che nel corso di una sessione un client può fungere da server e viceversa. Quando funge da client, dà inizio alla transazione originando richieste. Quando funge da server, ascolta le richieste e le soddisfa, se possibile.
  Uno User Agent è in sostanza una macchina a stati, che evolve in dipendenza di messaggi SIP, e registra informazioni rilevanti del dialogo. Il dialogo ha inizio quando si risponde positivamente al messaggio di Invite e termina con un messaggio di Bye.

- Registrar Server: è un server dedicato o collocato in un proxy. Quando un utente è iscritto ad un dominio, invia un messaggio di registrazione del suo attuale punto di ancoraggio alla rete ad un Registrar Server.

- Proxy Server: è un server intermedio; può rispondere direttamente alle richieste oppure inoltrarle ad un client, ad un server o ad un ulteriore proxy. Un proxy server analizza i parametri di instradamento dei messaggi e "nasconde" la reale posizione del destinatario del messaggio - essendo quest'ultimo indirizzabile con un nome convenzionale del dominio di appartenenza.
  I proxy possono essere stateless o stateful. Quando uno User Agent invia sistematicamente le proprie richieste ad un proxy "vicino" (di default) allora il proxy viene detto Outbound-Proxy. Viceversa, un Inbound-Proxy è un proxy che instrada le chiamate entranti in un dominio. Infine un Forking-Proxy può instradare una stessa richiesta in parallelo o in sequenza a più destinazioni.

- Redirect Server: reinstrada le richieste SIP consentendo al chiamante di contattare un set alternativo di URIs.

- Location Server: è un database contenente informazioni sull'utente, come il profilo, l'indirizzo IP, l'URL.

1.6 PjSip

PJSIP è una libreria che implementa anche il protocollo SIP Open Source, studiato e disegnato per essere molto performante e flessibile.

Anche se pubblicato nel 2003, la sua storia ebbe inizio molto tempo prima. L'autore creò un protocollo SIP nel 1999 durante l'era di RFC 2543, e dopo molti esperimenti e differenti approcci nella programmazione, ed anche dopo le evoluzioni del protocollo SIP stesso, l'attuale terza generazione di PJSIP può essere considerata stabile in termini di design ed incorpora tutte le modifiche e le considerazioni avute nel corso degli anni.

Nel dettaglio, PJSIP consiste in una collezione di molte APIs, ognuna delle quali costruita a strati sovrapposti. Per questo motivo, all'utente non esperto, il primo approccio apporta non poche difficolta nel cercare un punto di partenza.

Per aiutare nella comprensione ci sono due strade possibili.

Una è la PJSUA API – Hight Level Softphone API la quale comprende in se tutti i componenti SIP e si interpone creando una API ad alto livello adatta alla creazione di tipiche applicazioni SIP user agent. PJSUA API è dotata di strumenti di facile utilizzo per:

- Registrazione di client multipli (accounts)
- Sessioni SIP e multimediali ad alto livello (chiamate)
- Lista dei conoscenti con servizio di instant messaging
- Usare agevolmente e in maniera molto potente strumenti per la manipolazione dei media

Tutto questo è possibile pur mantenendo uno spazio riservato a personalizzazioni di alcuni componenti, necessari per l'utilizzo di alcune applicazioni.

L'altra opzione possibile è usare direttamente PJSIP e PJMEDIA

Per ottenere una flessibilità e una potenza all'avanguardia la via da percorrere è questa. Lo svantaggio sarà ovviamente derivato dall'apprendimento meno diretto e rapido.

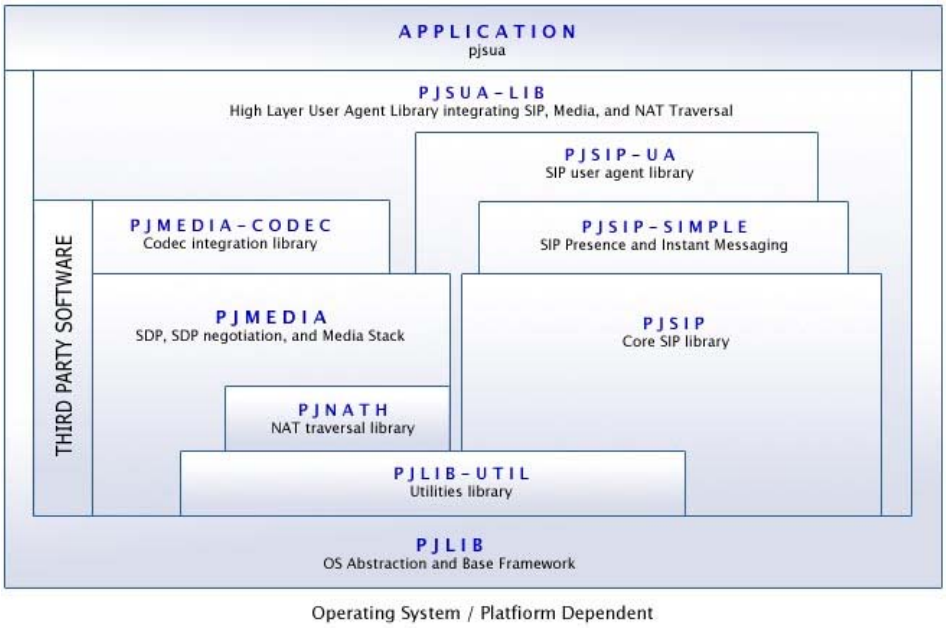


Figura 3 PJSUA: Struttura

Detto questo guardiamo un attimo l'architettura del protocollo.

Da qui risulta ancora più facile vedere come PJSUA mascheri tutto il protocollo e le sue componenti con una serie di ordinate e facili API.

PJSIP è da dire che è una libreria scritta in c++ e che con il passare degli anni ha trovato usi in diverse piattaforme. È difatti da notare che nonostante la libreria che si usa nel programma sia scritta in c++, l'accesso alle api viene fatto da codice c#. Un po' merito del fatto che la libreria sia stata riscritta con un c++ adeguato ai compilatori Microsoft e un po' merito anche del fatto che le potenzialità di trasformazione in linguaggio LI adottato dalla Microsoft ha reso possibile l'uso di più linguaggi per la creazione di applicazioni.

Capitolo 2

Windows: CE, Mobile

Prima di affrontare questo argomento, sarebbe opportuno fare luce su una particolarità che sta alla base di Windows CE. La caratteristica fondamentale di Windows CE è che più che un vero e proprio sistema operativo, si può definire più una sorta di collezione di moduli i quali scelti, uniti e compilati possono dare vita ai prodotti finiti quali i sistemi operativi. Questo spiega come da Windows CE nascano cosi tanti prodotti che pertanto di comune abbiano la sorgente, poi difficilmente riescano a trovare somiglianze interne.

Windows Phone, chiamato prima Windows Mobile, e ancor prima Pocket Pc, è il sistema operativo compatto di casa Microsoft, basato sulle api Win32. Questo sistema operativo può essere montato su dispositivi Smartphone, Pocket Pc e Portable Media Center.
Il termine Windows CE (CE=Compact Edition) è il nome ben più tecnico con cui si identifica la piattaforma generale di sviluppo di questo sistema Operativo.

2.1 Windows CE

Essendo “Windows CE” abbastanza flessibile e modulare, sono state prodotte diverse versioni per dispositivi con scopi differenti, i quali spesso e volentieri usano non solo processori x86 ma anche MIPS, ARM, SuperH e Hitach.

Le versioni prodotte portano quindi nomi differenti commerciali, come sono “MS Handheld 3.0”, “Microsoft Pocket PC 2000”, “MS Smartphone 2002”, “MS Windows Mobile 2003”, “MS Winsows Mobile 5.0” e altri ancora. Tutte queste varianti fanno riferimento a specifiche evoluzioni della piattaforma di riferimento “Windows CE”, la quale è passata dalla versione 1.0 alla 5.0 del 2006 e alla odierna 6.0.

Tutto questo ambiguo e contorto lavoro porta molti dubbi e la chiarezza viene sempre più compromessa.

