Università degli Studi di Bologna

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica
Reti di Calcolatori

Metodologie e strumenti di anticontraffazione in ambienti Android

Tesi di laurea di:

Sebastiano De Angelis

Relatore:

Chiar. mo Prof. Ing. **Antonio Corradi**

Correlatori:

Dr. Ing. **Giacomo Cancelli**

Dr. Ing. **Luca Capobianco**

Anno Accademico 2010-2011
Sessione III

Indice

Introduzione

Il fenomeno della globalizzazione ovvero della crescita progressiva e degli scambi a livello mondiale ha e continua ad avere come effetto principale la decisa convergenza economica e culturale tra i Paesi del mondo. Un mercato globale come quello odierno prevede potenzialità e rischi di portata assai maggiore di quella delle epoche passate, quando le piazze erano locali o nazionali, oltre a recare un naturale rafforzarsi della concorrenza in tutti i settori merceologici. Come è noto, la concorrenza è un bene per il consumatore e la spinta a migliorare per il produttore. Per tale motivo, in ciascun Paese, a sua tutela nascono complessi di norme giuridiche ai più note col termine *antitrust*.

Al contrario, sia il consumatore che il produttore vengono profondamente danneggiati quando si verifica una concorrenza sleale derivante dall'uso di tecniche e mezzi illeciti per ottenere un vantaggio sui competitori o per arrecare loro un danno: ne sono esempi la diffusione di informazioni che gettino discredito sulle attività dei concorrenti o l'utilizzo di nomi o marchi che ricordino quelli di altre aziende, per spingersi fino alla produzione di falsi.

Scenario di questa tesi è l'anticontraffazione e il suo contributo consiste nell'ingegnerizzazione e in parte dello sviluppo di un applicativo software dell'azienda ViDiTrust. Il programma, realizzato per dispositivi mobili evoluti, permette di acquisire informazioni attraverso un particolare timbro apposto sui beni di consumo di pronunciarsi sulla originalità o meno del prodotto.

Al fine di inquadrare meglio il lavoro di tesi, nel primo capitolo si introdurranno i principali argomenti di riferimento e le relative tecnologie.

Descritta l'architettura SOA, la sua incarnazione in Web Services e i principali protocolli d'uso, s'introdurranno i concetti base inerenti i sistemi di localizzazione quali il GPS, la risoluzione via network cellulare o Wi-Fi.

Nel secondo capitolo si presenterà una panoramica su alcuni sistemi di anticontraffazione, in particolar modo quelli relativi al settore grafico e delle immagini stampate. Successivamente verrà descritta la tecnologia di autenticazione passiva escogitata dall'azienda ViDiTrust e il suo fondamento, il concetto di "impronta digitale" *vIDfy*.

Nel capitolo terzo l'attenzione è posta sull'applicazione ViDiApp, sviluppata per dispositivi mobili su piattaforma Android. Fornite le informazioni relative ai requisiti minimi da soddisfare per poter usare il software, si sono indicate le aree d'intervento oggetto di tesi, nonché gli scopi e le finalità relative alle modifiche apportate.

Il capitolo quarto costituisce il cuore del lavoro svolto e presenta una descrizione tecnica delle funzionalità modificate o aggiunte all'applicativo ViDiApp. Per illustrare l'entità e le modalità di intervento, l'esposizione è stata arricchita con numerosi riferimenti al codice Java, sviluppandosi lungo un percorso logico fra i moduli e i blocchi funzionali.

Nel quinto e ultimo capitolo sono presentati un insieme di test temporali relativi ai gestori di comunicazioni HTTP e SOAP al fine di verificarne e confrontarne le prestazioni. Oltre alle varie piattaforme su cui sono stati eseguiti i test, al lettore vengono suggerite anche le linee guida seguite per realizzare le prove e i software necessari alla raccolta dei dati temporali. Il capitolo si conclude con alcune osservazioni relative agli sviluppi futuri.

Capitolo 1

Background e tecnologie

Il mondo contemporaneo è sempre più permeato dalla tecnologia: a esserne coinvolto non è il solo ambito lavorativo, ma anche il quotidiano, con innegabili mutamenti comportamentali individuali e sociali.

La tecnologia è sempre più rivolta alle masse sia per quanto riguarda i costi contenuti, sia per la semplicità d'uso. I dispositivi diminuiscono in peso e dimensioni divenendo portatili se non addirittura "indossabili", per contro hanno capacità di calcolo sempre più elevate, un set di sensori nutrito e numerose possibilità di connessione. Tutto ciò ha permesso di sviluppare applicazioni e servizi un tempo impensabili, nel senso letterale del termine.

La tesi interviene su uno degli applicativi ViDiTrust che si pone l'obiettivo di frenare la concorrenza sleale combattendo il fenomeno della contraffazione. Si vuole offrire un servizio di verifica di autenticità del prodotto che goda delle seguenti proprietà:

- **Autonomia**.
- **Leggerezza**.
- **Protezione**.

La prima qualità indica che la verifica può essere eseguita personalmente dal consumatore, sul posto e precedentemente all'acquisto. La leggerezza prevede che il servizio venga realizzato tenendo in considerazione

le reali capacità di esecuzione dei dispositivi mobili per cui è pensato, gestendone le risorse in modo oculato. Infine, il concetto di protezione ha un duplice significato: da un lato indica che è compito del sistema controllare l'uso appropriato del servizio di verifica prevenendone abusi e accessi illeciti; dall'altro che il segreto industriale ViDiTrust venga tutelato.

L'insieme delle proprietà sopra descritte sottende un servizio realizzato da più entità software distinte, sbilanciate in termini di complessità logica e applicativa, ma in grado di comunicare e coordinarsi.

1.1 Evoluzione e dispositivi mobili: smartphone

Il cellulare è un dispositivo in continua evoluzione e a oggi si presenta in una nuova forma, generalmente indicata con il termine *smartphone*. La sua diffusione non ha subito battute di arresto neanche in tempi di crisi e se in Italia, tra il 2009 e il 2010, le vendite dei cellulari sono aumentate del 32%, nello stesso periodo i "telefoni intelligenti" hanno fatto registrare un incremento del 72% [Srl11].

In questi dispositivi, le native caratteristiche di telefonia vengono potenziate con quelle di gestione dei dati personali tanto da somigliare sempre più a dei personal computer.

Piuttosto che un PC alleggerito e dotato di capacità telefoniche, è più corretto considerare lo smartphone come un cellulare arricchito da molteplici applicazioni, sopratutto in ambito multimediale.

Piuttosto che un computer alleggerito e dotato di capacità telefoniche, è più corretto considerare lo smartphone come un cellulare arricchito da molteplici applicazioni, sopratutto in ambito multimediale. Storicamente, i dispositivi mobili nascono con l'unico scopo di realizzare un telefono senza fili: per la massa di utilizzatori questo è un concetto semplice da far proprio e usare, diretta estensione di un'idea di telefonia acquisita da generazioni. Ignorando le complessità tecniche, realizzative ed economiche della rete mobile e dei protocolli di comunicazione, il cellulare si è dimostrato un prodotto utile e accessibile alla gente comune, fino a divenire uno status symbol. La sua evoluzione è per certi versi opposta a quella dei

personal computer. Nati per soddisfare esigenze di puro calcolo, gli elaboratori erano rivolti a un pubblico di tecnici esperti o di patiti elettronici e informatici, una ristretta cerchia di persone. Solo in seguito, affermando sempre più la loro natura *general purpose*, sono nate applicazioni appetibili per molti utenti, da cui la diffusione. In quest'ottica, mentre le interfacce di sistemi operativi e programmi per computer sono via via andate semplificandosi, viceversa quelle dei dispositivi cellulari si stanno complicando, o per meglio dire, arricchendo. Ciò discende dalla necessità di gestire dispositivi sempre più potenti e flessibili, nonché dalla realizzazione di servizi e applicazioni adatte a un pubblico vasto, maturo ed esigente, ma poco dedito a installazioni e configurazioni.

Gli smartphone presentano un insieme di caratteristiche comuni oltre al tradizionale servizio di telefonia mobile e alla possibilità di generare traffico dati. L'hardware odierno consente di disporre di schermi LCD di risoluzione apprezzabile e sensibili al tocco, fotocamere munite di flash e sempre più spesso di autofocus, una nutrita gamma di interconnessioni e molti sensori. Gli standard di comunicazione e connessione gestiti nativamente sono: Bluetooth, USB, spesso il Wi-Fi e il più recente NFC (Near Field Communication). I sensori più comuni messi a disposizione sono l'accelerometro, il giroscopio, il sensore di luce ambientale, quello di prossimità, il magnetometro e altri ancora.

Per una cifra ragionevole (intorno ai 200€, febbraio 2012) è possibile disporre di un ambiente molto ricco e capace di supportare applicazioni semplici da installare, spesso gratuite e che coprono le esigenze più disparate.

1.1.1 Applicazioni per smartphone

Fin dalla sua nascita, il cellulare integra funzionalità di base nel proprio software di gestione, ad esempio la rubrica telefonica. Arricchendosi, il mondo della telefonia mobile ha iniziato a considerare sempre più importanti funzioni secondarie ma appetibili per i propri clienti quali la riproduzione di brani musicali in ambito multimediale. Frequentemente

sviluppati e installati dalle stesse case produttrici dei dispositivi, i software di gestione sono divenuti massicci e articolati dovendo introdurre e controllare servizi sempre meno condivisi ma dipendenti dai singoli modelli di cellulare. Nasce il bisogno di realizzare piattaforme nuove, più efficienti, organizzate ed espandibili, in una parola, sistemi operativi orientati al mobile. Seppur limitati e specializzati rispetto a quelli dei calcolatori elettronici, tali programmi svolgono attività tipiche della categoria: gestiscono le risorse e i servizi, rendono possibile l'esecuzione delle applicazioni e si preoccupano della concorrenza dei thread. Su questo strato comune, ciascun produttore installa un pacchetto di funzionalità dedicate all'utente finale e calibrate sulle specifiche capacità dell'hardware a disposizione.

Oltre l'approccio tradizionale appena descritto, una delle novità più interessanti introdotte dagli smartphone è quella di rendere possibile l'installazione di funzionalità nuove o alternative a quelle fornite, assecondando le necessità e i gusti personali. Comunemente indicate col termine app, le applicazioni costituiscono spesso il discriminante che spinge il consumatore a preferire l'acquisto di uno smartphone a quello di un comune cellulare. Per loro natura, tali programmi risultano essere leggeri a tempo di esecuzione e per quanto riguarda l'occupazione di memoria. Un'altra caratteristica che li accomuna è l'elevata interattività con l'utente: che può muoversi fra schermate e menù in modo molto intuitivo, toccando lo schermo (*tap*).

Con politiche differenti e dipendentemente dal sistema operativo mobile installato, le app possono essere scaricate da appositi "mercati" detti *marketplace* o *store* accessibili dal cellulare o via web. Di recente sono nate piattaforme di distribuzione alternative: alcune create e mantenute dai costruttori hardware offrono applicazioni selezionate o gratuite per i propri clienti, altre ancora si distinguono per essere clandestine o illegali. A titolo di esempio citiamo piattaforme ufficiali quali l'App Store di Apple [Inc12a], l'Android Market di Google [Inc12b], il MarketPlace per Windows Mobile [Cor12].

Scaricare un'applicazione per smartphone è semplicissimo e può avvenire gratuitamente o prevedere un pagamento, generalmente fissato dal-

l'autore. Nella maggior parte di questi casi, la piattaforma di distribuzione tratterrà una percentuale degli introiti come rimborso. Una volta recuperata l'applicazione, l'installazione avviene automaticamente e nella maggior parte dei casi non sono previste operazioni di configurazione. D'ora in avanti l'app potrà essere usata in qualunque momento e ovunque l'utilizzatore deciderà di recarsi col proprio smartphone.

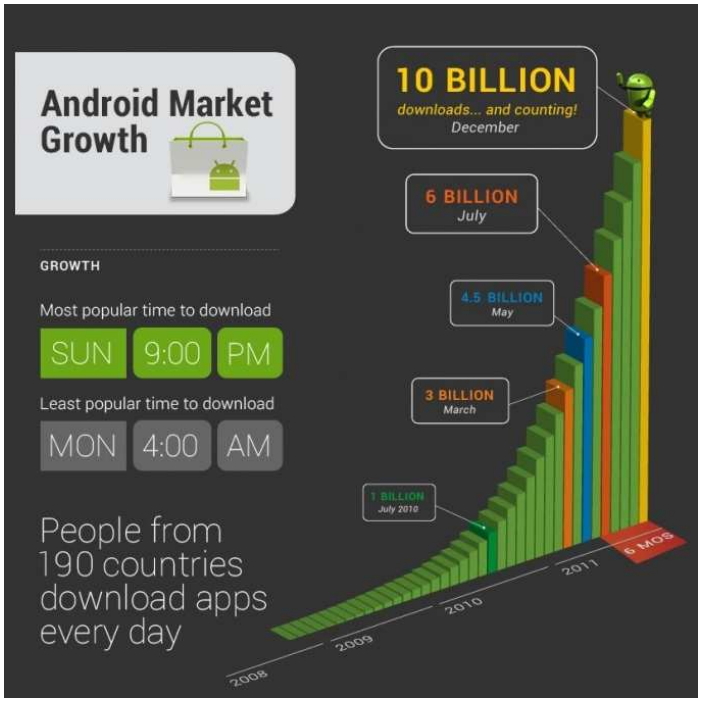


Figura 1.1: Android Market: andamento temporale del download di app

Per comprendere la portata del fenomeno della diffusione delle app, osserviamo i dati relativi all'Android Market riportati in figura 1.1. Da quando è stato introdotto sul mercato (ottobre 2008), il numero di download delle app è cresciuto nel tempo in maniera esponenziale. A fine dicembre 2011 ha raggiunto la quota record di dieci miliardi di applicazioni scaricate, con download quotidiani effettuati da utenti appartenenti a centonovanta paesi differenti.

In figura 1.2 sono riportate le dieci principali categorie di funzionalità richieste: svettano i giochi col 25,6% dei download, a seguire le app d'intrattenimento (12,2%) e gli strumenti di sistema (11,17%).

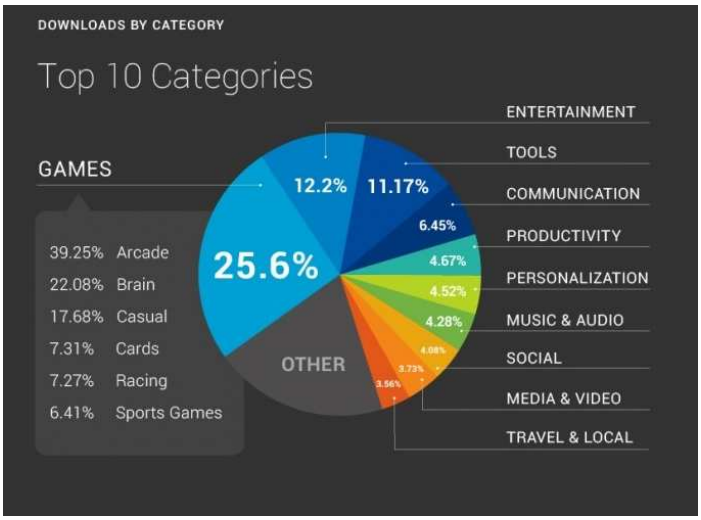


Figura 1.2: Android Market: principali categorie delle app scaricate

I dati numerici riportati illustrano chiaramente che le persone in possesso di uno smartphone utilizzano il dispositivo per una varietà di scopi non strettamente legati alla telefonia. Gli utenti sono interessati, curiosi e sempre più abituati a scaricare nuovi applicativi per divertirsi, ma anche per esplorare nuove funzionalità.

1.1.2 Android

Android è uno dei più moderni sistemi operativi per dispositivi mobili ed è in continua evoluzione. Di recente è stata rilasciata la versione 4.0.3 con nome in codice *Ice Cream Sandwich* capace di gestire in modo unificato cellulari, smartphone e tablet [Dev12].

Inizialmente sviluppato da Startup Android Inc., l'azienda fu in seguito acquisita da Google Inc. che assunse i cofondatori al fine di sviluppare una piattaforma basata sulla versione 2.6 del kernel Linux. Il 5 novembre 2007 il consorzio di produttori Open Handset Alliance (di cui Google è capofila) presentò pubblicamente Android. Il consorzio si prefigge l'obiettivo di sviluppare "standard aperti" per dispositivi mobili e pertanto il sistema Android è rilasciato secondo i termini della licenza Apache. Chiunque può contribuire al suo sviluppo e distribuirlo secondo termini

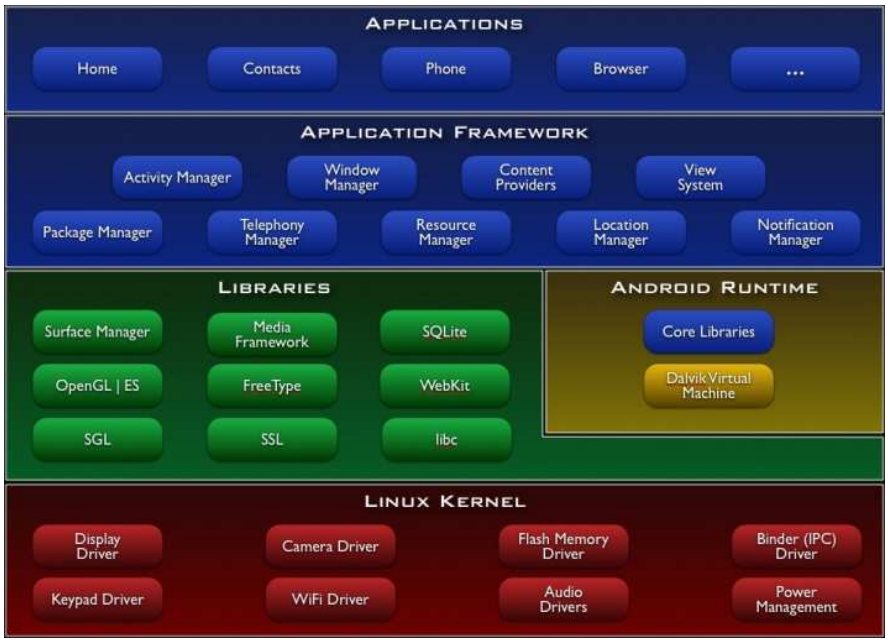


Figura 1.3: Architettura a stack di Android

di licenze non necessariamente "libere". L'unico vincolo degli sviluppatori è quello di rendere noto alla comunità quali componenti software sono state modificate. Questa apertura non ha soltanto conseguenze sociali o di principio, ma anche economiche e tecnologiche.

Per essere più precisi, Android è al tempo stesso un sistema operativo, una piattaforma di sviluppo e una collezione di software di base per l'utilizzo di un dispositivo portatile [Pel09]. Il suo target sono i dispositivi mobili, ma l'architettura che lo realizza ha poco da invidiare a quelle dei comuni sistemi operativi per desktop o laptop.

Essendo sviluppato su kernel Linux (versione 2.6), Android eredita un insieme di valide caratteristiche. Oltre l'affidabilità e la stabilità intrinseche al kernel, la piattaforma risulta essere largamente portabile e ciò ne ha favorito la diffusione. I costruttori di dispositivi non hanno problemi di licenza o di costi nell'implementare Android sui propri prodotti e possono scegliere numerose soluzioni hardware ben supportate dal sistema. Direttamente nel kernel sono inseriti i driver per il controllo dell'hardware del dispositivo. Sopra il kernel (figura 1.3) poggiano le librerie fondamentali tutte mutuate dal mondo Open Source. Fra di esse troviamo Open-

GL dedicata alla gestione della grafica, SQLite per l'amministrazione dei dati e WebKit per la visualizzazione delle pagine Web. L'architettura prevede anche una macchina virtuale e una libreria fondamentale che insieme costituiscono la piattaforma di supporto e sviluppo per le applicazioni Android.

La realizzazione delle app si basa su linguaggio Java, ma alla Java VM di Sun Microsystems (acquisita da Oracle nel 2009), Google ha preferito sviluppare una propria macchina virtuale fondata su Linux. Essa è nota col nome di *Dalvik Virtual Machine*: specializzata per dispositivi mobili, è capace di sfruttare meglio le limitate risorse hardware disponibili [Bur09].

Nel penultimo strato dell'architettura (Application Framework) è possibile rintracciare i gestori delle risorse del sistema che permettono di eseguire telefonate e controllarne lo stato, accedere al file system, installare e rimuovere applicazioni, etc.

Sullo strato più alto dell'architettura poggiano gli applicativi destinati all'utente finale ed è qui che eseguono le app scaricate dal market o realizzate in proprio. Gli smartphone includono nativamente un certo numero di funzionalità come il browser web, il player multimediale, un visualizzatore di immagini, il programma che permette di scattare foto; molte altre possono essere aggiunte e ve ne sono sempre di nuove, proposte in continuazione da una florida comunità di sviluppatori.

Figura 1.4: Bug Droid e le App

In figura 1.4 è rappresentata la mascotte di Android, Bug Droid, e alle sue spalle alcune icone relative alle numerosissime app realizzate per il sistema operativo di Open Handset Alliance.

I motivi che spingono i programmatori a prediligere la piattaforma Android sono principalmente tre:

- La programmazione è in stile Java.
- Gli strumenti di sviluppo sono gratuiti.
- Le app possono essere distribuite liberamente.

Il byte code eseguito dalla Dalvik VM è differente da quello Java, ma l'uso di *dex* (Dalvik EXecutable) è trasparente al programmatore.

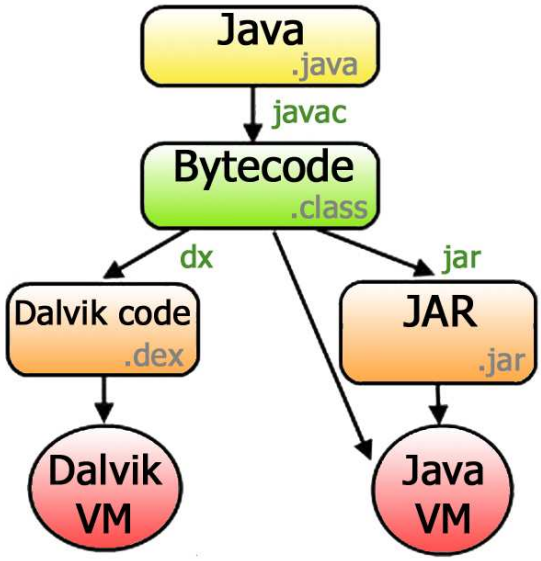


Figura 1.5: Processo di compilazione per Dalvik Virtual Machine

Come si può osservare in figura 1.5, il codice Java verrà compilato dapprima in modo standard fino a ottenerne il relativo bytecode; successivamente, l'ambiente di sviluppo lo trasformerà in dex, più compatto e ottimizzato per eseguire sulla Dalvik VM.

La libreria di base che affianca la macchina virtuale offre buona parte dei package propri di Java Standard Edition, ma il supporto non è completo: si può notare la mancanza di AWT e Swing d'altra parte sostituiti

da librerie grafiche dedicate. I pacchetti fondamentali sono presenti e in larga misura rispettosi dei corrispettivi Java. Da questo punto di vista, Android si dimostra generoso se non addirittura ambizioso: non si limita a garantire una compatibilità con Java ME (Micro Edition), ma offre una libreria molto più completa e orientata ai sistemi desktop (SE). In aggiunta esistono molti package con prefisso *android*, esclusivi di questa speciale piattaforma e capaci di estenderne potenzialità e flessibilità [Pel09].

Un altro aspetto molto importante di Android e che lo accomuna ai moderni sistemi operativi e di sviluppo è la seguente concezione: separare il codice dai dati. Ciò consente di scrivere applicazioni più snelle, semplici da manutenere e con un miglior grado di portabilità.

La filosofia open che lo pervade si ripercuote anche sugli strumenti di sviluppo: l'*Android SDK* (Software Development Kit) è disponibile in forma libera e gratuita per i principali sistemi operativi (Windows, Mac OS X, Linux). Per gli utilizzatori di Eclipse, rinomato ambiente integrato di sviluppo multi-linguaggio e multi-piattaforma, è possibile installare un comodo plugin dedicato: Android Development Tools (ADT) Plugin. Infine, il programmatore ha a disposizione due gestori. Uno di essi si occupa della manutenzione di Android SDK e permette di installare e disinstallare le varie versioni delle API, i plugin, i tutorial, gli esempi, e specifici driver. L'altro è chiamato AVD (Android Virtual Device) Manager e svolge due funzionalità principali:

- Realizzare e gestire dispositivi mobili virtuali.
- Gestire smartphone reali collegati al PC via USB.

Sui dispositivi virtuali o reali è semplice mettere in esecuzione le app sviluppate e testarle per le varie versioni del sistema operativo mobile.

L'immagine riportata in figura 1.6 rappresenta un dispositivo virtuale ottenuto per mezzo dell'Android Virtual Device. Il gestore permette di realizzare e operare con diversi profili in cui lo sviluppatore può impostare le capacità di memorizzazione su SD, la risoluzione dello schermo, l'emulazione di svariati sensori, nonché la versione di Android da adottare (in questo caso la 2.3.3, API 10).

Figura 1.6: Dispositivo virtuale Android ottenuto con AVD

Creata un'app, lo sviluppatore ha la libertà di pubblicarla sul market o di salvarla e rilasciarla come preferisce. Nel primo caso occorre stipulare un contratto di distribuzione per sviluppatori valido per l'Android Market. L'operazione prevede la registrazione e il pagamento di 25$ una tantum. Le app possono essere rese disponibili sulla piattaforma di distribuzione ufficiale in modo gratuito o dietro un corrispettivo che pagherà l'utente interessato all'acquisto. Effettuato il pagamento, il market tratterà il 30% degli introiti. È anche possibile caricare l'applicazione in Internet e renderla disponibile al download, oppure trasferirla direttamente su un qualsiasi numero di smartphone per mezzo del file di installazione *apk* (application package file).

Non è sempre così immediato sviluppare e distribuire app per altri sistemi operativi mobili. Spesso occorre essere muniti di un particolare

hardware per poter realizzare le applicazioni, possedere strumenti di sviluppo proprietari e a pagamento, disporre di licenze costose e soggette a scadenze. Ulteriori complicazioni possono manifestarsi come l'impossibilità di distribuire l'app liberamente senza passare dal market o dover usare uno store "chiuso". In quest'ultimo caso, l'app inviata diverrà disponibile al pubblico solo dopo un processo di validazione da parte di un team specializzato. Nell'eventualità che l'applicazione venga reputata non conforme (per contenuti o tecnologie usate) verrebbe rigettata, altrimenti pubblicata, ma dopo un tempo di incertezza medio di due settimane. Ottemperare le pratiche burocratiche necessarie introduce infine ulteriori tempi morti che si traducono in costi per lo sviluppatore.

Oltre che tecnicamente avanzata e flessibile, la piattaforma Android si dimostra essere una delle più semplici e veloci da usare per chi realizza le app. Il processo di produzione, distribuzione e diffusione è aperto, poco costoso e rapido: ne consegue un mercato molto appetibile formato da una numerosa schiera di utenti e sviluppatori.

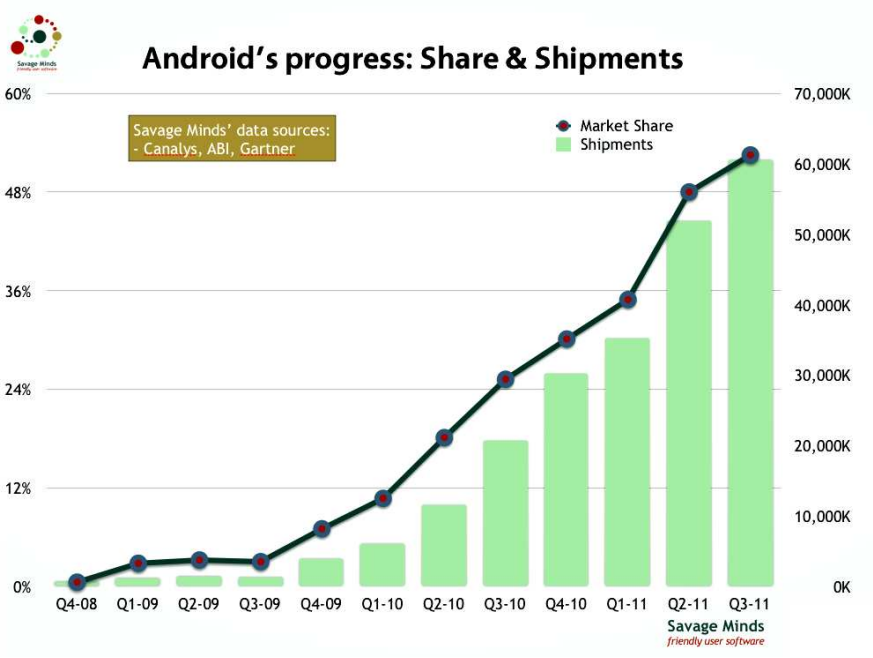


Figura 1.7: Mercato e numero di dispositivi Android venduti

In figura 1.7 è possibile notare la crescita di mercato ottenuta dai di-

spositivi Android. Sull'asse delle ascisse è riportato il tempo suddiviso in trimestri, da fine 2008 (Q4-08) a quasi tutto il 2011 (Q3-11). In corrispondenza di tali riferimenti temporali, sulle ordinate si possono osservare degli istogrammi e dei punti appartenenti a una spezzata. Gli istogrammi indicano il numero di dispositivi venduti che installano Android: nel penultimo trimestre del 2011 si contano oltre sessanta milioni di pezzi. La quota include cellulari, smartphone e tablet senza distinguere in base al costruttore. I punti che formano la spezzata riferiscono nel tempo la percentuale di mercato coperto dai dispositivi Android rispetto ai concorrenti: dal 12% di inizio 2010 si è passati a oltre il 50% nel settembre del 2011.

1.1.2.1 Android e modello di concorrenza

Pensate per dispositivi mobili, le applicazioni debbono prevedere un buon grado di interazione con l'utente, mantenersi intuitive ed evitare lo spreco di risorse. Per tali motivi, i progettisti di Android hanno pensato di definire alcune tipologie di componenti e un meccanismo di comunicazione fra gli stessi in grado di ottimizzare l'impiego del sistema ospite, permettere una elevata personalizzazione e garantire una forte estensibilità della piattaforma.

I componenti base offerti sono le *Activity*, usate per interagire con l'utente e i *Service* per l'esecuzione di funzionalità in background. Esistono poi gli *Intent* con i quali è possibile avviare componenti comuni a più applicazioni e gli *Intent Filter* utili alla loro gestione. Similmente agli Intent, il sistema prevede i *BroadCast Intent Receiver* capaci di generare interazioni fra componenti a seguito del verificarsi di eventi esterni; infine, i *Content Provider* permettono di gestire in modo condiviso fra più applicazioni un certo insieme di dati.

Una parte fondamentale delle applicazioni prevede la gestione dell'interfaccia utente che viene realizzata per mezzo delle activity. Una attività è un contenitore di componenti grafici che potranno occupare tutto il display, parte di esso o essere nascosti completamente; tramite la classe *View*

e sue specializzazioni è possibile realizzare gli elementi di una activity capaci di gestire le azioni dell'utente.

Gli intent realizzano un meccanismo in grado di riutilizzare le activity (e altri componenti) per eseguire quelle operazioni che possono essere comuni a più applicazioni: un classico esempio è la necessità di accedere alla rubrica telefonica e recuperare un contatto da un elenco. Evitare che ciascuna applicazione definisca un proprio metodo di accesso alla rubrica e di gestione dei contatti permette di risparmiare risorse, codice e ottenere vantaggi dal punto di vista dell'usabilità. Un'attività che vorrà accedere alla rubrica del telefono definirà un opportuno intent che poi potrà essere utilizzato per l'avvio di un'altra attività capace di gestirlo. A sostegno del meccanismo appena descritto esiste l'intent filter che permette ai componenti di un'applicazione di dichiarare l'insieme degli intent che sono capaci di gestire.

In Android, i service rendono possibile l'esecuzione di funzionalità che non hanno bisogno di interagire con l'utente e pertanto non necessitano di un'interfaccia grafica. Le operazioni di un service eseguono in background e comunicano con le applicazioni che ne fanno richiesta. Una activity o un altro componente può connettersi al servizio interessato e se ne ha i diritti, avviarlo o fermarlo.

Similmente all'intent filter, il broadcast intent receiver è un meccanismo di collaborazione fra le diverse tipologie di componenti, ma in questo caso gli eventi scatenanti sono esterni all'applicazione. È il caso, per esempio, dell'arrivo di una telefonata, di un certo orario o il segnale di batteria scarica.

Infine, il content provider si occupa della gestione dei dati condivisi da più componenti. Di norma i dati sono privati e legati alle singole applicazione, ma per alcuni di essi è necessario realizzare un *repository* di informazioni a cui è possibile accedere da diversi componenti attraverso una modalità standard.

Da questa descrizione sommaria è possibile intuire che una singola applicazione è composta da molteplici componenti che intervengono in vario grado durante l'esecuzione, con interazioni anche complesse.

Un aspetto fondamentale dell'architettura Android è che il ciclo di vita di ciascun componente è di completa responsabilità dell'ambiente e l'unico punto di intervento da parte dello sviluppatore è quello relativo all'implementazione dei metodi di *callback* invocati a seguito di modifiche nello stato del componente stesso. Per comprendere meglio questi concetti è utile riferirsi alle attività e descriverne la gestione da parte del sistema.

Tipico della maggior parte delle applicazioni è fare uso di diverse attività: ciascuna di esse è eseguita all'interno di un determinato processo Linux e la percezione che ne ha comunemente un utente è quella di "schermata" [Car10].

La piattaforma Android organizza le activity secondo una struttura a stack dove quella posta più in alto è attiva in un certo momento. Visualizzare una nuova schermata, e dunque all'avvio di una nuova activity, porterà quest'ultima in cima allo stack mettendo in uno stato di pausa le altre. Terminato il proprio lavoro, le eventuali informazioni raccolte saranno passate all'attività precedente, la quale diventerà nuovamente attiva. Ottimizzare le risorse prevede che una activity non visualizzata possa essere eliminata dal sistema per poi essere eventualmente ripristinata successivamente.

Gli stati possibili per una activity sono rappresentati in figura 1.8 e per maggiore chiarezza li descriveremo di seguito:

- **RUNNING**: l'attività è in cima allo stack, è visibile e ha il focus; può ricevere gli eventi da parte dell'utente.
- **PAUSED**: l'activity non è attiva ma è ancora visibile per la trasparenza di quelle superiori o perché queste non occupano tutto lo spazio a disposizione. Insensibile agli eventi da parte dell'utente viene eliminata dal sistema solo in caso di estrema necessità.
- **STOPPED**: è lo stato delle activity non attive né visibili; è insensibile agli eventi dell'utente ed è fra le prime candidate a essere eliminata.
- **INACTIVE**: una activity si trova in questo stato quando viene eliminata o prima di essere creata.

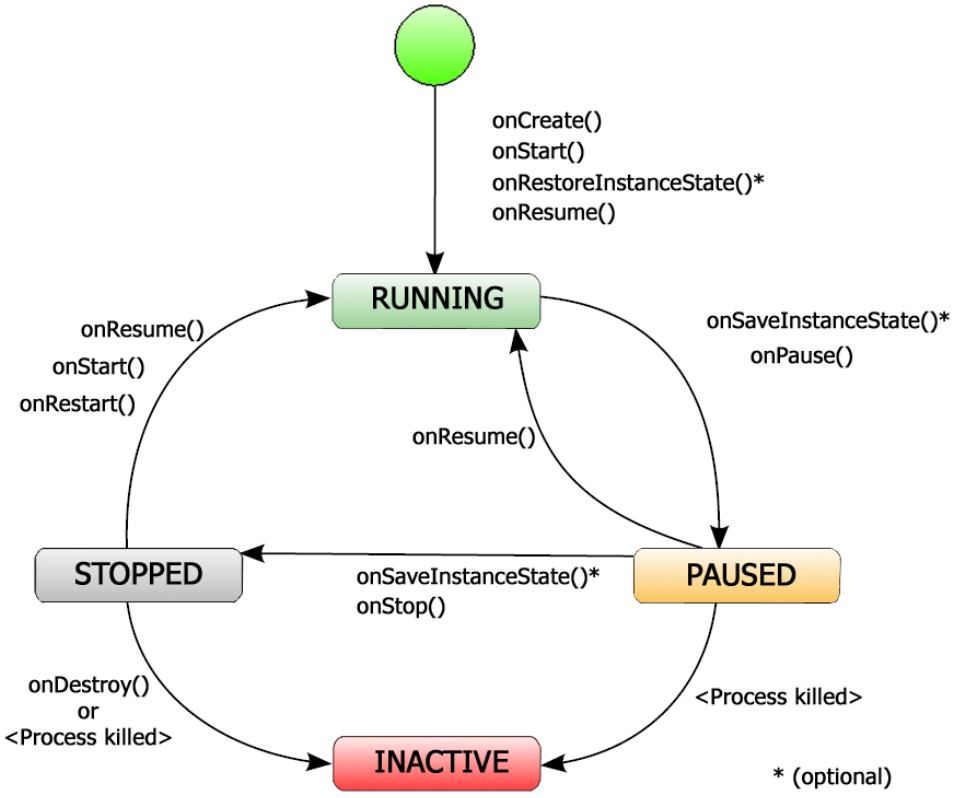


Figura 1.8: Ciclo di vita di una Activity

Quando l'ambiente decide per il cambiamento di stato dell'activity invocherà i metodi di callback predefiniti, anch'essi indicati in figura 1.8.

All'avvio dell'applicazione vengono invocati tre metodi di inizializzazione ciascuno dedicato a un aspetto specifico dell'activity: esistere, essere visibile, essere attiva.