Tale ambiguità viene ancor più sottolineata dal fatto che la piattaforma Windows CE è largamente distribuita a tutti i produttori di sistemi embedded (come possono essere robot industriali o citofoni), mentre Windows Mobile è molto più ristretta in termini di licenza ed uso. Pertanto Windows Mobile sia un diretto discendente di questa piattaforma, la struttura interna differisce e non di poco dal punto di vista della programmazione e quindi applicativi scritti per Windows Mobile il più delle volte non sono compatibili su Windows CE anche a parità di architettura del processore. Cosa da sottolineare, Windows Mobile è stato scritto solo per processori ARM. L’unico modo per avere una portabilità in questo senso è quello di scrivere programmi avvalendosi di un linguaggio tra c# e VB .NET, poiché l’unico anello di giunzione al momento è il .NET Compact Framework.

2.2 Windows Mobile

Per quanto riguarda Windows Mobile 6.0 ne esistono tre versioni, ognuna seguita da un suffisso per disambiguarle:

- Windows Mobile 6.0 Classic: studiata e creata per dispositivi palmari senza supporto telefonico;
- Windows Mobile 6.0 Standard: adatta invece per gli smarth phone, con il quale si identificano i terminali “data centric” non dotati di touchscreen;
- Windows Mobile 6.0 Professional: ultima versione per device tascabili dotati di una ricca collezione di tool quali lo schermo sensibile al tocco e classificati quindi nella categoria “data centric”.

Una cosa da precisare, di non poco conto, è che nonostante Windows Mobile in questione sia alla versione 6.0, il sistema base Windows CE di riferimento usato è solo il 5.0 e non l'odierno 6.0.

Per fare un po' più di luce sull'aspetto del versionamento dei vari sistemi operativi citati vorrei riportare una tabella riguardante i soli sistemi operativi per alcuni dispositivi mobile e far notare come le versioni si spalmano nel tempo.

	Pocket PC 2000	**Pocket PC 2002**	**Windows Mobile 2003**	**Windows Mobile 2003 SE**	**Windows Mobile 5.0**	**Windows Mobile 6**	**Windows Mobile 6.1**	**Windows Mobile 6.5**
Pocket PC (Senza dispositivo telefonico)	Pocket PC 2000	Pocket PC 2002	Windows Mobile 2003 for Pocket PC	Windows Mobile 2003 for Pocket PC SE	Windows Mobile 5.0 for Pocket PC	Windows Mobile 6 Classic	Windows Mobile 6.1 Classic	N/A
Pocket PC (Munito di dispositivo telefonico)	Pocket PC 2000 Phone Edition	Pocket PC 2002 Phone Edition	Windows Mobile 2003 for Pocket PC Phone Edition	Windows Mobile 2003 SE for Pocket PC Phone Edition	Windows Mobile 5.0 for Pocket PC Phone Edition	Windows Mobile 6 Professional	Windows Mobile 6.1 Professional	Windows Mobile 6.5 Professional
Smartphone (Senza touch screen)	N/A	Smartphone 2002	Windows Mobile 2003 for Smartphone	Windows Mobile 2003 SE for Smartphone	Windows Mobile 5.0 for Smartphone	Windows Mobile 6 Standard	Windows Mobile 6.1 Standard	Windows Mobile 6.5 Standard

2.3 Riassumendo

Un'altra utile visione la si può ottenere guardando una linea del tempo di Windows CE e le sue innovazioni principali passo passo.

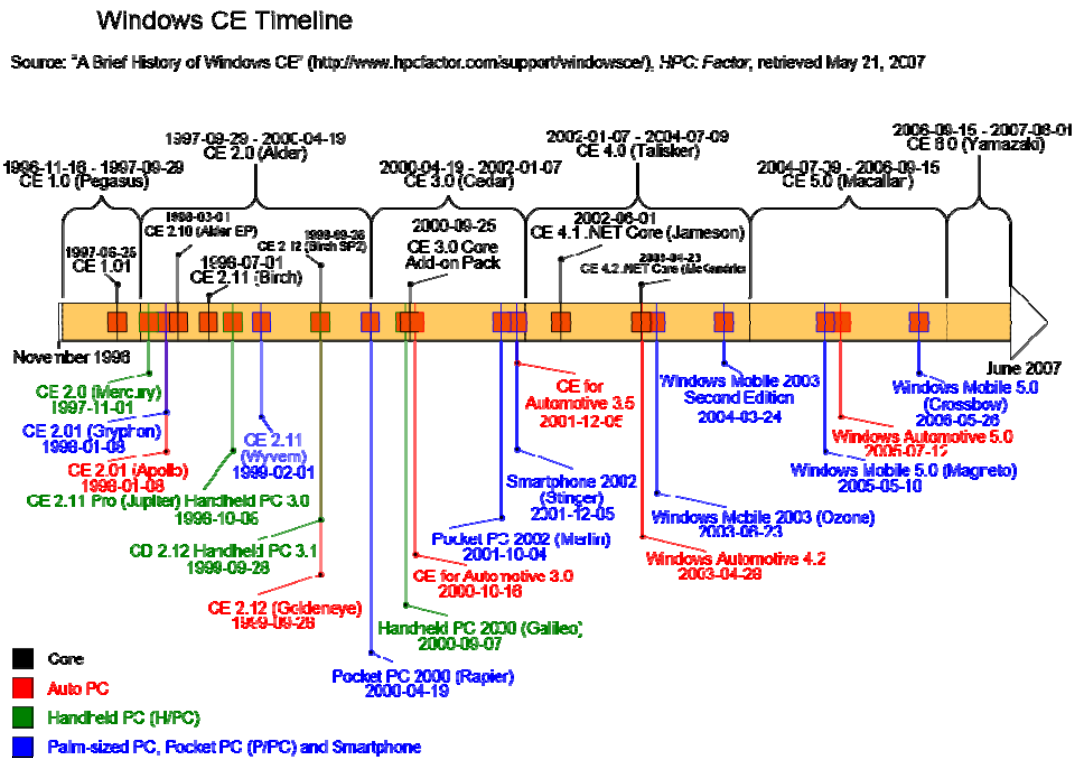


Figura 4 Linea del tempo delle evoluzioni di Windows CE

Per leggere questa visione panoramica bisogna guardare come i Core, che sarebbero la tecnologia di Windows CE, scandiscano il tempo in porzioni e nelle quali nascono prodotti per le varie architetture. Questa linea del tempo si aggiorna solo fino al 2007 e difatti pertanto sia segnalata la presenza del Windows CE 6.0 non compaiono prodotti derivati da esso, benché ce ne siano. Ai fini della spiegazione non sono necessari questi dettagli poiché la tecnologia in uso per questo progetto si arresta prima, visto che delle innovazioni su questi sistemi operativi sono state fatte usando Windows CE 5.0.

La cosa che lascia perplessi è come nonostante la Microsoft disponesse di un software base nuovo come Windows CE 6.0 ha mantenuto la versione precedente per produrre sistemi operativi per dispositivi telefonici. Come ricordato e ribadito anche prima la versione di Windows mobile 6.5 è basata su Windows CE 5.0.

Capitolo 3

.NET Framework

Il .Net Framework è l'ambiente per la creazione, distribuzione ed esecuzione di tutti gli applicativi che supportano .Net, sia applicazioni che Servizi Web.

Creato dai programmatori Microsoft e rilasciato il 13 febbraio 2002 con la versione 1.0 per i principali sistemi operativi del tempo (Windows 98, Windows NT 4.0, Windows 2000 e Windows XP), questo framework includeva al suo interno:

- Compilatori per i principali linguaggi supportati da Microsoft;
- Ambiente di esecuzione Common Language Runtime o CLR
- Libreria di classi.

I compilatori disponibili erano per c#, Visual Basic .NET, Javascript e J#. Oltre a questi era possibile però aggiungere altri compilatori aggiuntivi come per Delphi, Lisp e Eiffel i quali andavano richiesti dai rispettivi produttori.

La Common Language Specification (CLS), descrive un sottoinsieme della CLR che i compilatori devono supportare per permettere interoperabilità tra i diversi linguaggi di programmazione.

Il CLR è il motore d'esecuzione sulla piattaforma .NET, difatti è questo lo strumento che esegue codice IL (intermediate Language) compilato con i compilatori che posso avere come target il CLR. Al 2005 più di 40 sono i linguaggi supportati.

Questa macchina virtuale è usata principalmente nei sistemi operativi Microsoft, ma esistono anche alcune implementazioni, seppur incomplete, per sistemi Unix. Alcuni dei progetti più in rilievo sono Mono (implementazione della multi-piattaforma del CLS) e Portable .NET (parte del progetto DotGNU, anch'essa è una implementazione del CLR).

Storico delle versioni del framework:

- .Net Framework 1.0, rilasciato il 13 febbraio 2002. In contemporanea venne rilasciato anche l'ambiente di sviluppo Visual Studio .NET.
- .NET Framework 1.1, rilasciato assieme a Visual Studio .NET 2003 e come aggiornamento a sé stante, venne integrato nativamente in Windows Server 2003.
- .NET Framework 2.0, rilasciato da Microsoft il 27 Ottobre 2005 in contemporanea con l'uscita di Visual Studio 2005.
- .Net Framework 3.0, incluso in Windows Vista con nome in codice WinFX, trova un ruolo importante all'interno dello stesso sistema operativo, dato dal fatto che include una nuova gestione delle API di sistema.
- .Net Framework 3.5 del 19 novembre 2007 e incluso nell'ambiente di sviluppo Visual Studio 2008.
- .Net Framework 4.0 disponibile dal 12 aprile 2010 e incluso in Visual Studio 2010.

Volendo scendere sempre più nel dettaglio è utile avere più chiaro come il .Net Framework si interpone tra SO e linguaggio. Poiché la storia sta dimostrando come questo strumento sta diventando il cuore stesso del sistema e non più un semplice componente supplementare.

Una delle versioni con il più ricco assortimento di novità è la 3.0. Questa versione non comporta cambiamenti all'architettura 2.0, difatti ne è incrementale. Mantiene il CLR, tuttavia apporta diverse migliorie e nuovi componenti come:

- Windows Presentation Foundation (nome in codice Avalon)
- Windows Comunication Foundation (nome in codice Indigo)
- Windows Workglow Foudation

- Windows CardSpace

Windows Presentation Foundation è una libreria di classi del Framework .Net proprietarie di Microsoft, per lo sviluppo dell'interfaccia grafica delle applicazioni in ambienti Windows.
La principale innovazione di WPF è la rimozione di ogni legame con il modello di sviluppo di Windows tradizionale. Tutti i controlli sono stati riscritti e lo stesso meccanismo basato su scambio di messaggi, cuore del modello di programmazione di Windows, è stato abbandonato.
WPF è basato su un sistema di grafica vettoriale che si appoggia alle DirectX per sfruttare l'accelerazione hardware delle moderne schede grafiche. Può essere impiegato per realizzare applicativi eseguibili anche all'interno del browser Microsoft Internet Explorer o altri browser avanzati, purchè sia presente in Framework. Linguaggio base usato in questa libreria è l'XAML (eXtensible Application Markup Language), basato su XML.

Windows Comunication Foundation è un "sottosistema applicativo" proprietario della Microsoft, che offre la struttura API per la creazione di applicazioni distribuite in ambienti Windows. Se da un lato ogni protocollo di rete (es. http, FTP, SMTP, ecc..) ha un suo modello di programmazione, e necessita quindi di una conoscenza specifica da parte degli sviluppatori per poter essere utilizzata, WCF è stato realizzato con l'intento di ricondurre ad un unico modello diverse tecnologie, rendendo più semplice ed uniforme la programmazione in ambiente Windows. È stato sviluppato per Windows Vista, ma è disponibile anche per sistemi W. XP sp 2, W. Server 2003 e Windows 7.

Windows Workflow Foundation include il modello di programmazione, il motore e gli strumenti per la rapida compilazione di applicazioni Windows basate sul flusso di lavoro. È costituito da uno spazio dei nomi, un motore del flusso di lavoro interno al processo e finestre di progettazione per Visual Studio 2005. Windows Workflow Foundation è un framework che consente agli utenti di creare flussi di lavoro umani o di sistema nelle applicazioni scritte per Windows Vista, Windows XP e famiglia Windows Server 2003. Windows Workflow Foundation può essere utilizzato per risolvere semplici scenari, come ad esempio la visualizzazione di controlli di interfaccia utente basati su input dell'utente, oppure scenari complessi più consueti per le grandi imprese, come controlli per l'elaborazione di ordini e dell'inventario.
Per sequenze si intendono una serie di fasi o step distinti di un programma.

Ogni step è modellato in WF come un'attività. Il framework mette a disposizione delle attività un set di istruzioni. Alcune istruzioni personalizzate possono avere funzionalità aggiuntive. Le attività possono essere assemblate usando il Workflow Designer, tool in uso su Visual Studio.

Windows CardSpace è un tool che fornisce alle applicazioni una tecnologia di progettazione, esecuzione ed amministrazione dei diagrammi di flusso.

Capitolo 4

.NET Compact Framework

Denominato anche .NET CF, il compact framework è stato ridisegnato per essere usato in sistemi mobile e su device embedded come possono essere telefonini, palmari, pads, decoder, controller d'azienda, citofoni, ecc.. Questo framework usa le stesse classi e librerie del normale .NET Framework, più alcune librerie create appositamente per i dispositivi mobile e definiti nel .NET Compact Framewok Controls.

Tuttavia le librerie non sono una copia esatta del Framework originare. Difatti queste librerie occupano molto meno spazio.

È possibile creare applicazioni che usino la versione compact, in Visual studio .NET 2003, in VS 2005 ed in VS 2008, con i linguaggi c# o Visual Basic .NET. Per la versione 2010 è uscito solo di recente il supporto per la versione compact ma la compatibilità all'indietro non è stata mantenuta.

Nonostante quanto detto una piccola eccezione la si può fare per applicazioni scritte con Basicpcc4, le quali sono eventualmente compilabili per il .NET CF. Queste ultime di solito risultano applicazioni decisamente efficienti poiché progettati per device mobile speciali con un compilatore JIT ad alte performance.

Il compact framework 3.5 (usato per la creazione dell'applicazione) contiene le librerie del linguaggio comune e le librerie delle classi del framework base. Oltre alla versione 3.5, sono supportati codici di applicazioni per le versioni 2.0 e 1.0. Il .Net Compact Framework mette a disposizione anche alcune nuove caratteristiche, come il Windows Comunication Foundation, LINQ, Sound Player, nuove librerie per gli

strumenti di supporto runtime ed altro ancora. Lo sviluppo dell'interfaccia grafica è basato sulle Windows Form, utilizzabili anche in ambiente Windows per desktop. L'interfaccia utente può essere molto agevolmente creata grazie a visual studio, il quale grazie all'editor grafico permette di creare visualmente una interfaccia grafica in maniera pratica e con poche semplici mosse. Altre caratteristiche importanti sono i data-bindings accessibili anche al compact .Net Framework. Tutto ciò è dovuto principalmente al fatto che tutto il .Net Framework è orientato ad un ambiente desktop, sebbene sia possibile trovare librerie terze in grado di personalizzare molti dei controlli a disposizione.

Per quanto riguarda invece la disponibilità c'è da dire che il .Net Framework è una piattaforma già inclusa in molti dei principali sistemi operativi Microsoft, nati da Windows CE. Esempio lampante sono Microsoft Pocket PC, Smartphone 2003 ed altri ancora. Ancor più interessante è notare come molte di queste applicazioni se portate nel .Net Framework riescano a funzionare tranquillamente.