Il metodo *onCreate()* contiene la maggior parte delle operazioni di inizializzazione e può ricevere un parametro di tipo *Bundle* che permette di ottenere un riferimento a un eventuale stato che l'activity aveva prima di essere eliminata dal sistema. Se il metodo va a buon fine, l'activity è stata creata e si prepara alla propria visualizzazione.

Il metodo *onStart()* viene eseguito subito dopo onCreate() e al suo termine l'activity può aver ottenuto il focus ed essere posta in cima allo stack.

Se onStart() riesce nei sui intenti, il metodo *onResume()* viene eseguito ponendo l'activity nello stato RUNNING ove permarrà finché non sarà più quella attiva. Ciò può avvenire per diverse ragioni, ad esempio l'utente può aver premuto il tasto Back sul dispositivo: in questo caso l'attività corrente viene rimossa dalla cima dello stack e messa in stato PAUSED. Affinché ciò sia possibile deve essere invocato il metodo *onPause()* che permette di disabilitare gli input dell'utente.

L'attività che è posta in cima allo stack sarà quella da ripristinare e rendere nuovamente visibile: allo scopo il sistema invoca su di essa il metodo *onRestart()* e subito dopo onStart() e onResume(), secondo le modalità descritte. Notiamo che il metodo di callback onRestart() è molto simile a onCreate(), il primo però è usato su una activity esistente e deve preoccuparsi dell'eventuale recupero del suo stato.

Ripristinata l'attività in cima allo stack, dopo che è stata inserita nello stato di RUNNING e abilitata a ricevere gli input dall'utente, il sistema interviene su quella lasciata in PAUSED. Quest'ultima vi permane se resta parzialmente visibile all'utente, altrimenti su di essa viene invocato dapprima il metodo *onStop()* (con trasferimento nello stato STOPPED) e successivamente *onDestroy()* che la rende INACTIVE.

Il ripristino di un'attività spiegato precedentemente descrive la procedura prevista per una activity in stato STOPPED. Qualora l'attività avesse assunto stato PAUSED, si sarebbe eseguito unicamente il metodo di callback onResume(), come indicato in figura 1.8.

Come già indicato, lo sviluppatore può istruire il sistema Android su cosa fare nel momento in cui si verifichi un cambiamento di stato dell'activity per mezzo dell'invocazione dei metodi di callback relativi, ma non ha controllo su quando ciò accada. È l'ambiente a deciderlo e seppur fortemente influenzato dalla volontà dell'utente che in modo diretto interagisce col display o con i tasti Home e Back del dispositivo, può anche dipendere da eventi esterni. Il caso più ricorrente è quello della telefonata in arrivo: se il telefono squilla mentre si sta usando la calcolatrice, quest'ultima sarà automaticamente messa in stato di STOPPED e mandata in sottofondo. All'utente verrà quindi sottoposta l'activity di gestione delle chiamate e

potrà decidere di rispondere. Conclusa la telefonata sarà in grado di visualizzare l'attività interrotta e riportarla in primo piano (stato RUNNING), riprendendo i calcoli esattamente da dove li aveva interrotti.

Al fine di migliore la comprensione del modello di concorrenza in Android aggiungiamo in questa sezione introduttiva pochi ulteriori dettagli: il concetto di task e l'eliminazione delle attività.

Ciascuna applicazione è eseguita all'interno di un proprio processo Linux ed è frammentata in molteplici componenti con lo scopo di migliorare le prestazioni a tempo di esecuzione, ma è anche possibile eseguire i componenti in processi diversi. Ciò accade di norma quando si utilizza un'attività non definita nell'applicazione, ma richiamata tramite intent, per esempio è richiesto l'accesso a un servizio in background. Nonostante ciò, ovvero vi siano in gioco processi distinti, Android li aggrega in un unico task. Esso consiste in una successione di attività legate all'esecuzione di una applicazione e può includere activity appartenenti allo stesso processo o a processi differenti. È possibile eseguire più applicazioni e le relative attività simultaneamente, ma come descritto in precedenza, soltanto un'activity alla volta può occupare il display, in modo esclusivo. L'alternanza delle stesse è tenuta in considerazione e gestita dal task di appartenenza.

In termini di risorse di calcolo, le attività ibernate non ne consumano e il concetto di chiusura è secondario e tenuto nascosto all'utente. Lo sviluppatore può intervenire direttamente invocando il metodo *finish()*, ma solitamente evita di gestire questo aspetto demandandolo al sistema. Le attività non dispongono di un pulsante "x" o di un tasto equivalente con il quale terminarle, di conseguenza l'utente non può chiuderle, ma solo mandarle in background. I casi in cui un'attività può terminare sono due:

- ha completato la sua esecuzione, è ibernata e il sistema decide di eliminarla;
- il sistema ha poca memoria libera a disposizione e per recuperare spazio inizia a sopprimere bruscamente quelle in sottofondo.

Anche nei casi più critici, le attività da terminare sono scelte oculatamente, in base alla categoria di appartenenza del loro processo. Le tipologie di processo sono [Car10]:

- *foreground* process,
- *visible* process,
- *service* process,
- *empty* process.

I processi a priorità maggiore sono quelli foreground e che si occupano dell'esecuzione dei componenti di interazione con l'utente. Si tratta del processo che sta eseguendo l'attività in cima allo stack, le azioni d BroadCastReceiver o i metodi di callback di un particolare servizio. Questi non saranno eliminati se non nei casi estremi in cui scarseggiassero le risorse per la loro stessa esecuzione.

I processi visible sono a priorità inferiore ai precedenti, ma anch'essi importanti poiché eseguono le attività nello stato PAUSED: visibili parzialmente, non interagiscono con l'utente. Anche questi processi saranno eliminati per condizioni critiche del sistema, ma comunque prima di quelli foreground.

I service process includono i processi che non hanno bisogno di un'interfaccia per relazionarsi con l'utente, ma pur sempre di elevata importanza, come ad esempio un servizio che riproduce musica.

I processi in background sono invece quelli che si occupano di un'activity che non è più visibile all'utente e sulla quale è stato invocato il metodo onStop(). Il numero di questi processi è solitamente elevato rispetto a quelli foreground o visible e il sistema li ordina in base al tempo trascorso dall'ultimo utilizzo, attraverso la lista LRU (*Last Recently Used*).

L'ultima categoria di processi prevista da Android viene definita empty in quanto non legati ad alcun componente predefinito sulla piattaforma: sono tra i primi candidati all'eliminazione.

1.2 Servizio remoto

Durante il corso della giornata, moltissime persone si avvalgono di servizi remoti. Il loro impiego è così consolidato ed efficiente che è divenuto scontato in molti ambiti, fino a divenire "invisibile" per l'utilizzatore.

In prima battuta è possibile accontentarsi di descrivere un servizio remoto come il soddisfacimento di una esigenza per mezzo di una richiesta inoltrata a una attività che opera a distanza. Nel nostro ambito, gli elementi utili a fornire un servizio remoto sono apparecchiature elettroniche munite di risorse di elaborazione e di comunicazione. A differenza di "locale", il termine "remoto" indica che il servizio è richiesto all'esterno del dispositivo in uso, il quale generalmente si limita a farne domanda e riceverne i risultati.

Esempi comuni di servizio remoto sono la consultazione di pagine web, l'home banking, servizi di chat, audio e video streaming, condivisione e backup di dati e molti altri ancora. I più noti e diffusi sono servizi web e prevedono l'accesso a Internet.

1.2.1 Architettura di riferimento: Internet

Per poter realizzare un servizio remoto che si avvale di una pluralità di dispositivi, è utile comprendere come questi debbano essere intesi e organizzati. In realtà, l'architettura è strettamente legata al tipo di servizio considerato, ma limitatamente alla classe di nostro interesse (attività Web), l'organizzazione di Internet risulta essere il riferimento principale. Essa prevede un insieme di nodi distinti ed eterogenei in prestazioni e capacità. Ciascun nodo è autonomo in risorse ed esegue in modo indipendente dagli altri, tuttavia esiste un aspetto che li accomuna tutti: la capacità di connessione alla rete. Internet (con la "i" maiuscola) è nello specifico la rete ottenuta come interconnessione di reti ed è pensata e realizzata in modo gerarchico a causa della sua varietà e vastità.

I nodi più esterni sono comuni all'esperienza degli utenti e prevedono elaboratori e dispositivi mobili: personal computer, laptop, netbook, tablet, smartphone e simili. La rete è un'infrastruttura composta da un coacervo di tecnologie e strati software decisamente complesso. In essa sono da includere i collegamenti fisici e meccanici, i protocolli di comunicazione e un insieme di intermediari (dispositivi specializzati) utili alla sua gestione, nonché artefici della connessione fra i nodi.

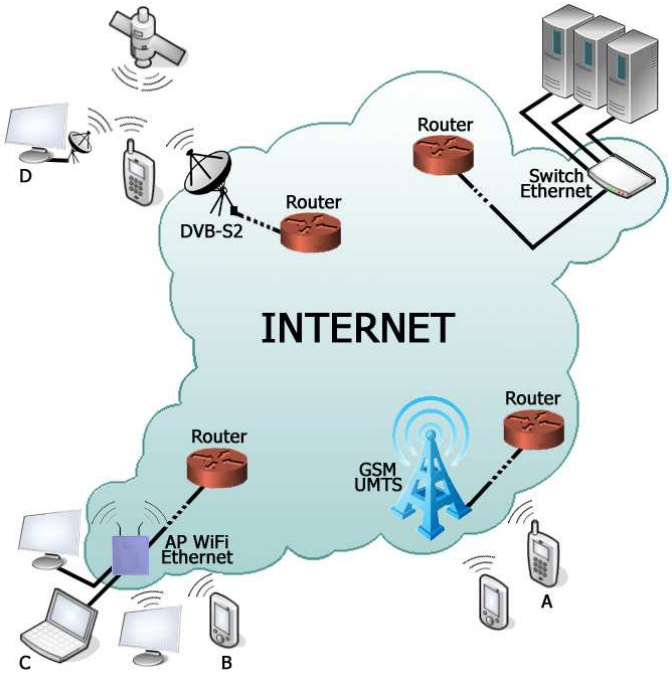


Figura 1.9: Schematizzazione dell'architettura Internet

In figura 1.9 sono indicati vari terminali (nodi) usati dagli utenti. Posti agli estremi di Internet e usando tecnologie e protocolli differenti, i dispositivi accedono alla grande rete e sono abilitati a richiedere servizi remoti alle macchine che li offrono. Possiamo ad esempio notare come il cellulare A acceda ad Internet attraverso la rete mobile, lo smartphone B usando una connessione Wi-Fi, il laptop C via cavo e il pc desktop D con un collegamento via satellite.

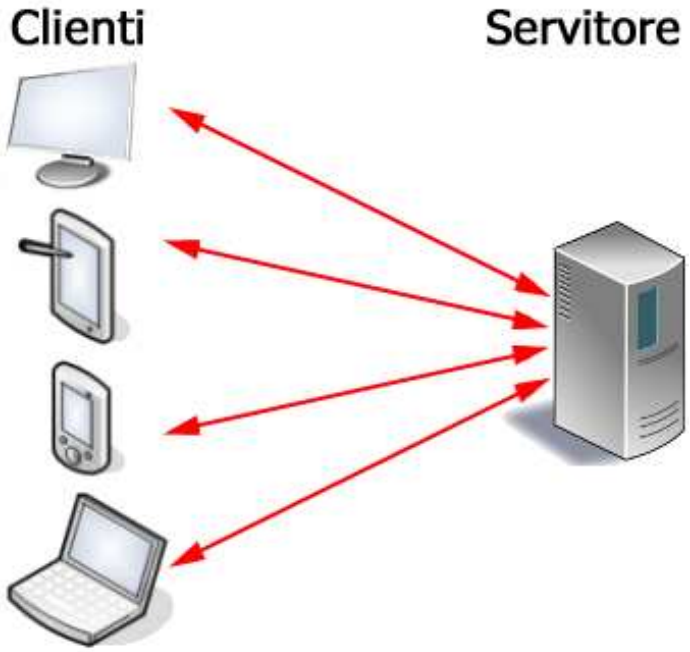
Realizzare un servizio remoto significa mettere in comunicazione almeno due nodi: nel caso più comune di attività web, occorre interfacciarsi con Internet.

1.2.2 Il modello Cliente/Servitore

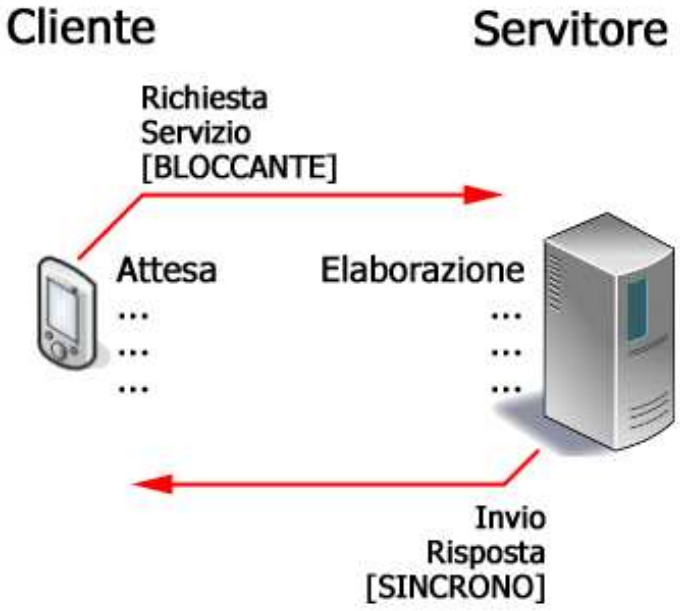
Uno dei modelli più tradizionali per realizzare un servizio remoto è quello Cliente/Servitore. Per quanto detto nel paragrafo precedente, cliente e servitore possono essere considerati nodi della rete e dunque dispositivi fisici, ma più spesso vengono intesi come processi in esecuzione su tali macchine. Ad esempio, l'attività di consultazione di una pagina web prevede l'utilizzo di un browser (applicazione in esecuzione sul cliente) che richiede e visualizza le informazioni fornite da un Web Server (programma ospitato sul servitore). Generalizzando, un cliente è un nodo capace di eseguire richieste di servizio remoto per mezzo di un apposito software applicativo, un servitore è un nodo distinto che ospita funzionalità progettate per rispondere ed evadere tali richieste.

Un modello Cliente/Servitore è di alto livello e garantisce un buon grado di trasparenza che si rifletterà sulle caratteristiche del servizio remoto. Come mostrato in figura 1.10(a), il modello è molti a uno e cioè esiste un servitore che offre il servizio e svariati clienti che possono richiederlo secondo i propri tempi e necessità. È sincrono e bloccante e ciò significa che effettuata una richiesta di servizio, il cliente si aspetta una risposta dal servitore e nell'attesa non compie altre operazioni (vedere figura 1.10(b)).

Per comprendere meglio queste due caratteristiche fondamentali del modello, torniamo all'esempio della consultazione di una pagina web. Più utenti potrebbero accedere contemporaneamente alla stessa risorsa online, ad esempio la pagina di un blog, e dunque svariati browser (molti clienti) inoltrerebbero la richiesta in rete. Indipendentemente dal loro numero e dal tipo di programma o dispositivo usato, tutte le interrogazioni verrebbero inoltrate allo stesso servitore (uno). Effettuata la richiesta (aprendo un segnalibro o inserendo un indirizzo web nell'apposita barra), il browser resta in attesa di ricevere le informazioni necessarie e ciò è normalmente testimoniato da un'icona animata che indica il caricamento della risorsa. La situazione appena descritta individua un sistema sincrono e bloccante: finché non ottiene la risposta dal servitore, il cliente non esegue altre operazioni.



(a) Modello molti a uno



(b) Modello sincrono e bloccante

Figura 1.10: Modello Cliente/Servitore

Il modello cliente/servitore è dinamico e ciò è tipico dei sistemi distribuiti che rendono possibili i servizi remoti. Come detto, il cliente conosce il servitore a cui rivolgersi, ma questa informazione che realizza il legame (binding) fra le due entità non è nota a priori, ma viene risolta a tempo di esecuzione [Cor10]. Proseguendo nel nostro esempio, supponiamo di conoscere l'indirizzo web ove è raggiungibile la pagina del blog. Inserita l'informazione nel browser e premuto il tasto invio, solo in quel momento il cliente cercherà di determinare l'identità del servitore, risolverla e instradare la sua richiesta.

Infine, un'altra caratteristica tipica del modello è la sua asimmetria: al cliente è nota l'identità del servitore, ma non è vero il viceversa. Solo in casi di necessità il servitore risolve l'associazione fra una richiesta e il cliente che l'ha eseguita.

Da questa descrizione preliminare del modello, si possono dedurre alcuni aspetti secondari non trascurabili.

Il servizio remoto diviene oggetto di un forte accoppiamento: può essere richiesto e ottenuto solo se vi è compresenza temporale di cliente e servitore. Inoltre, se la domanda di una certa funzionalità può avvenire in qualsiasi momento, il servitore preposto deve essere sempre attivo e disponibile. Forzando l'esempio che ci ha sinora accompagnati, supponiamo che un blog molto interessante sia ospitato su un server web che esegue su un computer presente nella camera dell'autore. Per un qualsiasi problema alla rete elettrica casalinga o semplicemente se per evitare caldo, consumi e rumori notturni il blogger decidesse di spegnere la macchina, nessun browser riuscirebbe più a recuperare i suoi articoli.

È anche necessario introdurre accorgimenti affinché un servizio bloccante non renda inattivo permanentemente un cliente in attesa di una risposta mancata. Un servizio remoto richiesto via Internet potrebbe soffrire di problemi di connessione, latenze eccessive dovute alla congestione del traffico di rete oppure si verifica il problema dell'invio di una richiesta a un servitore non attivo. In questi casi il supporto dovrà inviare un'indicazione di errore al cliente.

Alla luce di tutto ciò, si può intuire che la realizzazione di un servitore

prevede una complessità concettuale e implementativa molto maggiore rispetto a quella prevista per la progettazione di un cliente.

Preoccupazione del servitore è garantire la correttezza e l'integrità dei dati sulla base dei quali offre il servizio, soprattutto se prevede accessi concorrenti agli stessi. In caso le operazioni da eseguire fossero riservate, i clienti devono essere autenticati e dimostrare di possedere le necessarie autorizzazioni a procedere. Queste condizioni operative comporterebbero anche l'opportuna gestione di informazioni riservate e della privacy.

Una importante distinzione preliminare riguardante la realizzazione di un servitore viene fatta in base alla gestione della molteplicità dei clienti e delle loro richieste: si può rendere l'accesso al servizio concorrente o sequenziale. Nel primo caso, le richieste dei clienti accumulate su una coda vengono accolte una alla volta e ciò può dar luogo a ritardi anche consistenti nell'erogazione del servizio. Un servitore concorrente (non necessariamente parallelo) può gestire più richieste allo stesso tempo abbattendo i ritardi e favorendo la disponibilità delle risorse remote, ma tutto ciò ne accresce la complessità progettuale [Cor10].