Una versione del .Net Framework è disponibile anche per la console Xbox 360. Pertanto condividano lo stesso meccanismo di runtime c'è da precisare che per questa versione, solo una sottocategoria delle classi è disponibile. Per creare applicazioni per questa speciale distribuzione è nato un framework specifico, XNA Framework. Questo framework si appoggia a visual studio ed i prodotti creati hanno alcune limitazioni come ad esempio il numero dei thread utilizzabili non può essere maggiore di 256. A differenza della versione precedente del .Net Framework quest'ultimo gode della proprietà di dotare i thread di affinita ai processori, il che consente quindi una gestione più diretta da parte del programmatore dell'uso dei processori e quindi di implementare algoritmi per il bilanciamento del carico di prestazioni. Questo poiché il target hardware ha come caratteristica principale la disponibilità di più processori.

Un altro aspetto importante in tutti i prodotti Microsoft è la retrocompatibilità. Pertanto fino ad ora le versioni del .Net CF godano di retrocompatibilità, questa però viene ostacolata dal fatto che nonostante ci sia compatibilità del .Net CF con molti prodotti nati da Windows CE, la portabilità viene compromessa spesso dal cambiamento di cpu. Al momento è comunque possibile usare il .Net CF su piattaforme dotate di cpu MIPS, x86, ARM ed Hitachi SuperH.

Capitolo 5

C#

Dopo aver approfondito la nostra conoscenza del sistema operativo e degli strumenti che questo ci mette a disposizione c'è da dare qualche dettaglio in più anche al linguaggio usato.

Perché usare c#? Questa è la domanda con la quale si potrebbe partire.

Sapendo qual era l'obbiettivo forse una combinazione di c++ e java avrebbero raggiunto lo scopo con minori complicazioni. Pertanto la parte a più basso livello sarebbe stata implementata in c++ che è anche il linguaggio con la quale PJSIP è stato scritto, e la parte ad alto livello la si sarebbe potuta scrivere in java, il quale gode di molta portabilità sui diversi sistemi operativi attualmente in uso sui nostri dispositivi. Nonostante ciò sapendo come l'applicazione dovesse funzionare su Windows mobile e sapendo anche come era la struttura del sistema operativo abbiamo preferito usare anche il linguaggio casa Microsoft.

Detto questo quando si parla di c# si parla di un linguaggio ad oggetti, approvato poi come standard ECMA, con ereditarietà singola di classi e ereditarietà multipla di interfacce. La sintassi prende spunto dal c++ , dal java, dal delphi e dal Visual Basic, rimanendo meno ricco di elementi decorativi rispetto al java, e con meno simbolismo di c++.

In un certo senso c# è il linguaggio che meglio descrive le linee guida sulle quali ogni programma .NET gira.
Questo linguaggio è stato specificatamente creato per la programmazione nel Framework .NET, partendo da concetti sia del c, c++ che del java. Definire se c# sia un linguaggio compilato o interpretato è piuttosto

complicato, difatti, data la sua stretta integrazione con il Framework i codici sorgenti in c# sono normalmente compilati secondo i criteri JIT: la trasformazione in codice macchina avviene su richiesta solo al caricamento e all'esecuzione del programma. In prima istanza il codice viene convertito in codice intermedio detto CIL e solo all'esecuzione del programma in CLR specifico per il sistema operativo utilizzato converte il CIL in codice macchina mano a mano che questo viene eseguito. C'è da dire che esiste anche una diverso tipo di compilazione possibile, chiamata compilazione Ngen che porta tutto il codice CIL in codice macchina in una sola volta.

Due dei tanti vantaggi che risaltano e che sono stati i motivi per la quale è stato scelto questo linguaggio è che in c# è stato implementato un sistema di pulizia della memoria automatizzato efficiente, detto Garbage Collector, e l'altro vantaggio è che in c# è possibile importare funzioni e metodi di librerie scritte in altri linguaggi come il c++.

Capitolo 6

L'evoluzione del progetto

Ora che tutte le più importanti tecnologie e gli strumenti utili sono stati affrontati e spiegati rimane solo da spiegare il cammino e le tappe del nostro percorso.

Prima di tutto ho fatto una ricerca su internet per cercare se ci fossero stati progetti open source che adottassero quantomeno il linguaggio c# e che implementassero un generico client VoIP. Il programma dopo qualche tempo lo sono riuscito a trovare, Sipek, un programmino open source, scarno ma che usava al suo interno delle librerie PJSIP.

Trovata questa applicazione la prima cosa logica che il mio relatore ed io abbiamo appurato è che il traffico generato da questa applicazione passava solo per una sola interfaccia di rete. Cioè se per l'accesso ad internet veniva usata la wireless per tutto il tempo della connessione al server VoIP il programma usava solo quel canale. Quindi abbiamo deciso di partire creando un piccolo proxy che potesse accettare il traffico generato dal programma, interfacciarsi al server di destinazione e utilizzando una o più connessioni, fare da tramite per la trasmissione dei pacchetti.

In c# la creazione dei socket e l'uso sono una cosa molto facile.

Per avere una visione un po' più chiara ne faccio un esempio pratico.

```
private static Socket ConnectSocket(string server, int port)
    {
        Socket s = null;
        IPHostEntry hostEntry = null;
        hostEntry = Dns.GetHostEntry(server);
        foreach(IPAddress address in hostEntry.AddressList)
```

```
        {
            IPEndPoint ipe = new IPEndPoint(address, port);
            Socket tempSocket =
                 new Socket(ipe.AddressFamily, SocketType.Stream,
                 ProtocolType.Tcp);
            tempSocket.Connect(ipe);
            if(tempSocket.Connected)
            {
                s = tempSocket;
                break;
            }
            else
            {
                continue;
            }
        }
        return s;
    }
```

L'unico punto oscuro a quel tempo era: ma di che connessioni posso disporre?

La cosa più ovvia era prima individuare l'hardware per le possibili connessioni, che in termini pratici voleva dire trovare un modo in c# di identificare all'interno del sistema operativo i vari dispositivi come la scheda wireless.

Questo punto che è facile e banale comprendere è stato il crucio maggiore per tutto il tempo. Il motivo è che la documentazione per l'uso ed identificazioni è stato creato dalla Microsoft per il proprio Framework completo e non sono state create adeguate librerie per la versione compact.

La cosa più snervante era anche trovare diverse implementazioni di codice che arrivassero ad identificare i vari dispositivi, e questo era dipeso dal fatto che ogni versione del framework cambiava il metodo di accesso, ed arricchiva gli strumenti a disposizione cambiandone però ogni volta la sintassi.

Ogni volta che provavo a usare un codice c'era qualche problema, del tipo, librerie mancanti, inclusioni nel progetto inesistenti.

Il primo tentativo provato è stato il seguente:

```
NetworkInterface[] nics =
            NetworkInterface.GetAllNetworkInterfaces();
PhysicalAddress address = nics[0].GetPhysicalAddress();
 byte[] bytes = address.GetAddressBytes();
 for (int i = 0; i < bytes.Length; i++)
 {
    Console.Write("{0}", bytes[i].ToString("X2"));
```

```
            if (i != bytes.Length - 1)
            {
                Console.Write("-");
            }
        }
```

In questo codice si usa la classe NewtorkInterface, la quale al suo interno include un metodo GetAllNetworkInterface() il quale provvede a restituire un elenco di tutte le interfacce di rete. Successivamente per ogni elemento trovato si provvede a identificare il Mac address e a stamparlo.

Questa soluzione presentava il minor numero di problemi nella versione Windows mobile 7.0 poiché in quest'ultima usando il .Net framework 4.0 la classe NetworkInterface viene implementata ma offre solo una funzione che ci avvisa se c'è connessione o meno, ma oscura completamente i dati della connessione, la quale viene gestita dal sistema operativo.

Visto che non ha funzionato allora ho cambiato strategia. Andando a cercare nella rete ho trovato una libreria chiamata OpenNETCF con il cui uso in teoria si potevano rilevare i mac address dei dispositivi sulla piattaforma. Per l'uso della libreria bastava importare nel progetto la dll e poi con il comando di importo della specifica funzione si può usare nel codice le funzionalità della libreria.

Codice d'esempio per avere più chiare le idee:

```
        [DllImport ("iphlpapi.dll", SetLastError=true)]
        public static extern int GetAdaptersInfo( byte[] ip, ref int
size );
```

In Windows CE, usando alcune librerie base, era possibile avere controllo diretto sull'hardware e risolvere questi piccoli problemi con una facilità disarmante, ma cercare di forzare il sistema operativo è stato inutile.

Facendo altre ricerche sono riuscito a trovare un modo funzionante per ottenere l'indirizzo IP del mio dispositivo, ma anche avendo trovato questo metodo non ne avrei tratto vantaggio. Per questo metodo ho usato le librerie che lavorano sul dns e da li ho stampato le informazioni.

```
            // Get host name
            String strHostName = Dns.GetHostName();
            Console.WriteLine("Host Name: " + strHostName);
            // Find host by name
            IPHostEntry iphostentry =
                                Dns.GetHostByName(strHostName);
            // Enumerate IP addresses
```

```
int nIP = 0;
foreach (IPAddress ipaddress in iphostentry.AddressList)
{
    Console.WriteLine("IP #" + ++nIP + ": " +
                      ipaddress.ToString());
}
```

Anche se questo tentativo è andato a vuoto ho ritrovato fiducia e ho provato a cercare altri metodi. E così arrivai ad un altro metodo che alla carta sembrava molto interessante.

Questa volta il concetto era quello di fare una query sulla network adapter configuration e prendere in esame solo gli elementi il cui parametro IPEnable era settato a true.

Ovviamente alla fine non ha portato a nulla perché il compact framework non fornendo il supporto del netframework completo non aveva nessuna delle componenti richieste.

```
ObjectQuery oQuery = new ObjectQuery("Select MacAddress,IPAddress
from Win32_NetworkAdapterConfiguration where IPEnabled=TRUE");
RemoteConn = new ManagementScope("\\\\" + RemoteHost +
             "\\root\\cimv2",options);
query1 = new ManagementObjectSearche(RemoteConn,oQuery);
queryCollection1 = query1.Get();
```

Visto questi fallimenti allora ho cambiato strategia. Ho pensato, magari risolvo il problema se imparo prima a gestire una connessione wireless visto che alla fine i dispositivi dove il programma andrà ad eseguirsi è probabile abbiano una wireless.

Ho visto che come al solito c'erano diversi espedienti.

Il primo si basava sull'importazione di un paio di metodi da delle librerie esistenti ma che non erano direttamente usate nel .Net Framework e che quindi se avessero funzionato, le si sarebbe dovute importare a mano. Dico se avessero funzionato poiché questa tecnica come di routine, è valida solo per il framework completo e non per il compact edition.

Esempio:

```
[DllImport("wzcsapi.dll")]
        public static extern uint WZCEnumInterfaces(string
            pSrvAddr, ref INTFS_KEY_TABLE pIntfs);
        [DllImport("wzcsapi.dll")]
        public static extern uint WZCQueryInterface(string
          pSrvAddr, uint dwInFlags, ;ref INTF_ENTRY pIntf,
          ref uint pdwOutFlags);
```

Questi sono i 2 metodi importati, e di seguito un piccolo main per il test.

```
static void Main(string[] args)
{
    INTFS_KEY_TABLE pIntfs =&amp;nbsp;new INTFS_KEY_TABLE();
    INTF_ENTRY Intf = new&amp;nbsp;INTF_ENTRY();
    uint uiRet;
    uint uiInFlags = 0xFFFFFFFF;
    uint uiOutFlags = 0;
    uiRet = WZCEnumInterfaces(null, ref pIntfs);
    Intf.wszGuid = pIntfs.pIntfs.wszGuid;
    uiRet = WZCQueryInterface(null, uiInFlags, ref Intf,
                    ref uiOutFlags);
}
```

Molti dei tentativi successivi non hanno portato giovamenti, ne nuove idee per agire, ma sono riuscito a imbattermi in una serie di discussioni interessanti, dove molti programmatori sostengono che, pertanto che il codice sia “chiuso”, ci sono modi ancora poco chiari per forzare il sistema usando le API Win32 che risiedono nel IP Helper, oppure cercando di usare alcuni strumenti del WMI (Windows Management Instrumentation). Il concetto di fondo sarebbe che è noto che alcune terze parti riescono in qualche modo a ottenere ciò che con le api di base e gli strumenti Microsoft a disposizione non è possibile. Quindi da qualche parte devono pur aver preso qualche dato. Il WMI ha un sacco di classi wrapper (cioè classi che inglobano altre classi al loro interno e alla quale aggiungono funzionalità) e anche se non ho ancora visto dove, sicuramente in una di queste c’è anche la classe con le informazioni di sistema che servono. Il vero problema è riuscire a capire se una volta identificate le classi il lavoro si può in qualche modo portare sul mobile.

Il client VoIP che si è trovato non funzionava se non su distribuzioni di .Net CF maggiori alla 3.0, ed era quindi improponibile usare sistemi operativi precedenti al Windows mobile 6.0.

Vedendo il codice usato per Sipek, ho visto che le librerie PJSIP fossero implementate in c++, così ho fatto anche ricerche trasversali per vedere se era possibile aggirare l'ostacolo cambiando linguaggio.

Le ricerche supportate dal professore hanno portato alla luce il progetto nokia QT. La nokia ha sviluppato un framework c++ based per la creazione ed implementazione di apps per i propri cellulari.

Dopo qualche mese di lavoro dall'inizio del progetto ho potuto solo notare che forse una soluzione al problema potrebbe essere provare il framework nokia. Per motivi legati al tempo che si sarebbe dovuto adoperare allo scopo di settare la piattaforma di lavoro per l'utilizzo, imparare ad usare i nuovi strumenti, e tenendo presente di rinunciare a usare il linguaggio che avevo scelto, non ho optato per testare. Tuttavia anche se è ancora tutto da verificare e da testare, alcuni dubbi potrebbero già essere di spunto per chi volesse addentrarsi nel contesto tramite questa via. Uno dei tanti è capire se davvero l'uso di altre librerie possano incidere in una problematica le cui origini risiedono nel sistema operativo stesso e non negli strumenti. Alla fine è Windows stesso che con il .Net Framework non lascia modo all'utente di osservare e usare l'hardware, quindi sarebbe utile scoprire se e come le nokia QT riescano quantomeno a rilevare i dispositivi.

Sta di fatto che al momento la sezione hardware su Windows mobile è blindata e questo non si può cambiare.

Una cosa da sottolineare è che Windows mobile 7.0 sia uscito oramai già da qualche mese ma la documentazione per programmatori ha fatto la sua prima comparsa solo a meta dicembre e quindi il materiale utile per un corretto uso dei nuovi strumenti non è ancora interamente disponibile. Nonostante ciò anche nella versione 7.0 il controllo hardware è ancora impedito.

6.1 Cronologia

Per riassumere e fare un quadro generale riporto le tappe principali di questo lungo processo di ricerca:

- Ricercare l'indirizzo MAC
    - Usare librerie OpenNETCF's SDF
      La libreria è includibile ma alcuni metodi interni non ci sono per il .Net CF
    - Usare la classe NetworkInterface
      Fallito poiché non supportata dal .Net CF
    - Usare la libreria iphlpapi.dll
      L'import avveniva, bisognava comunque portare a mano la libreria nel dispositivo ma nonostante ciò il .Net CF non aveva tutti gli elementi necessari
    - Fare una query sul Network Adapter Configuration
      Il codice per il rilevamento degli IP funziona ma non restituisce altri strumenti per la identificazione del tipo di connessione ed altri dati inerenti
    - Adottare una libreria proprietaria che fungesse da canale intermedio per la manipolazione delle periferiche hardware (questa strada non è stata approfondita poiché il materiale era a pagamento e non si è riuscito a studiarlo)
- Cercare metodi per la configurazione della scheda wireless
    - Usare le funzionalità della libreria wzcsapi.dll
      Questa tecnica funziona solo su apparati desktop e non nel .Net CF
    - Usare funzioni del IP Helper
      Ip Helper ha supporti validi solo per il Framework Completo.
    - Studiare il WMI sia in ambiente desktop che mobile e cercare nelle classi interne alle classi wrapper, al fine di trovare le informazioni di connessione.
- Studiare la classe NetWorkInterface per la versione 7.0 di Windows Phone

    - Studio di NetworkInterface.GetIsNetworkAvailable() e ricerca di altri metodi o vie alternative*

- Ricerca di componenti e vie esterne al .Net Framework
    - Nokia QT
- Dirottamento del lavoro sull'interfaccia grafica

(*): Nell'ultimo capitolo viene affrontato meglio l'argomento riguardante alle soluzioni in Windows Phone 7.