Un ulteriore elemento di distinzione relativo al modello cliente/servitore è se durante l'interazione fra le parti si debba mantenere uno stato (*stateful*) oppure no (*stateless*). Nel caso stateless, non si tiene traccia dello stato e ogni operazione o messaggio è completamente indipendente dagli altri. Viceversa, quando è necessario disporre della storia dell'interazione, occorre mantenerne lo stato: una o più operazioni possono dipendere dall'esito delle precedenti. Lo stato è usualmente memorizzato nel servitore e deve esserne in grado per ciascuna richiesta di servizio, oppure demandare tale necessità (per esempio allo stesso cliente) e gestirne il recupero.

Un servitore stateful garantisce efficienza grazie a operazioni più snelle, messaggi di dimensioni contenute e una migliore velocità di risposta. Anche in questo caso la progettazione è però più complessa: per gestirne la storia, il servitore deve essere in grado di identificare il cliente specifico; inoltre, per accessi contemporanei allo stato, occorre amministrare correttamente la concorrenza.

Il caso stateless è più affidabile in presenza di malfunzionamenti, soprattutto causati dalla rete, ma è possibile realizzarlo solo se il protocollo applicativo è progettato con operazioni idempotenti. Esse producono lo stesso risultato a fronte degli stessi input, a prescindere dal numero di attivazioni consecutive effettuate.

Il modello cliente/servitore si presta bene alla realizzazione dei servizi remoti, specialmente di quelli orientati al web. Nel particolare, la ridotta complessità e i minori requisiti di calcolo e di memoria dell'applicazione cliente, rende quest'ultima adatta all'implementazione anche sui dispositivi mobili come gli smartphone. Viceversa, il software del servitore sarà in esecuzione su una macchina remota dedicata e facilmente raggiungibile.

1.2.3 Dal modello Cliente/Servitore a SOA

Nel tempo, un insieme di necessità ha condotto gli sviluppatori a realizzare varianti del modello cliente/servitore o a progettarne di completamente differenti per soddisfare esigenze che poco vi si adattavano. Realizzare applicazioni basate su servizi remoti prevedeva anche aggiustare (molto più spesso riscrivere) il codice relativo alle comunicazioni e all'interfacciamento delle parti o allo scambio dati. Sempre più forte veniva sentita l'esigenza di un supporto capace di gestire tali elementi comuni in modo semplice e generale, esonerando lo sviluppatore da un gravoso compito. In questa riflessione preliminare sui servizi remoti, molti altri aspetti accessori ma non meno importati sono da tenere in considerazione: l'adozione di linguaggi di programmazione orientati agli oggetti ha portato molti utili benefici come pure l'invocazione di procedure remote (*RPC*), ma con notevoli problemi di standardizzazione e frammentazione. Le difficoltà appena esposte divengono critiche quando occorre operare a livello aziendale (*B2B, Business to Business*) piuttosto che limitarsi a considerare servizi per gli utenti finali. Gli strumenti d'impresa e i servizi su cui poggiano (remoti e non) devono essere di facile integrazione, superare le eterogeneità e supportare anche sottosistemi *legacy* [Cor10].

Da queste premesse nasce una nuova idea che si identifica e concretizza in *SOA: Service Oriented Architecture*. Il fulcro dell'architettura è il servizio e non il servitore o l'elaboratore specifico su cui esso è in esecuzione: occorre superare le difformità rivedendo le relazioni fra le parti tutte in termini di contratto astratto di servizi offerti e richiesti. SOA è un modello in cui ciascuna interazione si realizza tramite servizi che sono del tutto indipendenti dalla piattaforma, dalla comunicazione e dalla rete. Allo stesso modo, anche le operazioni sono definite in modo indipendente tra loro e dalla piattaforma in cui sono realizzate, inoltre avranno proprietà note e negoziate prima dell'uso.

SOA vuole offrire le capacità di descrivere, trovare e comunicare con i servizi disponibili. Essi sono da intendersi come astrazione di un processo, risorsa o applicazione, possono essere standardizzati in interfaccia, resi pubblici e noti.

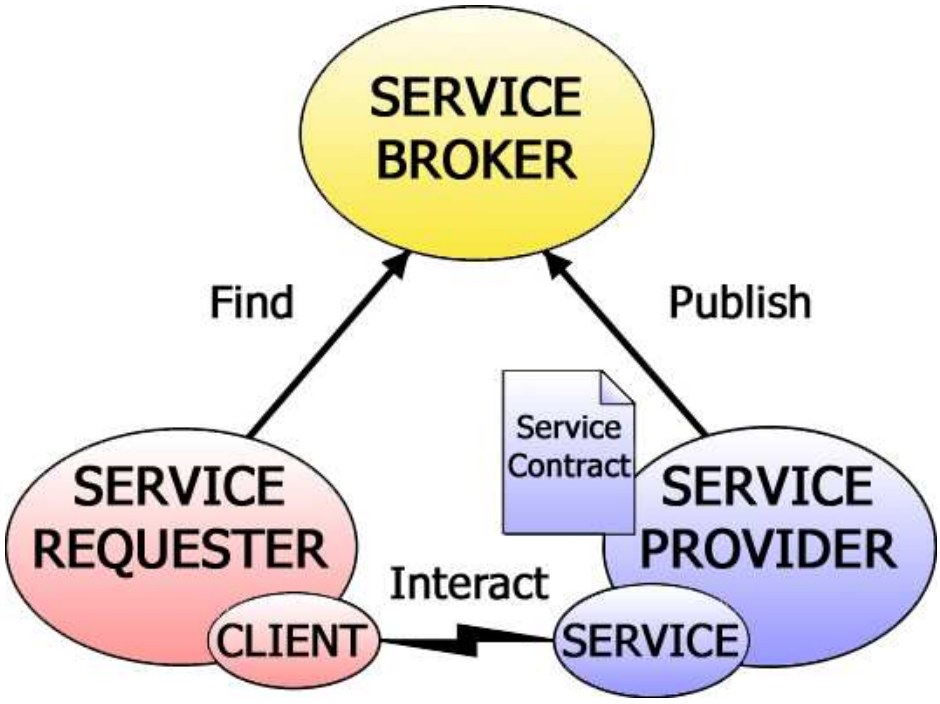


Figura 1.11: Service Oriented Architecture

Come è possibile osservare in figura 1.11, i servizi sono offerti dal Service Provider che li pubblica per mezzo di interfacce (Service Contract) su appositi sistemi di nomi definiti ad-hoc (Service Broker). Su iniziativa del Client, quando e se necessario, i servizi vengono cercati sul Broker per mezzo del Service Requester. Individuato e risolto il servizio, inizia l'interazione fra le parti.

Un servizio pensato e realizzato in SOA è:

- riusabile, cioè può essere riutilizzato anche in altri contesti;
- formale, sancisce in modo non ambiguo i termini e il contratto dello scambio di informazioni;
- a scarso accoppiamento, non presuppone conoscenze pregresse dell'ambiente d'uso;
- a black box, nasconde la logica della soluzione specifica scelta;
- autonomo, non dipende dal contesto ed è capace di autogestione;
- privo di stato, lo stato interno deve essere minimizzato (*stateless*), le invocazioni saranno indipendenti;
- soggetto a *discovery*, deve poter essere ritrovato dinamicamente in base all'interfaccia e in modo standard;
- componibile, cioè capace di aggregare le proprie caratteristiche a quelle di altri servizi per realizzarne di nuovi.

1.2.4 Web Services

Da non confondere con i servizi web, i *Web Services* sono una specifica diversa e precisa per ottenere a livello Web tutto quello che è possibile realizzare a livello di linguaggio di programmazione e di computazione [Cor10]. Web Services (WS) può considerarsi una particolare incarnazione di SOA e anch'esso prevede componenti software indipendenti dalla piattaforma e dall'implementazione, operando tipicamente B2B [PD08].

La vicinanza dei due modelli può essere notata dal raffronto delle figure 1.11 e 4.2.

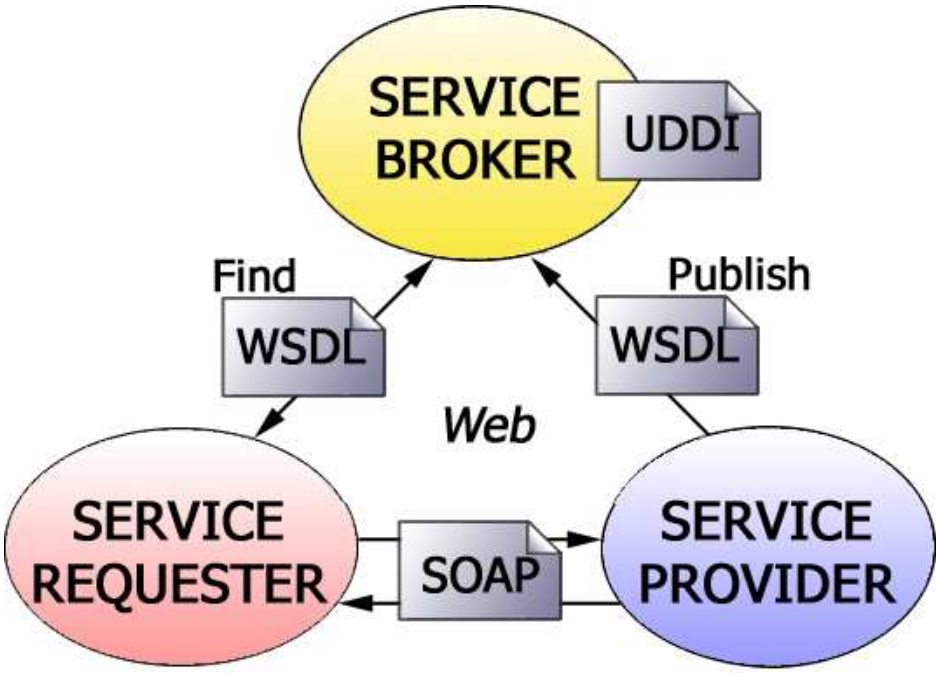


Figura 1.12: Architettura di Web Services

Web Services realizza un *middleware* con organizzazione standard per la fruizione di servizi in ambito Web, si poggia su un ambiente HTML compatibile e fa uso di strumenti che lo estendono come *XML* (eXtensible Markup Language). L'uso di HTTP è relativo al trasporto dei messaggi e ciò determina un considerevole vantaggio, soprattutto in ambito aziendale: di norma non occorre applicare modifiche alle regole di sicurezza implementate per mezzo dei filtri sul firewall.

Facendo riferimento alla figura 4.2, possiamo osservare che gli elementi WS sono descritti usando un linguaggio di rappresentazione del servizio (*WSDL*), pubblicati in un registro di servizi (*UDDI*) e individuati mediante una politica definita con meccanismi standard di discovery (a runtime o a tempo di progetto). Rispettando il modello SOA, i servizi possono essere composti fra loro per realizzane di nuovi e invocati mediante una API remota, solitamente tramite la rete web a mezzo di *SOAP*. Particolare attenzione meritano i protocolli usati da Web Services appena indicati e, ai fini della tesi, SOAP.

1.2.4.1 WSDL: Web Services Description Language

WSDL (*Web Services Description Language*) è un linguaggio basato su XML usato per la creazione di "documenti" descrittivi delle modalità di interfacciamento e di utilizzo del Web Service. Esso formalizza e realizza la descrizione dei servizi che si possono richiedere e ottenere. Per raggiungere lo scopo, il linguaggio è pensato per specificare in modo esatto, portabile e standard il formato dei messaggi di richiesta e di risposta e tutti i dettagli necessari. Il suo compito è suggerire cosa un servizio può fare, ove risiede e come può essere invocato. L'approccio considerato permette di usare anche un WS non noto: se ne richiede il documento WSDL e lo si analizza al fine di recuperarne la locazione, le chiamate dei metodi e i parametri, nonché le modalità di accesso [Cor10].

WSDL rappresenta i Web Service per mezzo di messaggi da scambiare fra Requester e Provider in cui si hanno descrizioni dapprima astratte e poi concrete. I vantaggi di questo approccio sono essenzialmente due:

- Le descrizioni astratte possono essere riutilizzate, tutte o in parte, nella creazione di nuovi servizi.
- Basandosi sulla stessa rappresentazione astratta, il servizio può avere implementazioni diverse.

A livello astratto, il servizio viene definito in modo generico riferendosi alle operazioni offerte e al tipo di messaggi scambiati per ciascuna di esse: ciò lo rende generalizzabile, flessibile e facilmente estendibile. Gli elementi XML previsti per questa parte sono: *message*, *operation*, *interface*.

A livello concreto, le descrizioni astratte vengono istanziate legandole a una implementazione reale (protocolli, indirizzi di rete, ecc.). Le specifiche concrete sono definite solo in ciascuna delle parti costituenti il servizio e prevedono i seguenti elementi XML: *type*, *binding*, *endpoint*, *service*.

Evitando di aggiungere troppi dettagli, osserviamo la figura 1.13 e concentriamo l'attenzione su come i vari elementi sono messi in relazione. Le interface (X e Y) sono collezioni di operation offerte dal servizio e che lavorano tramite i message scambiati fra requester e provider. Per mezzo

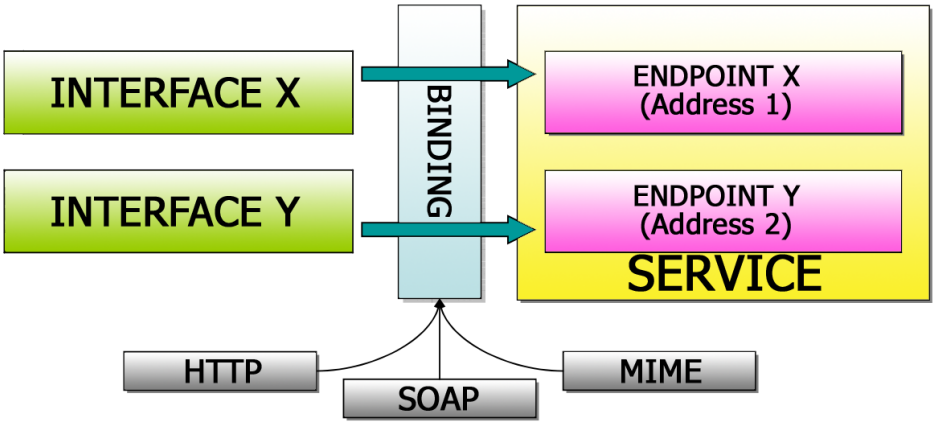


Figura 1.13: WSDL: Descrizione concreta

del binding, ciascuna interface viene legata a un protocollo e a uno specifico formato dati e ci si preoccupa d'indicarne trasporto e codifica: HTTP, SOAP, SMTP, FTP. Le interface istanziate dai binding si avvalgono degli endpoint per ottenere riferimenti concreti a indirizzi di rete reali (Address 1 e 2). Infine, i service sono composti da insiemi di endpoint correlati fra loro che spesso offrono modalità differenti di accesso alla stessa tipologia di servizio.

Segnaliamo infine che ogni interface prevede modalità operative diverse, sincrone e asincrone. Ad esempio, la *Request_response* prevede una risposta da parte del service provider al messaggio inviato del client e ciò caratterizza operazioni sincrone come visto per il modello cliente/servitore.

1.2.4.2 UDDI: Universal Description Discovery & Integration

Un altro protocollo fondamentale di Web Services è UDDI (*Universal Description Discovery & Integration*). Utilizzare un servizio WS prevede innanzi tutto rintracciarlo e pertanto si ha la necessità di realizzare un sistema di discovery ossia di *naming* e *directory*. Per molti aspetti, il protocollo UDDI ricorda il servizio *DNS* di Internet. Quest'ultimo opera a basso livello e permette di tradurre gli indirizzi IP, il primo si occupa invece di rin-

tracciare servizi che a loro volta hanno nomi di host da risolvere mediante DNS. Anche UDDI è stato pensato come unico servizio mondiale (Universal) e seppure non prevede una organizzazione gerarchica, è composto da una serie di server che supportano la replicazione [BCC+02]. L'interazione col protocollo prevede due classi di utenti:

- Publisher, ovvero la compagnia che offre Web Service.
- Client: utente o compagnia che ricerca un Web Service.

Ciò rende necessarie operazioni di base quali la pubblicazione dei servizi da parte dei publisher e la ricerca (discovery) degli stessi da parte dei client. Lo standard UDDI si propone di rintracciare i WS organizzandosi su tre tipi di funzionalità i cui stessi progettisti hanno associato agli analoghi telefonici.

- *White Pages* (pagine bianche): permettono di recuperare un servizio per nome e contengono le informazioni tecniche e gli indirizzi per contattare i publisher per le necessarie negoziazioni preliminari o per ottenerne supporto tecnico.
- *Yellow Pages* (pagine gialle): permettono di cercare e trovare un servizio per categoria nell'ambito di molteplici classificazioni; contengono le informazioni sui WS messi a disposizione dalle aziende.
- *Green Pages* (pagine verdi): forniscono informazioni tecniche aggiuntive sul servizio e possono includere anche la sua descrizione WSDL.

Chiunque è autorizzato a effettuare richieste per ottenere un certo servizio (discovery), al contrario, occorre essere un fornitore di servizi (publisher) per poter registrarne uno presso l'*UBR* ovvero l'*UDDI Business Registry*. La sicurezza è un aspetto fondamentale per il protocollo: chi effettua modifiche deve essere autorizzato in quanto un concorrente potrebbe eliminare i servizi di un altro publisher. Allo scopo è previsto un sistema di autenticazione per mantenere traccia localmente ai server che gestiscono

UDDI delle modifiche effettuate: solo chi ha pubblicato un servizio avrà la possibilità di modificarlo o eliminarlo.

Ciascuna azienda necessita di realizzare gli UDDI per i propri servizi e deve decidere come organizzare il proprio UBR: esiste una classificazione dei registri e se ne hanno di pubblici, privati e condivisi (semi-privati). Per quanto detto, i server su cui opera UDDI non sono coordinati: mancano protocolli ad hoc e anche la volontà da parte delle aziende di condividere i propri spazi per la registrazione di servizi altrui. Come detto, il protocollo considerato è di alto livello e si poggia su SOAP per la trasmissione dei messaggi, ma deve provvedere autonomamente all'aspetto della sicurezza, non prevista al livello sottostante [BCC+02].

1.2.4.3 SOAP: Simple Object Access Protocol

SOAP è un protocollo che nasce dalla convergenza di due attività rimaste disgiunte per molto tempo: le cosiddette "applicazioni middleware" e il *Web publishing*. Uno dei suoi obiettivi principali è l'impiego di XML per effettuare chiamate RPC mediante HTTP. L'introduzione delle procedure remote (e di middleware dedicati) ha di molto alleggerito il lavoro degli sviluppatori che non hanno più necessità di gestire direttamente le funzionalità di rete. La realizzazione di software distribuiti si semplifica, ma se tutto funziona bene in reti private di tipo LAN o intranet, creare applicativi che operino correttamente in Internet è difficoltoso. È possibile che la necessaria riconfigurazione delle regole di filtraggio sul firewall sia incompatibile con le procedure di sicurezza aziendale, inoltre, a fronte di una programmazione semplificata, le applicazioni e i servizi realizzati spesso soffrono di una scarsa disponibilità. In Internet è più semplice consultare un sito web che non raggiungere e utilizzare un'applicazione di questo tipo. SOAP è pensato per risolvere queste limitazioni e dunque consentire chiamate RPC servendosi di un normale server web [BCC+02].