6.2 L’interfaccia

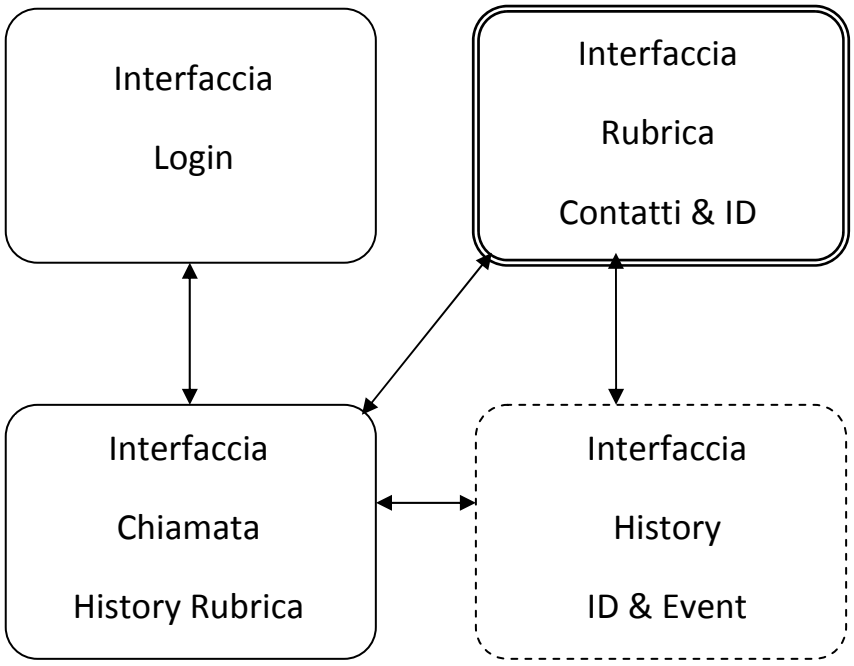
Dato che questo punto del progetto non è stato possibile superarlo, insieme al professore abbiamo deciso di cambiare direzione e mi sono concentrato sull’aspetto interazione programma utente.

Sipek era scarno e permetteva solo il login ad un server e poi chiamare un contatto.

La prima cosa da fare è stato creare una rubrica per la gestione dei contatti.

Anche se ero dubbioso e pensavo di trovare chissà quali difficoltà, l’implementazione non mi ha dato problemi e posso dire che almeno per quanto riguarda questa parte più visual il linguaggio e i tool del Visual Studio rendono piacevole e rilassante la programmazione, riuscendo a regalarmi qualche soddisfazione nell’aver scelto questo linguaggio che fino a poco tempo prima stava perdendo ogni ragione valida per essere scelto.

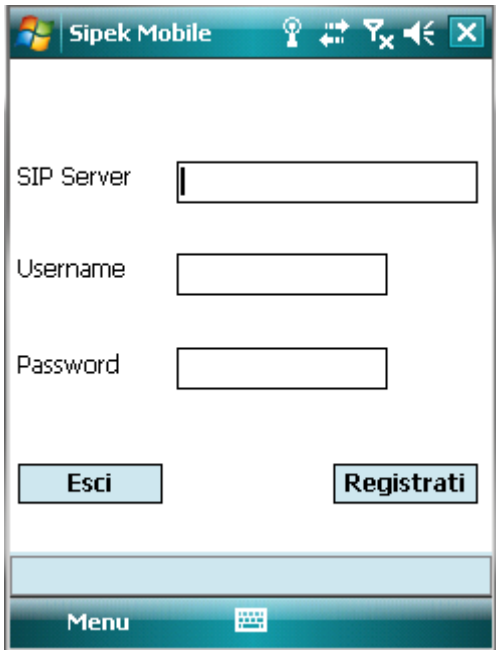
Per fare luce su quello che era l’obbiettivo ripropongo uno schema logico delle interfacce e delle loro connessioni.



Le due interfacce, Login e Chiamata erano la base di partenza.

Le altre sono invece state create da zero.

A programma avviato ci si ritrova la prima interfaccia



Il client richiede il server sip dove connettersi, uno username e una password.

Per le prove effettuate sono stati creati due account presso il server ekiga.net.
Per come è strutturato il codice è comunque possibile fare test e avere esito positivo su altri server SIP online. Basta registrare l'account e provare.

Dopo la registrazione al server che avviene per mezzo della libreria pjsip che inoltra una richiesta da [nome utente]@[server], l'applicazione setta i parametri base per una sessione e rilascia a disposizione dell'utente l'interfaccia telefono.

Da qui abbiamo la possibilità di:

- Comporre manualmente il numero (si precisa che il numero non è necessariamente una sequenza di soli numeri ma è previsto il supporto alfanumerico, poiché al numero corrisponde un login)
- Accedere all'agenda del telefono
- Accedere quando presente alla history delle chiamate (non visualizzata in immagine poiché non ancora presente)
- Rispondere a una chiamata in arrivo
- Chiamare il contatto inserito nel campo del numero
- Scollegarsi dal server

Procedendo con la navigazione visitiamo l'agenda.

L'agenda non presenta nulla di speciale. Restituisce una lista di contatti salvati in memoria locale e fornisce su questi contatti determinate operazioni:

- Crea Contatto, permette di creare un contatto in memoria
- Seleziona, riporta nella form del telefono il numero del contatto selezionato
- Modifica, permette di modificare i dati interni di ogni contatto
- Dettagli, restituisce tutte le informazioni inerenti ad un contatto
- Elimina, rimuove dalla memoria il contatto selezionato
- Annulla, ritorna alla schermata del telefono senza alterazioni

Visto che le funzioni non sono poche punto per punto affronto ogni singola funzione qualora richieda spiegazioni.

Memorizzazione

Tutti i contatti del telefono risiedono in un file interno che qualora manchi viene creato automaticamente pulito, all'avvio dell'applicazione.

Ogni contatto presenta alcune regole per una corretta memorizzazione e visualizzazione:

- Nome: deve essere sempre presente ed insieme al numero forniscono i due dei quattro campi necessari per la memorizzazione inseribili dall'utente
- Numero: deve essere sempre presente e come detto per il nome è uno dei quattro campi obbligatori
- Gruppo: campo opzionale che permette all'utente di creare dei gruppi simbolici per i contatti, i quali possono essere usati per un diverso ordinamento in fase di visualizzazione
- ID Contatto: Campo autonomamente deciso dal programma e che permette di identificare in maniera univoca il contatto. Questo server poiché con la previsione di creazione di più strumenti che operano sui dati come la history, risulta utile avere un dato più piccolo ma sufficiente per le distinzioni che una più completa ma ingombrante struttura di dati
- ID Memoria: questo campo è la chiave univoca per la corretta memorizzazione dei dati all'interno del file database.

L'algoritmo di memorizzazione e lettura è basato su queste regole:

- Ogni campo è memorizzato su linea singola, quindi non sono previste memorizzazione di dati su più linee, e nel caso l'ultimo dato acquisito sostituisce il precedente
- Per la memorizzazione esiste una stringa prefisso composta da [nome campo][ID memoria].
- Successivo al prefisso e separato dal carattere di separazione ":" c'è lo spazio riservato al campo e terminato dal termine di fine linea. (Lavorando su Windows mobile, soggetti all'uso di un solo linguaggio e avendo a disposizione file System supportati da Microsoft non vi sono problemi di conversione del carattere di terminazione come viceversa può avvenire durante il trasferimento di un file di testo da un sistema linux a uno Windows)
- Per ogni riga presente nel file database il programma analizza il contenuto di quelle sole righe che rispettano le prime tre regole e nel caso procede alla compilazione di una lista temporanea interna, aggiungendo il dato acquisito

- Al termine dell'analisi del database si procede con l'analisi della lista.
- Durante la seconda fase di analisi vengono scartati tutti gli elementi che non presentano i campi necessari
- Al termine di questa il sistema in caso sia necessario procede a ridimensionare i dati qualora necessario:
    - Se il numero dei contatti validi è differente dal numero di dati parziali raccolti durante la lettura, ne riscrive il contenuto in memoria affinché la successiva raccolta non preveda dati inutili
    - Se il numero ID memoria dei contatti è di ordine superiore con un limite attuale del doppio, al numero effettivo di contatti della lista, avviene una riassegnazione degli ID memoria. Questa regola non è volutamente applicata anche agli ID contatto, poiché successive implementazioni di strumenti che usino il campo ID contatto per il riconoscimento, darebbero lavoro extra allo strumento base, che in tal caso dovrebbe andare a sostituire ogni ID contatto cambiato in ogni parte della memoria ove salvato.

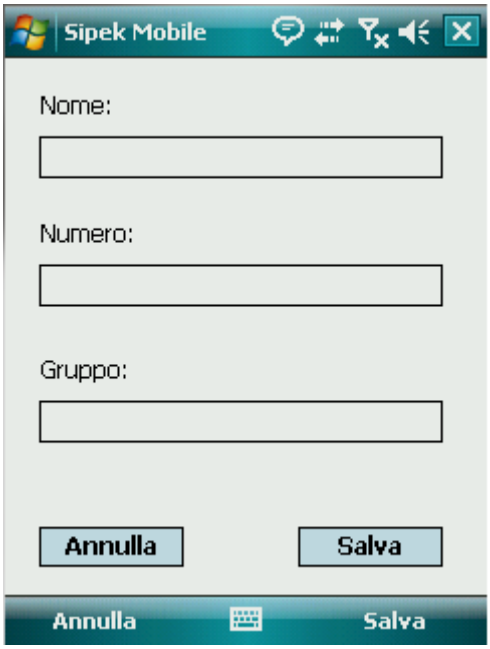
Queste regole forniscono lo stretto necessario per una memorizzazione sufficientemente efficiente per il dispositivo e senza appesantire con strumenti magari più eleganti (come xml) ma che in molti casi richiedono l'uso di altri strumenti.

I dati che risiedono sul dispositivo sono attualmente in chiaro ma questo vincolo può essere tranquillamente modificato premettendo una cifratura e decifratura prima e dopo gli accessi al file sorgente. Al momento un dispositivo non è stato ritenuto avere questo genere di esigenze ma non per questo non è possibile integrarlo.

Visto il meccanismo e le regole di memorizzazione passiamo a visionare gli strumenti tra quelli precedentemente citati che ne fanno uso.

È da precisare che una volta all'interno dell'agenda il programma ha già analizzato e istanziato i dati in memoria.

- Crea Contatto / Modifica:



  La form per la creazione e modifica è la medesima, cambiano solo i dati di partenza. Se si annulla l'operazione si chiude semplicemente la form, nel caso di salvataggio viene aggiornato il database con la modifica o l'aggiunta.

- Elimina:
  Questa funzione ricerca il contatto nel database tramite l'ID memoria e lo elimina riaggiornando il database

- Visualizza:

Accede al database e visualizza i dati.

Per la history, qual'ora ci sia è accedibile e mette a disposizione per ogni chiamata fatta o ricevuta il totale minuti secondi di conversazione e la possibilita di:

- Seleziona: per impostare il numero nella chiamata
- Salva: apre la finestra di dialogo per aggiungere il contatto
- Chiudi: chiude l'history

La history viene salvata in locale su un file in chiaro e le regole per la memorizzazione sono:

- In ogni riga vengono salvati dati di una singola conversazione.
- Ogni riga ha un prefisso HISN[numero_utente]:gg-mm-aa-ss-mm-*o oppure HISS[ID Contatto]:gg-mm-aa-ss-mm-*o dove le ore hanno un numero indefinito di caratteri numerici associati. Il prefisso muta se il contatto è presente o no in memoria.
- All'avvio del programma avviene un controllo della history
- Ad ogni chiamata fatta e ricevuta la history viene aggiornata
- La richiesta di salvataggio di un numero dalla history avvia un aggiornamento della history per trasformare il contatto in elenco da numero a ID Contatto. (Al contrario la cancellazione dell'elemento

dalla rubrica non incide sulla history, tuttavia quando viene aggiornata una history con ID Contatto inesistente viene scartato l’elemento e raiggiornata la history)

Capitolo 7

Per il futuro

Il lavoro è appena cominciato e i traguardi sono stati già segnalati in mappa.

Ora è solo il lavoro di ricerca che deve fare il suo corso. Al momento molti ostacoli non sono stati ancora superati, ma è possibile già lavorare su parte di essi anche da adesso.

Nella parte di interfaccia con l'utente la parte della history potrebbe essere migliorata e adattata includendo l'uso dei gruppi.

Successivamente a cambiamenti e miglioramenti delle utility da parte della Microsoft c'è da espandere meglio e più approfonditamente la ricerca di metodi per la manipolazione più profonda dell'hardware del dispositivo.

Uno dei possibili spunti alternativi a questo problema potrebbe risiedere in uno studio più approfondito ed accurato del framework suite della Nokia, QT, che andrebbe testato.

Altri tentativi potrebbero essere fatti verso le Open .Net CF Api, le quali anche se in maniera ancora insufficiente sembrano aver dato maggiore spunto a ciò che un programmatore può ricercare nelle utility di un framework per dispositivi portabili.

Per le librerie pjsip bisognerebbe sforzarsi affinché adattare una versione più recente delle librerie al codice attuale, poiché quelle utilizzate fino ad ora non hanno incluso molte features al codice c#, per esempio non ho trovato modo di poter incanalare nella sessione, in maniera semplice, un messaggio di testo.

Vale la pena spendere anche due ulteriori parole per quanto riguarda gli strumenti di informazione e di help da parte della Microsoft.

Pertanto online siano messe a disposizione un set nutrito di documentazione riguardante le varie componenti e strumenti, ad un lettore ancora inesperto, la comprensione della documentazione, non è quasi mai, semplice ed intuitiva. C'è poi da ricordare che buona parte del materiale di documentazione non fa riferimento diretto alla versione Compact del Framework generando spesso disagi al programmatore che vuole avere una idea più chiara e giusta degli strumenti e della piattaforma.

Capitolo 8

Un futuro con Windows Phone 7

Mentre si “attendono” risposte effettive dagli ambienti di sviluppo e dai produttori dei sistemi operativi, è saggio guardare avanti e pensare già a quali sarebbero i vantaggi ottenibili dal portare il progetto sul nuovo sistema operativo Windows Phone 7.0.

Con la nascita di Windows Phone 7, l’azienda produttrice, fa un passo in avanti verso l’intuitività dell’utente e abbandonando sempre di più gli schemi di visualizzazione classici.

Presentato al Mobile World Congress il 15 febbraio 2010, questo prodotto si rivolge al mercato consumer invece che al mercato enterprise, al contrario dei propri predecessori, abbandonando alcune caratteristiche presenti nei Windows Mobile.

È radicalmente differente dalle precedenti versioni Mobile, con le quali è incompatibile, ma supporta il multitouch, gli schermi capacitivi, ha una interfaccia grafica simile a Zune HD e riunisce in una unica piattaforma i contenuti sia di Zune che di Xbox Live.

Una piccola curiosità è che il nome di battesimo fu “Windows Phone 7 Series”, ma per una esigenza di mercato e rendere il logo più snello fu eliminato “Series”.