Il protocollo prevede l'utilizzo di XML per la serializzazione dei dati e di servirsi di HTTP per il loro trasporto. SOAP impiega messaggi per trasmettere parametri e valori e per l'invocazione remota di oggetti basa-

ti su tecnologie Web, ma specifica solo alcuni aspetti. Occorre indicare la gestione degli elementi XML, come realizzare il trasporto e lo stile di interazione interessato. Quest'ultimo può essere *RPC style* (Client/Server) e dunque operare in modalità sincrona, *Document style* (*one way*) che prevede un'interazione di tipo asincrona, o anche multicast o di altra natura.

D'altro canto, SOAP non fornisce alcun supporto per informazioni semantiche sul contratto fra le parti, come pure non si occupa dei dettagli locali dell'interazione: configura un protocollo privo di stato e tende a usare GET e POST come operazioni web.

Un messaggio SOAP è un documento XML strutturato come in figura 1.14 e contiene i seguenti elementi [PD08, ACKM04]:

- *Envelope* incapsula il contenuto del messaggio.
- *Header* contiene informazioni aggiuntive ad esempio riguardanti la sicurezza o per diversi destinatari nel caso il messaggio dovesse attraversare più punti di arrivo (opzionale).
- *Body* include le richieste e le risposte, tipicamente il messaggio da comunicare.
- *Fault* incapsula eventuali casi distinti di errore e di eccezione (opzionale).

La struttura di messaggistica definisce gli elementi XML di base in termini di *blocchi SOAP* ove ciascuno di essi è una singola unità logica computazionale. Un blocco è identificato da un particolare combinazione costituita da nome locale e *namespace*. È opportuno evidenziare che SOAP definisce soltanto la struttura dei messaggi e non il loro contenuto, dichiarazioni di tipo o istruzioni utili al processamento. Gli elementi usati per descrivere una richiesta di elaborazione o un risultato vengono definiti in uno schema specifico e utilizzati all'interno della struttura SOAP sfruttando il meccanismo dei namespace.

Figura 1.14: SOAP Envelope

Si definiscono e riferiscono almeno due namespace per SOAP/1.1, uno per l'envelope e uno per la serializzazione:

- `http://schemas.xmlsoap.org/SOAP/envelope/`
- `http://schemas.xmlsoap.org/SOAP/encoding/`

Per la nuova versione SOAP/1.2 vanno rispettivamente considerati:

- `http://www.w3.org/2003/05/soap-envelope/`
- `http://www.w3.org/2003/05/soap-encoding/`

Come inizialmente affermato, con SOAP si riesce a lavorare con RPC veicolate attraverso il protocollo HTTP. La norma delle attività svolte consiste nell'inviare messaggi e ricevere risposte inserendo informazioni secondo le modalità di un linguaggio di programmazione. Ciò consente di realizzare l'indipendenza dalle molteplici realizzazioni concrete dei servizi richiesti e dalla eterogeneità delle diverse architetture. Il raggiungimento di questo scopo prevede come scotto un certa inefficienza dovuta al livello applicativo molto alto con cui il protocollo lavora.

Utilizzando SOAP e gli altri protocolli offerti da Web Services è possibile, nello specifico, realizzare l'interazione classica del modello Cliente/-Servitore, ma senza dover conoscere qual è il server che offre il servizio, in che linguaggio è realizzato o per quale architettura.

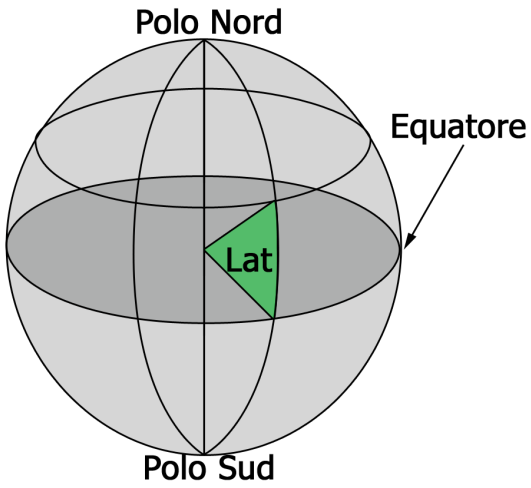
1.3 Localizzazione

Col termine localizzazione s'intende l'individuazione della posizione di un oggetto (spesso schematizzato con un unico punto) rispetto a un sistema di riferimento noto e tipicamente intercalato nel mondo reale. Oltre l'interesse a scopo militare, oggigiorno sono state sviluppate in ambito civile numerose tecnologie di supporto alla localizzazione, sia per realizzare strumenti dedicati alla sicurezza personale (individuazione in caso di valanghe) sia per usi quotidiani come il navigatore satellitare.

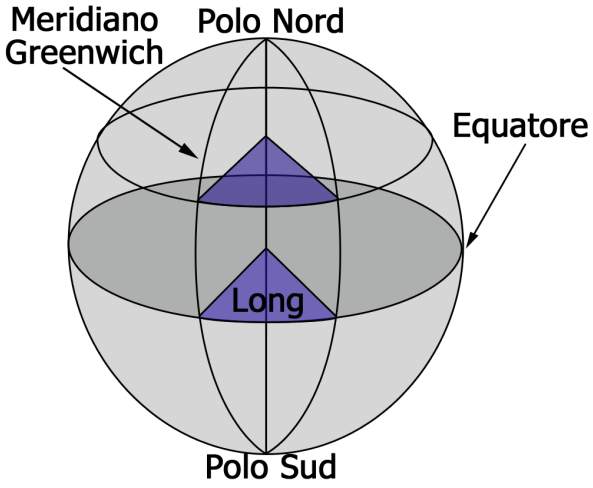
La diffusione di dispositivi mobili equipaggiati con adeguati sensori e connettività, ha reso possibile la localizzazione dell'utente in modo semplice. Questa informazione è stata immediatamente sfruttata da numerose nuove applicazioni appartenenti alla categoria *location-aware*, ovvero consapevoli della posizione: si può pensare a un programma che consenta di conoscere i film in programmazione presso le sale della città in cui ci si trova, oppure a carte geografiche interattive capaci di presentare solo i luoghi di una certa categoria d'interesse (monumenti), fino al già citato navigatore satellitare, tutte funzionalità disponibili su un unico strumento.

Localizzare l'utente, o più precisamente il suo dispositivo mobile, significa identificare la sua posizione sulla Terra (geolocalizzazione) attraverso un opportuno sistema di riferimento. La posizione geografica di un punto sulla superficie terrestre è tipicamente espressa in termini di due coordinate angolari chiamate *latitudine* e *longitudine*.

Osservando la figura 1.15(a), si può notare che la latitudine di un punto geografico è la sua distanza angolare a Sud o a Nord dall'equatore che per convenzione ha latitudine 0°. Altri riferimenti noti sono il Polo Nord che assume latitudine +90° e il Polo Sud con latitudine -90°. Si definiscono *curve di latitudine* l'insieme di punti a latitudine costante: parallele fra



(a) Latitudine



(b) Longitudine

Figura 1.15: Coordinate geografiche

loro sono più comunemente note col termine di *paralleli*. Per definizione l'equatore individua il *parallelo di riferimento*.

La figura 1.15(b) rappresenta l'altra coordinata geografica principale, la longitudine. Essa indica la posizione di un punto verso Est o verso Ovest: le linee che congiungono il Polo Nord e il Polo Sud individuano i luoghi dei punti di stessa longitudine che prendono il nome di *meridiani*. Esiste un meridiano particolare che ha longitudine 0° ed è utile alla determinazione di tutti gli altri. La sua denominazione è quella di *meridiano di riferimento* o

meridiano di Greenwich, dal nome dell'osservatorio di riferimento situato in Inghilterra. La longitudine di un qualsiasi punto del pianeta viene misurata a partire dal meridiano di riferimento e può variare da -180° a +180°, valori coincidenti che individuano il cosiddetto antimeridiano di Greenwich. Per convenzione, un punto a Ovest del meridiano di Greenwich ha una longitudine negativa, un punto a Est ha longitudine positiva.

Infine, esiste una terza coordinata detta *altitudine* o *quota* che individua la distanza verticale del punto geografico dal livello zero, spesso riferito al livello del mare. A differenza delle altre, questa coordinata non è una distanza angolare, ma una lunghezza.

Definito il sistema di riferimento, descriveremo le principali tecnologie utilizzate per esprimere un punto sulla Terra in coordinate geografiche: latitudine, longitudine e altitudine. Utili allo scopo esistono una pluralità di metodi e di strumenti che si differenziano principalmente per il grado di accuratezza con cui riescono a localizzare la posizione del punto sul riferimento. Alcune applicazioni potrebbero necessitare di una bassa risoluzione, o non aver bisogno di conoscere tutte e tre le coordinate geografiche, tipicamente l'altitudine: il programma che consente di conoscere i film in programmazione in città ne è un esempio. Viceversa, le indicazioni stradali fornite da un navigatore risulterebbero inservibili se non fossero accurate con una precisione di una decina di metri.

In figura 1.16 sono riportate alcune delle sigle riguardanti diverse tecnologie a supporto della localizzazione. Esse sono disposte in modo da identificare delle aree rettangolari dipendenti da due fattori: la precisione (espressa in basso) e il campo d'impiego (riportato a sinistra). Il loro grado di accuratezza diminuisce da sinistra a destra: da quelle più precise (cm), alle meno accurate (100m). Di nostro interesse sono quelle posizionate in alto, ovvero incluse in prodotti pensati per le grandi masse. In particolare considereremo i *Mobile Phone* che nell'accezione degli smartphone permettono di ricoprire anche le aree relative al Wi-Fi e al GPS, grazie alla varietà di connettività e sensori integrati.

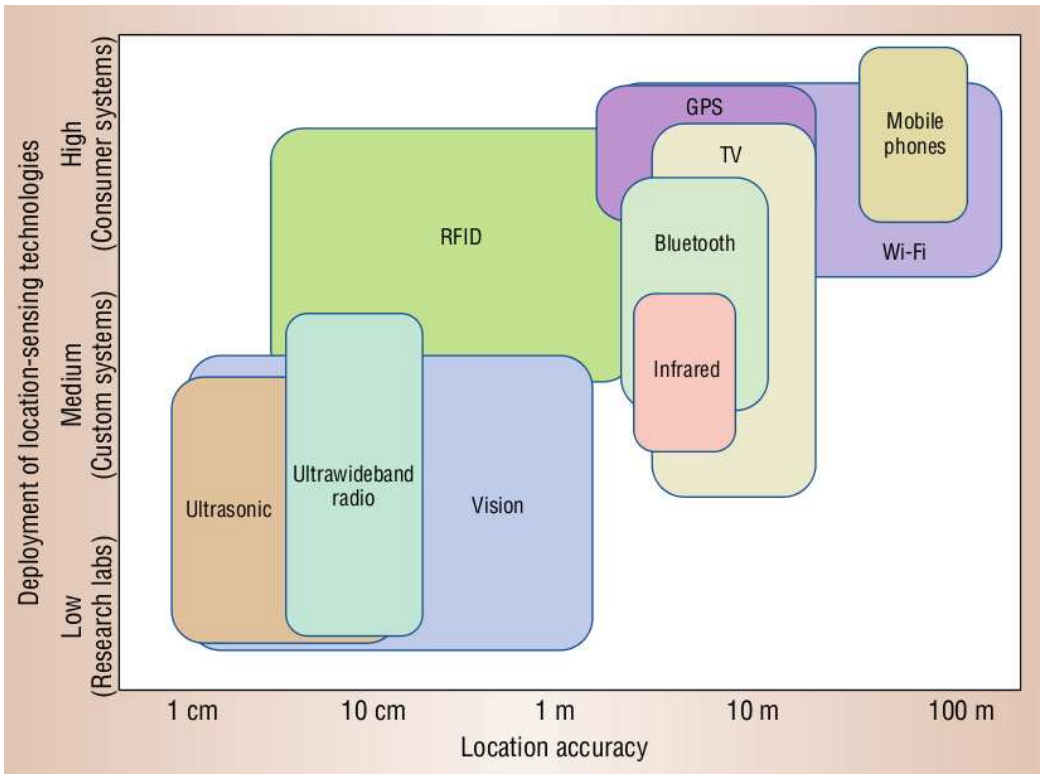


Figura 1.16: Tecnologie utili alla localizzazione

1.3.1 Localizzazione via GPS

Il sistema di localizzazione più preciso ed elaborato fra quelli che esporremo è il GPS, *Global Positioning System* [Bia09] capace di fornire posizione e orario grazie ai dati forniti da un numero di 24/32 satelliti (alcuni di riserva) e da un sistema di controllo a terra composto da apposite stazioni. Avere informazioni sulla propria posizione è possibile in qualunque punto della terra ove vi sia un contatto privo di ostacoli con almeno quattro satelliti: per mezzo di quelli visibili al momento, il ricevitore satellitare calcolerà la distanza da ognuno di essi e userà i dati ricevuti per la localizzazione.

Il sistema GPS è gestito dal governo degli Stati Uniti d'America ed è liberamente accessibile a chiunque sia dotato di un apposito ricevitore [DoD08]. Il suo grado di accuratezza è dell'ordine dei metri, ma ciò dipen-

de da molti fattori: dalle condizioni meteorologiche, dalla disponibilità e dalla posizione dei satelliti rispetto al ricevitore, dalla sua qualità e tipo, dalla riflessione del segnale, dagli effetti della ionosfera, della troposfera e della relatività [Bia09]. Per questioni di sicurezza nazionale, le misurazioni civili sono volutamente rese imprecise anche se nel corso degli anni i fattori di degrado intenzionalmente introdotti sono stati progressivamente ridotti.

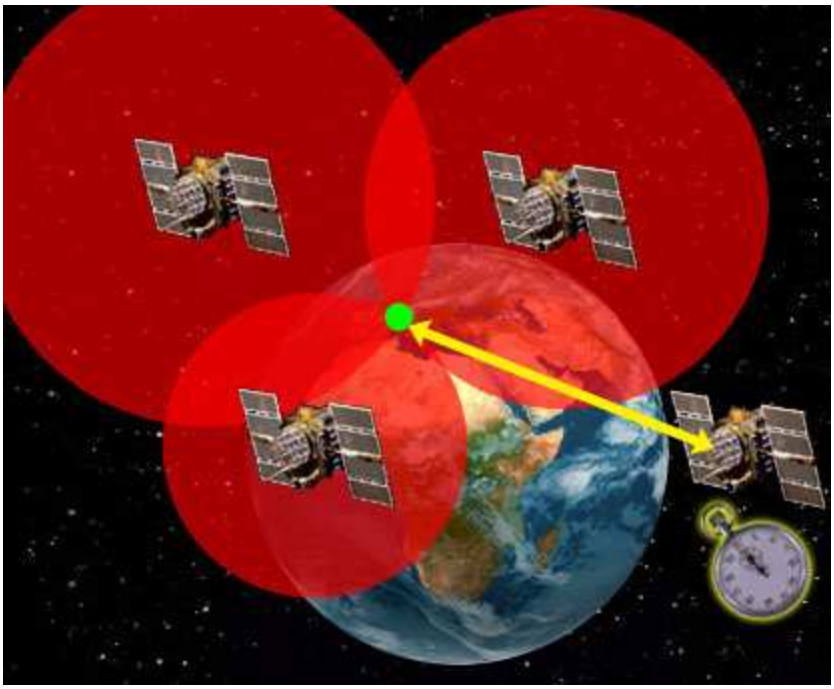


Figura 1.17: Localizzazione per mezzo dei satelliti GPS

Il metodo di calcolo della posizione del ricevitore avviene per mezzo di triangolazione satellitare ove a intersecarsi sono tre sfere, ciascuna centrata in uno dei satelliti visibili considerati, come mostrato in figura 1.17. In verità, sia per conoscere l'informazione oraria, sia per l'elaborazione dei dati, è necessario disporre delle informazioni provenienti anche da un quarto satellite. Rilevare con cura le distanze e dunque la posizione dipende dalla misurazione ad altissima precisione del tempo che intercorre tra l'invio e la ricezione dei segnali scambiati tra satelliti e ricevitore. Un errore anche piccolo (mezzo millisecondo) introdurrebbe una distanza falsata di 150 Km e questo perché le velocità da tenere in considerazione sono re-

lativistiche e confrontabili con quella della luce nel vuoto (approssimata a 300000 Km/s). Per mantenere contenuti i costi dei ricevitori mobili a terra, su di essi vengono installati orologi interni poco precisi, ma capaci di sincronizzarsi con quelli satellitari. Il processo di sincronizzazione avviene all'accensione del dispositivo utilizzando l'informazione (continuamente aggiornata) proveniente dal quarto satellite. La precisione può essere ulteriormente incrementata grazie all'uso di sistemi come il WAAS statunitense [DoD05] o l'EGNOS europeo [Age12]: fra loro compatibili, sono capaci di inviare dei segnali di correzione per mezzo di uno o due satelliti geostazionari. La modalità *Differential-GPS* (DGPS) utilizza un collegamento radio per ricevere i dati da una stazione di terra e ottenere una incertezza sulla posizione di al massimo un paio di metri [MOJ95, HB01]. La modalità *DGPS-IP* sfrutta la rete Internet per l'invio di tali informazioni di correzione anziché le onde radio [GL01].

Un problema del GPS è rappresentato dalla disponibilità del servizio che non è ottenibile in ambienti chiusi o in condizioni atmosferiche particolarmente sfavorevoli. Relativamente al mondo telefonico, la localizzazione via GPS presenta un ulteriore svantaggio poiché prevede un notevole consumo di batteria, un fattore ancora più sentito per gli smartphone. Per ovviare in parte a tali problemi, in molti dispositivi di questo tipo è stato introdotto negli ultimi anni il sistema *Assisted GPS* (A-GPS) [VD09]. Attraverso la rete di telefonia mobile si fanno pervenire al terminale GPS le informazioni sui satelliti visibili dalla cella radio a cui l'utente è agganciato. Assumendo che i satelliti in vista dalla cella siano gli stessi visibili dai terminali sotto la sua copertura radio, i dispositivi dotati di A-GPS possono in pochi secondi ricavare la propria posizione iniziale.

1.3.2 Localizzazione via Rete Cellulare

La localizzazione tramite rete cellulare (network) prevede due vie principali: l'approccio *Cell-based*, ovvero per mezzo di una singola cella telefonica e quello *Triangulation-based*, ove le celle considerate sono (almeno) tre. In ambedue i casi si assume che un telefono cellulare sia sempre

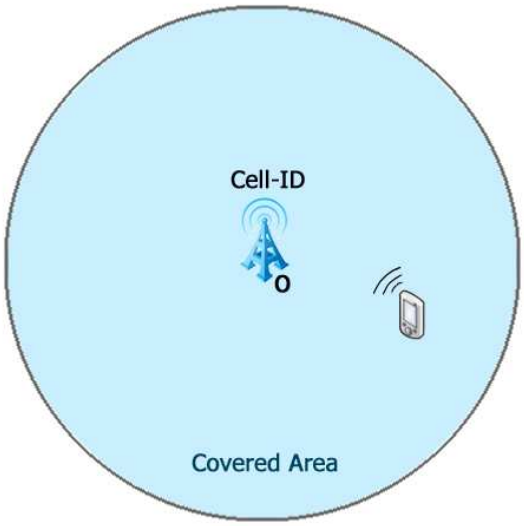
in comunicazione con i più vicini ripetitori di segnale telefonico, anche quando non è in corso alcuna chiamata.

I metodi operano su una *Public Land Mobile Network* (PLMN) e cioè su una rete che ciascun operatore instaura e rende operativa allo scopo di fornire al pubblico il servizio di telefonia mobile. In una singola nazione possono esserci operatori diversi e quindi più reti PLMN distinte. Ciascuna di esse è suddivisa in aree che sono individuate da un codice che i tecnici chiamano LAC (*Location Area Code*) e la cui grandezza non è fissa, ma dipende da come è progettata la rete del singolo operatore. All'interno di queste aree sono presenti svariate stazioni radio base capaci di fornire il servizio telefonico: chiamate comunemente *celle* o BTS (*Base Transceiver Station*), prevedono una numerazione univoca (*Cell-ID*) nella specifica area considerata [Gar07].

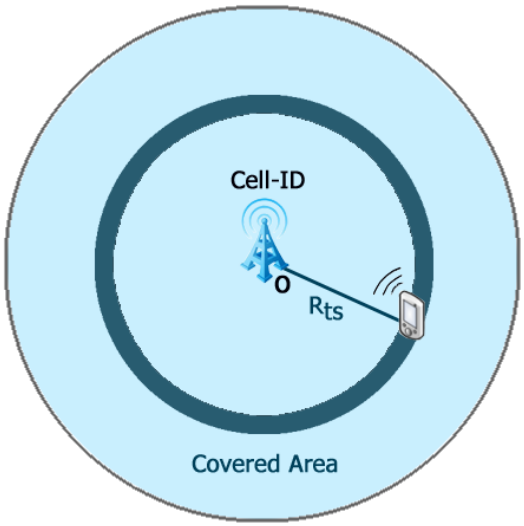
1.3.2.1 Localizzazione Cell-based

La soluzione più semplice al problema del posizionamento via rete cellulare è la cosiddetta *Cell-ID positioning*, alternativamente nota come metodo COO (*Cell Of Origin*) [Inc02]. L'approccio considerato prevede che la posizione corrente del dispositivo mobile sia la stessa di quella associata alla stazione radio base a cui il terminale è agganciato. Quanto appena affermato discende dall'ipotesi che il terminale sia agganciato alla cella col più forte segnale rilevato che di norma è anche quella a minor distanza dal dispositivo. Tuttavia, anche in condizioni ottimali, la precisione è comunque relativa alla distanza con cui sono disposte le stazioni base all'interno della LAC. Esistono zone ben servite in cui la densità e la distanza fra le celle è piccola, come negli agglomerati urbani, viceversa alcune aree rurali o secondarie prevedono poche stazioni, anche poste a grandi distanza le une dalle altre. Nel primo caso la precisione ottenibile è di qualche centinaio di metri, nel secondo si hanno stime con accuratezza assai inferiore, nell'ordine del chilometro.

Quando un utente è agganciato a una cella con un determinato Cell-ID, è possibile approssimare la sua posizione con quella della stazione base.



(a) Posizione in O e incertezza Covered Area



(b) Posizione in O e incertezza Rts

Figura 1.18: Localizzazione Cell-based

In figura 1.18(a) si può far riferimento all'origine *O* del cerchio azzurro che idealmente rappresenta l'area di copertura della BTS. L'intera area di copertura (*Covered Area*) indica l'incertezza del posizionamento. Per raffinare questa tecnica è possibile tenere in considerazione le informazioni

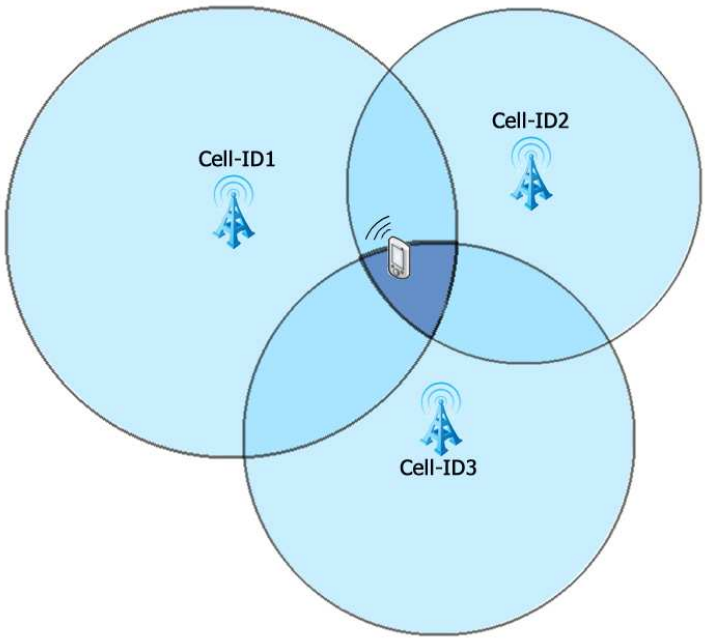
temporali (*timestamp*) che il dispositivo e la cella si scambiano al momento in cui accadono determinati eventi di sistema utili a realizzare il servizio telefonico. Come indicato in figura 1.18(b), ciò consente di individuare con maggior precisione l'ubicazione del dispositivo posizionandolo su una corona circolare (in celeste scuro) che nel caso ideale individua una circonferenza. In questo caso, l'incertezza della localizzazione diminuisce e non è più l'intera area di copertura della cella, ma si restringe sul cerchio di raggio *Rts*. Ulteriori raffinamenti si hanno quando le aree di copertura delle celle non sono ottenute per mezzo di antenne omnidirezionali come quelle considerate, ma da un insieme di antenne direttive.

Oltre alla già citata impredicibile e minore accuratezza rispetto al sistema GPS, il metodo COO si basa su un assunto non sempre rispettato: non è detto che la cella agganciata dal dispositivo mobile sia quella a potenza maggiore e anche qualora lo fosse, non è necessariamente vero che quest'ultima sia la più vicina. Inoltre, apprendere i valori di LAC e Cell-ID (espressi con numeri decimali o esadecimali) non significa ottenere le coordinate geografiche in latitudine, longitudine e altitudine: allo scopo occorre ricorrere a opportune elaborazioni.

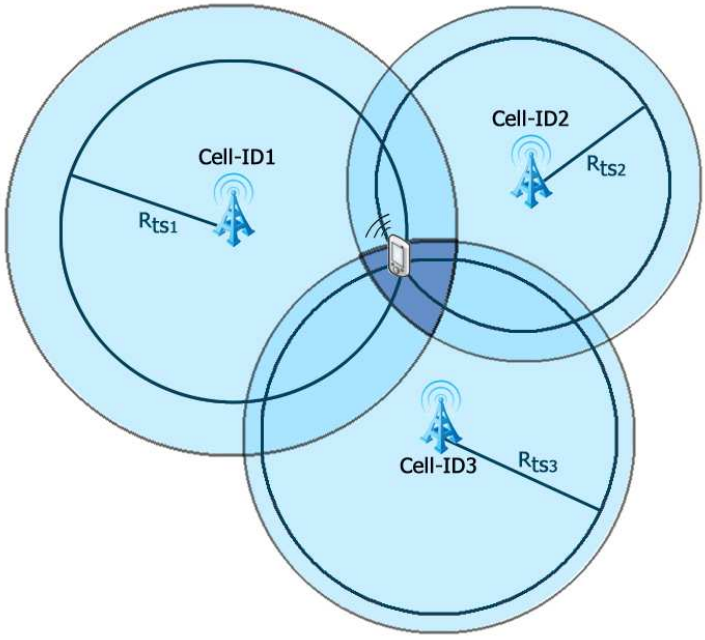
1.3.2.2 Localizzazione Triangulation-based

Un altro approccio comunemente utilizzato per risolvere il problema della localizzazione si basa sull'idea della triangolazione. Estendendo quanto descritto nel paragrafo precedente, è anche possibile prendere in considerazione un maggior numero di BTS, nel qual caso la triangolazione dei dati forniti può consentire una più corretta ipotesi sulla posizione del cellulare.

Il metodo prevede di usare una funzione capace di restituire la distanza da ciascuna cella visibile dal dispositivo mobile, in base alla misurazione della potenza dei segnali ricevuti. Come rappresentato in figura 1.19(a), quando la distanza di almeno tre celle viene determinata, l'intersezione dei relativi cerchi incentrati nelle celle permette d'individuare la posizione corrente del terminale (area di colore celeste scuro). Anche in questo caso,



(a) Posizione nell'area intersezione delle tre Covered Area



(b) Posizione all'intersezione delle tre circonferenze di raggio Rts1, Rts2, Rts3

Figura 1.19: Localizzazione Triangulation-based

se vengono prese in esame le informazioni temporali scambiate con le varie stazioni base (timestamp), la localizzazione può essere più precisa. In figura 1.19(b) è possibile notare come il dispositivo mobile all'interno dell'area frutto dell'intersezione delle Covered Area è posizionato all'incrocio delle tre circonferenze rispettivamente di raggio *Rts1*, *Rts2*, *Rts3*.

La localizzazione per mezzo della triangolazione è sicuramente più precisa di quella ottenibile con l'approccio Cell-based, ma possono esserci dei seri problemi quando anche una sola delle stime della distanza dalle stazioni radio base è di scarsa qualità. Le misurazioni del segnale sono intrinsecamente rumorose a causa di svariati fattori e pertanto i cerchi disegnati intorno alle BTS potrebbero non intersecarsi, nel quel caso il metodo non produrrebbe alcun risultato utile. L'approccio fallisce anche se non si riescono ad osservare almeno tre celle e ciò può accadere quando ci si trova in luoghi con ingombranti ostacoli ambientali.

Anche l'approccio Triangulation-based prevede la conoscenza delle informazioni relative alla posizione delle stazioni radio base, altrimenti è impossibile realizzare la localizzazione.

1.3.3 Localizzazione via Rete Wi-Fi

Un altro tipo di localizzazione possibile per i dispositivi mobili dotati di connessione Wi-Fi si fonda sulla presenza e la mappatura degli *Access Point* (AP). Rilevati gli AP nei paraggi del dispositivo mobile e conoscendone la posizione, è possibile localizzare il terminale tipicamente con una precisione di alcune decine di metri. Le coordinate geografiche non sono direttamente fornite dagli Access Point, progettati per altri scopi, ma accedendo ad appositi database a cui vengono trasmesse alcune informazioni da essi ricavate, quali il *SSID* e il *MAC address*. Questo tipo di servizio è offerto da alcuni fornitori commerciali come ad esempio Google Inc., a volte in modo gratuito.

Localizzare un dispositivo tramite rete Wi-Fi non presenta particolari svantaggi o limitazioni tecniche quanto piuttosto problemi legati alla privacy e alle legislazioni dei singoli paesi. A titolo di esempio, Google Inc.

è stata criticata dall'organo tedesco per la protezione dei dati a causa dell'utilizzo delle auto di *Street View* (figura 1.20) per la scansione delle reti wireless.

L'azienda ha pertanto pubblicato una risposta dettagliata che spiega come i dati vengono raccolti e le finalità per cui sono impiegati. Nel documento [Lei10] si sostiene che l'acquisizione delle informazioni di rete Wi-Fi non è illegale e molte altre aziende li collezionano per gli stessi fini: migliorare e realizzare sistemi di localizzazione.

Senza entrare nel merito non tecnico della questione, si vuole unicamente sottolineare che il posizionamento dei dispositivi mobili via rete Wi-Fi potrebbe essere sospeso in alcuni paesi o sottoposto a divieto, con conseguente perdita di generalità del metodo.

Figura 1.20: GoogleCar munita di antenna omnidirezionale Maxrad BMMG24005

1.4 Conclusioni

In questo capitolo abbiamo descritto come i semplici cellulari si siano rapidamente evoluti negli smartphone, dispositivi sempre più potenti e versatili. Si è anche osservato come la diffusione delle app sia aumentata a dismisura raggiungendo un buon grado di maturità pur rimanendo semplici da usare e dunque accessibili alle masse.

In seguito ci siamo occupati di Android, uno dei più importanti sistemi operativi per dispositivi mobili presente sulla scena del mercato contemporaneo. Ciò si giustifica poiché il sistema è aperto e gratuito come pure gli strumenti di sviluppo forniti. La diffusione delle app è libera, ma è anche possibile utilizzare un canale ufficiale di distribuzione poco costoso, rapido e di semplice utilizzo. Del sistema operativo mobile sono stati descritti brevemente i componenti principali utili alla realizzazione delle applicazioni e il ciclo di vita di una activity. Sempre sul suo modello di concorrenza è stato introdotto il concetto di task e l'amministrazione della chiusura delle attività.

Un altro argomento trattato è il servizio remoto: dopo averne fornito una definizione comune e alcuni esempi tipici, si è fatto cenno all'architettura Internet alla base dei servizi web. Descritto il modello Cliente/-Servitore, ottimo candidato per realizzare di tali servizi, viene introdotta l'architettura SOA e la sua incarnazione in Web Services, nonché alcuni dei suoi protocolli caratteristici: WSDL, UDDI e SOAP.

Le applicazioni che operano in base alla posizione (location-aware) devono recuperare tale informazione: a fine capitolo sono state introdotte le coordinate geografiche (latitudine, longitudine, altitudine) e il relativo sistema di riferimento. A seguire è stata presentata una panoramica sulle tecnologie di localizzazione, in particolare quelle usabili dalla maggior parte degli smartphone: GPS, network (Cell-based/Triangulation-based) e Wi-Fi.

Capitolo 2

Generalità sull'anticontraffazione

Per *anticontraffazione* s'intende l'insieme delle metodologie, degli strumenti e delle tecnologie sviluppate al fine di combattere il fenomeno della contraffazione, piaga del libero mercato e responsabile del danneggiamento tanto del produttore quanto del consumatore.

Il significato proprio del termine *contraffazione* è riconducibile all'attività di chi riproduce qualcosa in modo tale che possa essere scambiata per l'originale, violando il diritto di proprietà intellettuale e/o industriale: marchi d'impresa e altri segni distintivi, brevetti per invenzione, modelli di utilità, industrial design, indicazioni geografiche, denominazioni di origine, diritti d'autore, prodotti in generale. Le modalità con cui il fenomeno illecito si manifesta sono molto varie e altrettanto diverse sono le conseguenze non solo sul piano giuridico, ma anche su quello economico e sociale.

Con il termine contraffazione, intesa nella sua accezione più ampia, si riferisce una serie di fenomenologie essenzialmente riconducibili alla:

- produzione e commercializzazione di merci che recano - illecitamente - un marchio identico a un marchio registrato;
- produzione di beni che costituiscono riproduzioni illecite di prodotti coperti da copyright, attività meglio nota col nome di "pirateria".

In questo capitolo si vuole porre l'attenzione sulle tecnologie più comuni ideate per realizzare sistemi di anticontraffazione, per poi presen-

tare l'idea di *impronta digitale vIDfy* sviluppata dall'azienda *ViDiTrust* e descritta nei paragrafi 2.3 e 2.3.1 a partire da pagina 71.

Prima di procedere in tal senso, si rende necessaria una premessa relativa al concetto di identità (digitale). Solo per mezzo dell'identità e del suo riconoscimento è possibile esprimersi sull'autenticità, ovvero su ciò che è autentico, genuino, ossia che non è falso o falsificato e che può dimostrarsi o imporsi come vero.

2.1 Identità digitale

Il concetto di identità assume diversi significati in vari campi, ma come scopo ha quello di permettere l'identificazione in modo univoco di una certa entità fisica o astratta: una persona, una risorsa in Internet, un dispositivo, un prodotto.

Nello specifico, il significato più comune attribuito all'idea di identità digitale è quello dell'insieme delle informazioni riguardanti una persona fisica; oltre ai dati puramente anagrafici, le nuove tecnologie forniscono una serie di altre informazioni che vanno a costituire un profilo digitale dinamico e "fluido" da affiancare al concetto classico di identità. L'ammontare delle informazioni hanno due origini principali:

- dati immagazzinati su sistemi e archivi digitalizzati da parte della pubblica amministrazione, come ad esempio le anagrafi dei comuni, l'Agenzia delle entrate, il Servizio sanitario nazionale;
- dati che i soggetti cedono volontariamente a sistemi digitali di vario genere: social network, compilazione di formulari online.

Realizzare una identità e manutenerla è un processo complesso e che presenta diversi problemi non banali. Innanzi tutto, salvo un piccolo insieme di campi informativi comuni, la maggior parte dei dati da aggregare sono eterogenei, relativi agli specifici soggetti e legati alla loro storia, in continua evoluzione. Inoltre, le stesse informazioni sono spesso non condivise, ma duplicate in raccolte afferenti a categorie distinte la cui gestione è affidata ad autorità o enti separati: difficilmente è possibile garantire

l'integrità dei dati. A titolo di esempio consideriamo che, al momento dell'acquisto di una autovettura, il nominativo e altri dati dell'acquirente presenti presso l'Ufficio Anagrafe verranno integrati anche nell'archivio del Pubblico Registro Automobilistico (PRA). Comprare una casa, usare servizi bancari quali un conto corrente o il bancomat, l'acquisto di un cellulare e la sottoscrizione di un abbonamento, prevede la comunicazione dei dati a svariati uffici e associazioni, come il Catasto, la banca, l'operatore telefonico.

L'esigenza di unificare le informazioni e proteggerle sono divenute nel tempo sempre più pressanti ma di difficile realizzazione tecnica, almeno in tempi brevi. Al momento, la maggior parte degli archivi dei dati su cui si costruiscono le identità digitali non sono interoperabili e neanche compatibili tra loro, salvo alcune eccezioni, in cui sono interni a uno stesso ente, spesso pubblico.

Tralasciando l'aspetto *social* del Web 2.0, i relativi profili online e la crescente preoccupazione per il trattamento e la conservazione dei dati personali, riportiamo alcuni esempi pratici di realizzazione di identità digitali: per un cittadino, per una entità in rete, per un dispositivo mobile.