A luglio 2010 è stata rilasciata la beta di Windows Phone Developer Tool, un pacchetto software gratuito per sviluppare applicazioni per Windows Phone 9, contentente:

- Visual Studio 2010 Express per Windows Phone

- Windows Phone Emulator
- Silverlight per Windows Phone
- Microsoft Expression Blend per Windows Phone
- XNA Game Studio 4.0

Guardando un po' nel passato troviamo che il progetto, con lo scopo di creare una nuova generazione di Windows Mobile, è iniziato nel 2004 sotto il nome in codice "Photon", ma poiché il progetto andava avanti lentamente fu abolito e lasciato perire.

Nel 2008, la Microsoft, ha riorganizzato i gruppi di development e ha fatto partire un nuovo progetto per la creazione del nuovo Windows Phone, la cui data di uscita sarebbe stata prevista per il 2009, ma per diversi problemi e una serie di ritardi il progetto fu posticipato e fecero uscire prima la versione Windows Mobile 6.5 in attesa del nuovo Sistema Operativo.

Il lavoro si svolse in fretta, e questo ha portato le sue conseguenze visibili. Il prodotto finito alla fine non era retrocompatibile. Tant'è che Terry Myerson, ingegnere di sviluppo dei Windows Phone ha dichiarato, "Con lo sviluppo degli schermi capacitivi, senza necessità di pennino, e l'adozione di particolari scelte di hardware per il Windows Phone 7, abbiamo dovuto interrompere la compatibilità delle applicazioni del Windows Mobile 6.5.

Figura 5 Linea del tempo dei principali aggiornamenti

Il 4 febbraio 2011 viene rilasciato il primo grande aggiornamento del sistema operatvio, nome in codice NoDo. Dopo il rilascio la casa produttrice rassicurò gli utenti che nonostante le sostanziali modifiche, i codici esistenti non si sarebbero dovuti ricompilare, anche se le modifiche fossero principalmente rivolte alle nuove strumentazioni per lo sviluppo.

Alla fine del settembre 2011 la Microsoft ha rilasciato l'aggiornamento annuale definitivo di Windows Phone 7.5 (denominato Mango), il quale si presenta il più imponente aggiornamento dal lancio del sistema operativo. Con questo aggiornamento la casa produttrice annuncio la presenza di ben oltre 500 novità.

Anche se per ora non si conosce la data di pubblicazione è già in progetto l'ultimo aggiornamento del sistema operativo, nome in codice Tango.

Ora che abbiamo un po' visto le informazioni generali che giustificano il motivo per la quale il progetto di questa tesi non è nato sotto Windows Phone ma su Windows Mobile 6.5, cerchiamo in questo nuovo SO quali potrebbero essere i vantaggi che potrebbero ottenersi.

Forte di una nuovissima ed innovativa interfaccia utente dal nome in codice Metro, stravolge completamente la classica impostazione dei menu di navigazione usata fino alla versione precedente. La schermata principale, chiamata “Schermata Start”, è composta da “Live Tiles”. Questi collegamenti ad applicazioni, funzione o oggetti individuali (come contatti, pagine internet, applicazioni o altri), possono essere maneggiati dall’utente che può aggiungerli, spostarli o eliminarli dal menu. I Tiles sono dinamici e si aggiornano in tempo reale. Per esempio un Tiles associato agli sms può mostrare quanti nuovi messaggi non letti ci sono, oppure uno associato al canale meteo può visualizzare costantemente le mutazioni.

Molte delle caratteristiche di Windows Phone 7 sono organizzate in “hub”, che combinano contenuti online e locali, come per esempio l’hub delle foto può associare i contenuti offline ripresi dalla fotocamera e legarli ai photobook dei social network.

Una delle principali novità è la presenza del supporto multi-touch. L’interfaccia è stata resa nera per un accorgimento per la quale la batteria dura di più sui monitor OLED, visto che in questo tipo i punti neri non emettono luce.

Andando più nel dettaglio, tutte queste innovazioni a livello di interfaccia, che differiscono dalla classica struttura basata su form, vengono implementante grazie all’uso combinato di più elementi:

- Xaml, linguaggio base per la scrittura della struttura grafica
- Silverlight e C#, per la generazione dei metodi funzioni con la quale scrivere i programmi
- XNA, framework basato su c# che imposta il nuovo sistema di refresh della parte grafica ottimizzando l’uso dei dispositivi hardware grafici

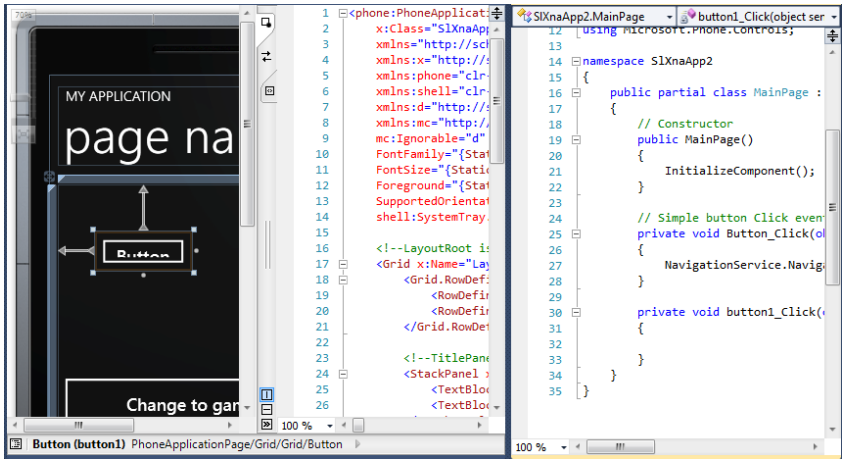


Figura 6 Esempio sull'uso combinato di Xaml e C#

Passando da una implementazione grafica fortemente identificata negli standard del sistema operativo per ambiente Desktop, le operazione di render, soprattutto quelle a real time, risultano molto onerose e complicate da ricreare, poiché lo scopo con la quale era nata questa UI non era la dinamicità. In questo invece lo spunto preso dal campo Html ha permesso di avere un codice i cui risultati in termini di prestazioni fosse più fluido.

Separando la strutturazione grafica dalle funzionalità si è creato un maggiore ordine a livello cognitivo in fase di programmazione.

Il mantenimento dello stesso linguaggio non vuol dire compatibilità netta garantita, difatti il concetto di fluidità e reattività dell'applicazione non risiedono solo nel linguaggio, ma anche nella meccanica interna di come i sottoprocessi, gli handler, e tutti gli altri strumenti vengono usati e chiamati.

XNA basato su Directx 9 e .NET Framework, permette di poter lavorare sia ad alto livello che a basso livello a discrezione dello sviluppatore.

Questo concetto differisce di molto dalla classica impostazione dei programmi in .NET, dove la programmazione viene spronata a rimanere ad alto livello e quindi a vantaggio della compatibilità ma non della performance.

Partendo da questa rivoluzione, e contando sul fatto che la linea di pensiero XNA venga mantenuta anche in Windows Phone 7, è possibile pensare a riapplicare il progetto della tesi su questo nuovo framework.

La parte pjsip deve essere ristrutturata da capo visto che per integrarla si faceva uso di una parte di .NET che verrebbe a mutare e quindi non più utilizzabile.

D'altro canto l'accesso alle risorse hardware sarebbe molto più facilitato anche grazie all'uso delle Direct X.

Questa combinazione XNA, DirectX ha avuto modo di far notare i propri vantaggi in una console sempre di casa Microsoft, Xbox, per la quale è possibile anche programmare applicazioni fruibili sia su computer desktop e sia su console.

Ringraziamenti

Per la pazienza e la disponibilità portata durante tutte le fasi del progetto e della relazione del documento ringrazio immensamente il mio relatore Vittorio Ghini.

Per il sostegno morale e per un aiuto nella ricerca degli errori nella stesura del documento vorrei ringraziare in particolar modo Erica Bianchi e Alan Torrisi, senza i quali difficilmente avrei potuto arrivare alla fine del percorso.