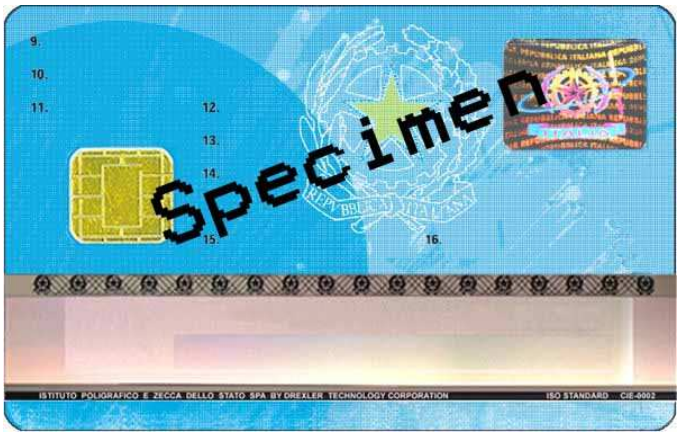
2.1.1 Carta d'identità elettronica (CIDE)

La Carta d'Identità Elettronica della Repubblica Italiana (CIDE) è una *smart card* che integra nel supporto in policarbonato una banda ottica e un microprocessore. Più specificamente, i dati del titolare, compresa la foto, sono impressi in modo visibile sia sul supporto fisico, per l'identificazione "a vista", che sulla banda ottica e poi memorizzati informaticamente sul microchip e ancora sulla banda ottica.

In figura 2.1 è riportata la CIDE che riassume in modo oggettivo le esigenze e i problemi che abbiamo discusso nel precedente paragrafo. Essa vuole realizzare diversi obiettivi:

- la completa interoperabilità su tutto il territorio nazionale;
- la maggiore sicurezza nel processo d'identificazione ai fini di polizia;
- l'utilizzo quale strumento d'identificazione in rete per servizi online.

(a) CIDE - fronte



(b) CIDE - retro

Figura 2.1: Carta d'Identità Elettronica della Repubblica Italiana

Nel tentativo di aggregare le informazioni personali in un unico documento, la nuova carta include diversi altri campi, uno di essi (il 14) è dedicato al codice fiscale, non previsto sul previgente supporto cartaceo. Tale codice è esplicitamente pensato per realizzare un identificativo unico (anche se possono verificarsi omocodie) e pertanto è bene che sia facilmente accessibile.

Il microchip integrato è predisposto per contenere numerose altre indicazioni oltre ai certificati di firma digitale. I dati supplementari immessi dalle pubbliche amministrazioni sono utili per accedere ai servizi qualificati da queste implementati e possono essere il gruppo sanguigno, le opzioni di carattere sanitario, le caratteristiche biometriche. Il supporto

elettronico permette anche il riconoscimento in rete del titolare ed è utile alle negoziazioni transazionali tra chi richiede il servizio e chi lo eroga. I servizi che prevedono la memorizzazione di dati sulla CIDE possono essere comunali o nazionali. I primi sono predisposti in piena autonomia dai comuni, per quelli nazionali è necessaria un'autorizzazione da parte del Dipartimento della Funzione Pubblica [dU09].

La carta d'identità elettronica è dunque uno strumento di identificazione personale nonché di *autenticazione* per l'accesso ai servizi web offerti dalle Pubbliche Amministrazioni. La sperimentazione del documento è stata avviata nel 2000 per iniziativa del Ministero dell'Interno in collaborazione con 156 Comuni [dI12].

La diffusione del certificato elettronico è aumentata, ma a oggi (febbraio 2012), esso è in possesso solo di una minoranza di cittadini, a dimostrazione che la fusione degli archivi digitali, la manutenzione dell'integrità dei dati e il processo di migrazione sono operazioni complesse e laboriose.

2.1.2 Digital Object Identifier (DOI)

Con altre finalità, ma pur sempre per creare un'identità univoca, nasce il DOI® (*Digital Object Identifier*) [Fou12]. Figlio di esigenze editoriali, la sua portata e i suoi obiettivi sono cresciuti nel tempo, fino ad assurgere al grado di standard ISO/DIS 26324 (via ISO TC46/SC9), approvato nel novembre 2010.

Nel mondo della stampa tradizionale, il modo di esprimere le identità ricorre agli standard ISBN, ISSN e ISMN che però risultano essere troppo legati alle caratteristiche delle pubblicazioni su carta.

Il codice ISBN (*International Standard Book Number*) è un codice numerico usato internazionalmente per la classificazione dei libri. È definito da uno standard ISO e permette di identificare in modo univoco ogni specifica edizione di un certo libro, ma non le diverse ristampe. I codici ISSN (*International Standard Serial Number*) e ISMN (*International Standard Music Number*) hanno significati analoghi e sono sempre standard ISO, ma

vengono utilizzati rispettivamente per la classificazione di pubblicazioni periodiche come riviste e quotidiani e per gli spartiti musicali.

Il DOI System è invece utilizzato per identificare gli oggetti presenti all'interno di una rete digitale e l'insieme dei suoi nomi è assegnato a qualsiasi entità ivi definibile [Fou12, Taj05]. Tali nomi hanno la finalità di fornire informazioni sempre aggiornate, anche qualora fossero rintracciabili su Internet in modo diretto o ricavabili da metadati.

L'obiettivo dello standard è quello di realizzare un'identificazione duratura di qualsiasi entità che sia oggetto di proprietà intellettuale: mentre le informazioni su un oggetto digitale possono variare nel tempo, come pure la sua localizzazione all'interno della rete, il nome DOI non cambierà. I *DOI Name* possono essere usati per qualsiasi forma di gestione dati, commerciale e non.

Le principali caratteristiche del DOI System sono le seguenti:

- *unicità*: il nome DOI esprime una identità e pertanto deve essere unico;
- *persistenza*: il nome DOI continuerà a funzionare correttamente nonostante venga modificata l'ubicazione del materiale, venga riordinato o inserito nei "preferiti" di un browser web;
- *indipendenza dalla piattaforma*: il DOI System è indipendente dall'architettura degli elaboratori e dai sistemi operativi su di essi installati;
- *aggiornamenti dinamici*: cambiare i metadati o fare aggiornamenti di applicazioni e servizi non prevede l'interruzione della fruizione degli stessi;
- *estensibilità*: il nome DOI può essere esteso, aggiungendovi nuove caratteristiche e servizi attraverso l'amministrazione dei *Gruppi dei DOI Name*;
- *cooperazione* con altri dati, provenienti da altre fonti, a vantaggio della realizzazione dei servizi.

Il sistema di numerazione DOI Name segue una sintassi standardizzata (ANSI/NISO Z39.84 – 2005) e prevede l'uso di *stringhe opache*, ovvero che non esprimono nulla di significativo riguardo l'entità che hanno come oggetto. L'unico modo per ottenere le informazioni a essa associate è quello di analizzarne i metadati. Come osservato, modificare i metadati o qualche altra proprietà dell'entità non comporta un cambiamento del DOI Name (persistenza).

Il DOI Name è composto da due componenti separati da una barra, come indicato in figura 2.2.

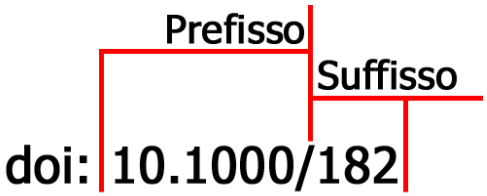


Figura 2.2: DOI Name: prefisso/suffisso

1. Prefisso: tutti i nomi DOI iniziano con "10.", questo per distinguere un nome DOI da un'altra qualsiasi implementazione del *Handle System* [Sys12]. La stringa seguente esprime l'identificativo dell'organizzazione che ha registrato l'elemento e può essere una combinazione qualsiasi di caratteri alfanumerici. Ogni organizzazione può avere più di un identificativo (non è previsto un limite massimo) ed è consentita una suddivisione in prefissi secondari.
2. Suffisso: insieme al prefisso forma una stringa unica (unicità del DOI Name) e anch'esso è realizzato per mezzo di caratteri alfanumerici. Esistono due modi per creare un suffisso, dipende se le entità sono già numerate in qualche maniera oppure non lo sono. Nel primo caso, la numerazione può utilizzare schemi standard (per esempio il codice ISBN dei libri) oppure un sistema di classificazione interno previsto dall'ente registrante. Tali schemi possono dunque essere usati per formare il suffisso.

Il DOI Name non prevede un limite alla lunghezza delle due parti e ciò permette di averne un'ampia disponibilità.

2.1.3 International Mobile Equipment Identity (IMEI)

Un altro esempio di identificazione che si vuole brevemente citare è quello riguardante il mondo dei dispositivi mobili: cellulari, smartphone, telefoni satellitari. Essa si realizza per mezzo dell'*International Mobile Equipment Identity* (IMEI), un codice numerico a 15 cifre. Solitamente è stampato all'interno del vano batteria dei vari dispositivi e può essere visualizzato sullo schermo inserendo da tastiera la stringa " *# 06 #".

Il numero IMEI viene utilizzato dagli operatori della rete GSM per identificare i dispositivi che vi si connettono con lo scopo di bloccare i telefoni rubati. In caso di furto, il proprietario è invitato a comunicare il codice IMEI al proprio provider o alle autorità di pubblica sicurezza a mezzo denuncia che provvederanno a inserirlo su un'apposita "lista nera". I dispositivi iscritti su tale lista possono essere esclusi dall'accesso ai servizi di telefonia mobile e pertanto divenire inservibili.

L'IMEI viene impiegato solo per identificare il dispositivo e non ha alcuna relazione permanente o semi-permanente con l'abbonato. Associazioni più stringenti possono essere realizzate per altre vie, ad esempio tramite il codice IMSI che è memorizzato sulla scheda SIM rilasciata dagli operatori ai propri abbonati dietro la sottoscrizione di un contratto.

Osservando la figura 2.3 è possibile individuarne la sua struttura. Il codice composto da 15 cifre è suddiviso in quattro parti (AAAAAA – BB – CCCCCC D), il cui significato è illustrato qui di seguito.

- AAAAAA: le prime sei cifre rappresentano il *Type Approval Code* (TAC) che identifica la casa costruttrice e il modello del dispositivo mobile;
- BB: le seguenti due cifre indicano il *Final Assembly Code* (FAC) che individua il luogo di costruzione o di assemblaggio del prodotto;
- CCCCCC: le ulteriori sei cifre identificano il numero di serie del cellulare;
- D: l'ultima cifra è detta *Check Digit* (CD) o *Spare* (SP) ed utile per verificare la correttezza del codice IMEI.

Figura 2.3: Codice IMEI riportato nel vano batteria di un cellulare

Nel tempo sono state apportate alcune modifiche al codice: dall'aprile del 2004, il FAC è stato eliminato e il suo valore fissato a "00"; la sigla TAC ha cambiato significato ed è divenuta acronimo di *Type Allocation Code*. Esistono infine IMEI a 16 cifre in cui il Check Digit è sostituito con il campo *Software Version Number* (SVN) a due cifre che indica la versione dell'applicativo installato sull'apparecchio.

Il codice IMEI è un buon deterrente contro i furti, ma il sistema con cui vengono attuati i controlli presenta delle lacune. La realizzazione delle *black list* è responsabilità dell'operatore e seppure il codice risulta essere internazionale, le liste spesso restano "locali" essendo di rado condivise. Un dispositivo mobile a cui è stato bloccato l'IMEI da un certo operatore potrebbe continuare a funzionare sotto un altro provider con una SIM differente, anche all'interno dello stesso paese. Tale situazione è generalmente sempre verificata quando il cellulare rubato e bloccato viene esportato all'estero.

Infine, seppure l'IMEI è pensato per realizzare un sistema di identificazione internazionale e dunque dovrebbe essere unico, in realtà non lo è. Contrariamente a quanto si ritiene, possono esistere due o più dispositivi

mobili con lo stesso identificativo: in Internet sono rintracciabili svariate applicazioni capaci di modificarlo, operazione illegale nella maggior parte degli stati. Procurarsi un nuovo IMEI valido è ancora più semplice poiché è sufficiente usare il codice di un altro dispositivo non rubato. Gli operatori inglesi BT Cellnet e Vodafone, ad esempio, dichiarano che circa il 10% dei codici che circola sulle loro reti è costituito da doppioni [Bra02].

2.2 Tecnologie di anticontraffazione

Definito ed esemplificato il concetto di identità digitale, utile a distinguere fra le varie entità di un certo insieme, è fondamentale poter garantire l'attendibilità della stessa. Per *autenticazione* s'intende quel settore che studia il modo di certificare l'autenticità di un dato oggetto, fisico o digitale. Dipendentemente dal tipo di elemento da validare e dalla natura realizzativa della sua identità, sono previsti sistemi e strumenti di autenticazione dedicati che si distinguono a livello metodologico e tecnologico. Nonostante la grande varietà dei casi, i processi di autenticazione possono essere suddivisi in due macro-categorie: attivi e passivi. I sistemi attivi intervengono a monte, direttamente nel processo di produzione dell'oggetto da autenticare, al contrario, i passivi, recuperano le informazioni necessarie alla validazione del prodotto a posteriori della sua realizzazione e ciò comporta l'abbattimento dei costi.

Limitatamente all'autenticazione di package e documenti stampati, oggetto di tesi, è utile introdurre un insieme ristretto di tecnologie comunemente utilizzate in questi campi.

2.2.1 Stato dell'arte: sicurezza della cartamoneta

Come linea guida e per esperienza diretta del lettore, verranno brevemente esaminate le principali caratteristiche di sicurezza adottate per le banconote in uso nella Comunità Europea. Per ovvi motivi, la lista completa di tali caratteristiche è coperta da segreto, ma la Banca Centrale Eu-

ropea (BCE) ne ha descritto alcune fra le più evidenti, al fine di consentire al pubblico di distinguere le banconote autentiche dalle false.

Presentiamo dapprima in forma di elenco, a seguire a mezzo di una breve descrizione, alcuni degli elementi di sicurezza adottati [Wik12]:

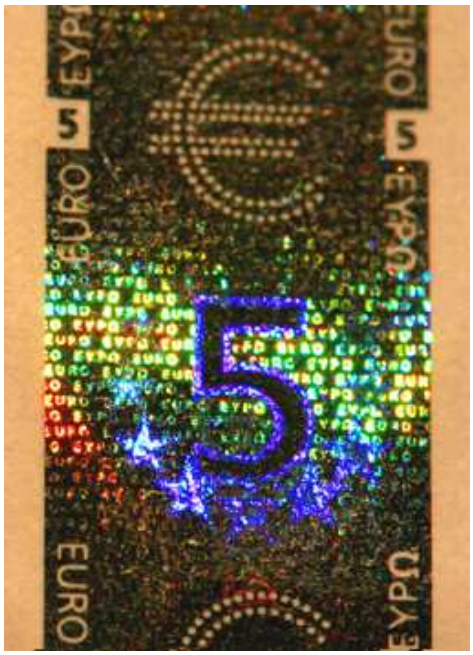
- ologrammi;
- inchiostro cangiante;
- striscia iridescente;
- cifra di controllo del numero di serie;
- EURion constellation;
- filigrana;
- inchiostro magnetico, registro di stampa, stampa in rilevo, filo di sicurezza, codice a barre...

Nelle banconote di taglio 5€, 10€ e 20€ si ha una *banda olografica* sul lato destro della faccia frontale: questa banda viene stampata col valore nominale della banconota ripetuto più volte (figura 2.4(a)). Per le banconote di taglio maggiore (50€, 100€, 200€ e 500€), al posto della banda viene impressa una placchetta olografica.

Al centro del retro delle banconote da 5€, 10€e 20€ è impressa una *striscia iridescente* che riporta anche in questo caso il valore nominale della banconota ripetuto più volte. La striscia è ben visibile se si inclina la banconota sotto una fonte luminosa.

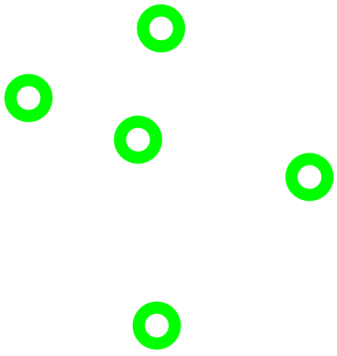
Sul retro delle banconote di taglio 50€, 100€, 200€ e 500€, il valore nominale nell'angolo in basso a destra è stampato con un *inchiostro cangiante* (*otticamente variabile*), ossia che ha la capacità di cambiare colore a seconda dell'inclinazione. Di fatto, osservando la banconota con angolazioni differenti, il colore delle cifre muta dal viola al marrone, dal marrone al verde oliva e viceversa (figura 2.4(b)).

Ciascuna banconota ha un proprio *numero seriale*. A ogni nazione della comunità europea emettitrice di cartamoneta è associata una cifra di



(a) Banda olografica (5€)

(b) Inchiostro cangiante (50€)



(c) EURion constellation

(d) Banconota da 100€ sottoposta a raggi UV

Figura 2.4: Elementi di sicurezza delle banconote della Comunità Europea

controllo che consente di verificare l'autenticità della banconota attraverso una semplice procedura di calcolo, omessa in questa sede (per i dettagli si rimanda a [Wik12]).

Le banconote della Comunità Europea contengono sul loro sfondo un motivo noto col nome di *EURion constellation* e basato sul disegno formato dalla posizione delle stelle della costellazione di Orione (figura 2.4(c)). Il disegno è stato aggiunto per aiutare i software a rilevare la presenza di una banconota in un'immagine digitale e dunque impedirne l'acquisizione.

Vengono inoltre utilizzati vari tipi di filigrana:

- *standard*: ciascuna banconota è stampata con una filigrana che raffigura un particolare dello stile architettonico su di essa riportato e il suo valore nominale. La filigrana è visibile ponendo la banconota in controluce;
- *digitale*: similmente all'EURion constellation, nel disegno delle banconote è integrata anche una filigrana digitale. Anche in questo caso i programmi di fotoritocco e di elaborazione grafica rifiutano l'acquisizione dell'immagine dopo aver rilevato tale filigrana;
- *infrarossa* e *ultravioletta*: se esposte ai raggi infrarossi, le banconote originali mostrano aree più scure in zone differenti a seconda del valore nominale. Esposte invece ai raggi ultravioletti, l'EURion constellation e alcuni altri dettagli vengono messi in risalto con un forte contrasto risultando fluorescenti (figura 2.4(d)).

Infine, esistono molte altre caratteristiche di sicurezza implementate nella produzione delle banconote, per esempio la realizzazione di stampe e micro-stampe ad altissima risoluzione e l'utilizzo di materiali riservati non disponibili sul mercato.

In base a quanto detto, i numerosi sistemi di validazione risultano essere di tipo attivo poiché agiscono direttamente sull'oggetto da autenticare e devono essere congiuntamente verificati al fine di poter validare l'autenticità della banconota.

In conclusione, risulta arduo se non impossibile riprodurre fedelmente tutte le caratteristiche di sicurezza sopra descritte. Anche se le caratteristiche di sicurezza introdotte nella produzione di banconote è pressoché la norma per la sua salvaguardia della cartamoneta, gli elevati costi di realizzazione e produzione propri di una Zecca di Stato non possono essere sostenuti per la messa in sicurezza di prodotti comuni o di valore minore.

2.2.2 Digital Watermarking

L'alternativa economica per ciò che riguarda l'autenticazione delle stampe è il *digital watermarking*.

Il termine digital watermarking letteralmente significa "filigrana digitale". Come la filigrana è uno dei sistemi adottati per assicurare l'autenticità e l'integrità delle banconote (vedi paragrafo precedente), così i watermark devono garantire l'autenticità e l'integrità dei documenti digitali in cui sono inseriti. Per documenti digitali è lecito intendere elementi testuali, grafici, audio o video.

L'idea di base di questa tecnologia è di inserire all'interno del documento da proteggere una sequenza di bit (W) detta *watermark* che permette di verificarne l'autenticità.

In estrema sintesi, uno schema di watermarking stabilisce due cose:

- come la sequenza W deve essere inserita nel documento;
- come la sequenza W deve essere recuperata.

Il digital watermarking discende degli *studi steganografici* che hanno lo scopo di incapsulare un messaggio segreto, quale potrebbe essere un copyright o un numero seriale, in un cosiddetto messaggio di copertura [Mon05, BP09]. L'inserimento di tali informazioni è tipicamente parametrizzato da una chiave, senza la conoscenza della quale è "impossibile" o almeno molto laborioso, rimuovere o riconoscere l'elemento incapsulato. L'approccio steganografico prevede che il documento utilizzato per contenere il messaggio segreto sia soltanto una maschera senza particolare valore; la corrispondenza è completamente invertita nel caso del digital watermark. In effetti, il messaggio nascosto che funge da protezione non ha alcun valore per l'utilizzatore finale, mentre è il documento ad assumere il ruolo di protagonista [DSG09].

Utilizzare il digital watermarking a vantaggio dell'autenticazione delle stampe, prevede di agire attivamente sulla versione "digitale" delle stesse, alterandone impercettibilmente l'immagine prima che venga stampata, in accordo con l'informazione autenticante che deve esservi inserita.

Relativamente all'ambito immagine/stampa, gli schemi watermarking che sono stati sviluppati negli ultimi anni possono essere classificati come segue:

- *visibili* e *invisibili*;
- *fragili*, *semifragili* e *robusti*;
- *pubblici* e *privati*.

I watermark *visibili* sono costituiti da elementi visivi semitrasparenti sovrapposti alle immagini principali. Generalmente consistono in loghi o marchi delle organizzazioni che detengono i diritti delle immagini da proteggere e sono realizzati in modo tale da essere facilmente individuabili dall'osservatore. In questi casi il watermark è fuso con l'immagine e non può essere estratto. Il principale vantaggio di tali watermark è la capacità nello scoraggiare l'uso illegale delle immagini. Un esempio che si può riportare è quello del mondo televisivo: ciascuna emittente pone il proprio logo in un angolo dell'immagine principale per "firmare" il materiale mandato in onda (riquadro verde in figura 2.5).

Figura 2.5: Watermark visibile, logo nell'angolo in basso a destra

I watermark *invisibili* invece, non sono percettibili dall'occhio umano sotto le normali condizioni visive e risultano maggiormente d'aiuto nell'individuare e perseguire, piuttosto che nello scoraggiare un eventuale

ladro. Sempre costituiti da un'immagine sovrimpressa, possono essere individuati solo processando algoritmicamente l'elemento grafico.

Un watermark è detto *fragile* se viene distrutto e reso irriconoscibile quando l'immagine digitale subisce una qualsiasi manipolazione dei dati. Esso è concepito per quelle applicazioni in cui si desidera sapere se una certa informazione è stata modificata nel passaggio dal creatore all'utilizzatore, nel qual caso il watermark non deve essere rilevabile o, comunque, deve presentare alterazioni. In questi casi diviene semplice determinare se un'immagine si presenta ancora intatta o se invece ha subito modifiche.

I watermark *semifragili* sono progettati in modo da andare distrutti in seguito a qualsiasi cambiamento che superi una certa soglia specificata da chi lo inserisce: una soglia zero individua perciò un watermark fragile [DSG09].

Un watermark *robusto* è pensato per resistere alle più comuni operazioni di trasformazioni sui dati ed è generalmente utilizzato quando occorre provare la proprietà dell'immagine, per esempio a mezzo di informazioni relative al copyright. L'informazione che trasporta deve poter essere recuperata anche se l'elemento grafico viene modificato e se necessario, far fronte ad attacchi intenzionali volti alla sua rimozione.

Un watermark *privato* può essere estratto solo se è noto a priori il suo contenuto e si è in possesso di un documento originale non marcato. Viceversa, un watermark *pubblico* è rilevabile anche se il suo contenuto non è noto e si è sprovvisti il documento originario: è più semplice da identificare, alterare o rimuovere, ma è utile per individuare il proprietario del documento.

Operare per mezzo del digital watermarking permette solo a chi possiede i diritti necessari per autenticare anche la facoltà di modificare opportunamente la geometria della stampa, mentre chiunque può avere la possibilità di verificarne l'autenticità. Naturalmente, durante l'inserimento dell'informazione autenticante, il digital watermarking deve poter introdurre alterazioni sufficientemente percettibili affinché siano robuste al processo di stampa, ma non troppo invasive da degradare la qualità dell'immagine. Inoltre, la robustezza deve essere sufficiente an-

che per sopportare il successivo passo di ri-digitalizzazione della stampa (acquisizione via scanner), necessario all'attuazione di una verifica di autenticità.

Il maggior punto debole di questa tecnologia consiste proprio nella corretta stima e implementazione del grado di robustezza. Qualora l'immagine stampata non sia sufficientemente robusta alle distorsioni e alle alterazioni introdotte dal processo di stampa, allora si incorre facilmente in un falso positivo, ovvero l'elemento grafico risulterà non autentico quando invece lo è. Al contrario, se la robustezza è troppo marcata, pur restando contenuta a livello ottico in modo da non degradare l'immagine, il sistema di autenticazione basato sul watermarking potrebbe dar luogo a falsi negativi: il prodotto digitalizzato e nuovamente stampato risulta alla verifica come autentico, nonostante sia una copia o una riproduzione non autorizzata. In uno scenario applicativo, a un contraffattore basterebbe venire in possesso di una stampa, un documento o un package originale per essere in grado di riprodurne di "autentici" al fine di realizzare prodotti contraffatti.

Questo limite di robustezza fra una stampa-acquisizione e un processo di stampa-acquisizione-stampa-acquisizione è molto difficile da ottenere con l'utilizzo di tecniche di digital watermarking, se non in specifici contesti sperimentali.

2.3 Tecnologia ViDiTrust

Quanto discusso nei precedenti paragrafi illustra l'interesse nello sviluppare un sistema affidabile e dai costi contenuti, utile all'autenticazione di package e documenti stampati. In tal senso opera l'azienda ViDiTrust [ViD10] che propone una soluzione di autenticazione dei supporti stampati applicabile alle attuali tecnologie di stampa più comuni e diffuse.

L'innovatività della tecnologia proposta per l'autenticazione e l'anticontraffazione dei prodotti consiste essenzialmente nella sua flessibilità e facilità di integrazione in un qualunque processo di stampa, senza la necessità di intervenire sulla sua meccanica, sull'inchiostro usato e senza

l'introduzione di ulteriori fasi di produzione o macchinari specializzati. Anche il processo di verifica e validazione dell'autenticità prevede l'uso di strumenti economici e sempre più diffusi quali scanner di tipo consumer e smartphone.

L'idea imprenditoriale realizzata da ViDiTrust si basa sullo sviluppo di una tecnologia passiva che, non intervenendo a monte del processo di stampa, valida oggetti già stampati. L'autenticazione avviene pertanto solo conoscendo o "leggendo" le caratteristiche statistiche proprie della particolare macchina che ha prodotto la riproduzione. Al contrario di quanto ravvisato per il digital watermarking, il modo di operare appena descritto non altera la qualità di stampa e l'ottenere il sorgente digitale o un'accurata digitalizzazione del supporto che si vuole copiare, non è sufficiente a produrre stampe contraffatte che risultino autentiche. Chi vuole produrre autentiche stampe contraffatte deve anche aver accesso alle stesse apparecchiature che producono le stampe autentiche, ma questo scenario di contraffazione è altamente improbabile.

Sebbene l'autenticazione attiva sia ampiamente studiata, solo di recente il mondo della ricerca si sta interessando allo studio e all'impiego della tecnologia passiva nei predetti ambiti applicativi. Ciò comporta che le tecniche di autenticazione fondate su logica passiva non sono ancora fruibili sul mercato, mentre lo scopo della ricerca industriale che ViDiTrust intraprende è proprio la progettazione e realizzazione di questo tipo di soluzioni. Rivolte a coloro che sono interessati a proteggere i propri brand con elevati standard di sicurezza ma con costi contenuti, la tecnologia ViDiTrust mira ad assicurare i package, i documenti e i supporti stampati che accompagnano i prodotti nella fase di diffusione sui mercati dalla sempre più incidente piaga della contraffazione.

2.3.1 Impronta digitale vIDfy

L'impronta digitale vIDfy e la tecnologia ViDiTrust sono protette dal seguente brevetto: *Procedimento di marcatura anticontraffazione di prodotti a stampa e relativo sistema* [ViD11]. In accordo con l'azienda, in questa

sede verranno presentate al lettore alcune informazioni utili a comprendere i principi fondamentali di funzionamento alla base del sistema di anticontraffazione proposto.

L'autenticazione dei documenti stampati avviene per mezzo di una tecnologia passiva che prevede la definizione, l'uso e il riconoscimento di un'impronta digitale, ovvero di un insieme di caratteristiche statistiche proprie della specifica macchina che ha realizzato la stampa e di nessun'altra.

Analizzando accuratamente la fisica dei principali processi di stampa in uso e integrando idee intuitive con la teoria dell'*halftoning*, è possibile realizzare una impronta digitale unica ed essere in grado di riconoscerla.

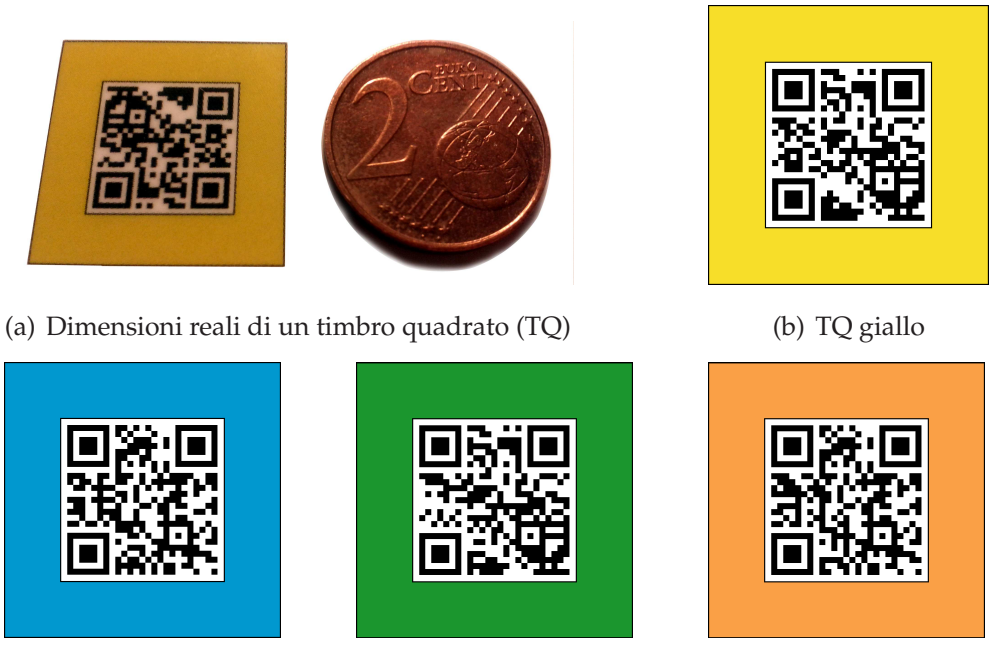
Introduciamo alcuni concetti: l'halftoning, o tecnica dei mezzi-toni, è un procedimento di riproduzione grafico che simula la rappresentazione di immagini a tono continuo attraverso l'uso di punti, variandone opportunamente la dimensione, la forma e la spaziatura e può essere applicata a tutte le immagini, in scala di grigi o a colori.

Il *dithering* invece, è una forma di rumore volontariamente aggiunto ai campioni, utile a minimizzarne la distorsione introdotta da troncamenti di quantizzazione. In campo grafico, le tecniche di dithering permettono di creare l'illusione della profondità di colore in immagini dotate di una tavolozza limitata (quantizzazione del colore). I colori mancanti a video e/o in stampa vengono approssimati dalla distribuzione delle tinte disponibili.

I più comuni processi di stampa vengono attuati tramite tecnica elettrofotografica (comunemente detta tecnologia laser) o a getto d'inchiostro. Tutte le altre classi di stampa, da quella a sublimazione fino alla rotocalcografia utilizzata in ambito tipografico, possono essere accomunate, in termini di caratteristiche relative all'impronta digitale, alle due precedentemente indicate. A prescindere dal particolare tipo di processo considerato, per sua natura esso sarà schematizzabile in due fasi principali: la prima riguarda la preparazione digitale della stampa in cui vengono generate con tecniche *dithering* le matrici di stampa; la seconda riguarda la fase realizzativa vera e propria in cui parti meccaniche ed elettroniche concorrono alla stesura dell'inchiostro sul supporto: carta, cartoncino, plastica, etc.

Dalla realizzazione della meccanica e della modalità di trasferimento del colore sul supporto si evince che le incertezze realizzative dell'hardware incidono profondamente sulla possibilità di estrarre delle caratteristiche in grado di definire e garantire in modo soddisfacente l'unicità dell'impronta digitale. Dipendentemente dalla tecnica di stampa utilizzata, dalla casa produttrice, dal modello e da diversi altri fattori tipici di ogni singola stampante, diviene possibile distinguere fra stampe percettivamente identiche ma realizzate con stampanti differenti.

Analogamente a quanto accade nella balistica forense, in cui considerato un proiettile esploso è possibile stabilire attraverso un'analisi se a esso è associata o meno una presunta arma, allo stesso modo, caratterizzando opportunamente l'impronta digitale del processo di stampa, risulta possibile capire se una certa riproduzione è stata realizzata da una specifica stampante.



(a) Dimensioni reali di un timbro quadrato (TQ)

(b) TQ giallo

(c) TQ celsete

(d) TQ verde

(e) TQ arancione

Figura 2.6: Esempi di codici a barre bidimensionali: timbri ViDiTrust

L'impronta digitale della stampante unitamente a un codice identificativo associato al prodotto vengono cifrati e codificati in un codice bi-

dimensionale chiamato più semplicemente "timbro". In figura 2.6 sono riportati alcuni esempi: la sottofigura 2.6(a) presenta le dimensioni reali di un timbro quadrato, le successive (da 2.6(b) a 2.6(e)) realizzano alcune varianti colorate. I codici bidimensionali appena descritti possono essere utilizzati su dépliant, brochure, etichette, oppure per creare *smartcard* o essere apposti su package e documenti.

Il timbro così realizzato rende possibile anche il reperimento di ulteriori informazioni sul produttore, per esempio i suoi contatti, o sul prodotto come la sua provenienza, la sua composizione, le date di produzione e scadenza; ciò consente di realizzare servizi aggiuntivi quali la tracciabilità delle merci o la verifica dei contenuti. Realizzata la vIDfy (una tantum) e certificato il processo di stampa, il codice bidimensionale verrà stampato nell'etichetta del prodotto su cui si intende applicare la tecnologia anticontraffattiva così da garantirne l'originalità.

Il processo di verifica di genuinità del prodotto in base al timbro che lo accompagna sarà trattato nel prossimo capitolo.

2.4 Conclusioni

In questo capitolo sono state introdotte le definizioni di contraffazione e anticontraffazione e si è discusso della necessità di definire delle identità per poter distinguere fra entità diverse. Allo scopo di contestualizzare il concetto di identità digitale, sono stati riportati alcuni esempi: la Carta di Identità Elettronica della Repubblica Italiana (CIDE) a uso del cittadino, il DOI® System, utile a distinguere le diverse entità all'interno di una rete digitale, il codice IMEI per identificare i singoli dispositivi mobili. A riguardo sono state presentate anche alcune ombre dei vari sistemi per sottolineare quanto sia complesso, a causa di svariati fattori, tradurre in realtà il concetto di identità digitale.

Definito il significato di autenticazione, frutto dell'esigenza di poter verificare in modo oggettivo la genuinità di una certa identità digitale, l'attenzione è stata rivolta al mondo dei documenti stampati. Come esempio portante è stato citato il modello di eccellenza della produzione di carta-

moneta, utile per presentare al lettore un ampio spettro di tecnologie volte a realizzare un poliedrico e robusto processo di autenticazione.

I costi da affrontare per realizzare un livello di protezione così elevato come quello descritto nel caso delle banconote, mal si adattano alla produzione di beni comuni e a basso costo. In questi casi esistono tecnologie alternative e più economiche come il digital watermarking, brevemente presentata in questo capitolo. Le buone caratteristiche di tale approccio mostrano però dei limiti quando le immagini protette subiscono processi di stampa.

Per superare i problemi intrinseci del digital watermarking e mantenere i costi limitati, l'azienda ViDiTrust rende disponibile un approccio di autenticazione passiva fondato sul concetto di impronta digitale: vIDfy. Definita l'impronta digitale come l'insieme delle caratteristiche statistiche proprie di certo processo di stampa (e dunque uniche), si conclude illustrando la realizzazione di un codice bidimensionale (timbro) a essa associato. Costituito dall'impronta digitale e da un ulteriore codice, il timbro permette di costituire un sistema di autenticazione alternativo e al tempo stesso capace di offrire un pacchetto flessibile di servizi aggiuntivi.

Capitolo 3

Applicativi ViDiTrust

L'ideazione dell'impronta digitale unitamente alla sua capacità di essere unica permette di poter distinguere fra una stampa autentica e una contraffatta, anche se realizzate a partire dalla stessa sorgente dati originale. La singolarità del processo di stampa riassunto nella vIDfy opererà da discriminante, ma sarà necessario poter rilevare e verificare le informazioni contenute nel timbro applicato all'oggetto da proteggere.

La fase di verifica dell'autenticità del prodotto prevede tre fasi:

1. l'acquisizione ottica del codice bidimensionale (timbro);
2. l'estrazione della vIDfy dal codice;
3. la verifica via servizio remoto della genuinità dell'impronta digitale.

La prima fase del processo può essere svolta egregiamente per mezzo di un qualsivoglia dispositivo provvisto di sensore ottico: uno scanner, una fotocamera, uno smartphone. In ogni modo, i tre stadi saranno controllati e realizzati per mezzo di software dedicato e sviluppato internamente all'azienda ViDiTrust.

A oggi esistono due soluzioni applicative distinte, una è specificatamente ideata per postazioni fisse ed è al momento disponibile per piattaforme Windows. Principalmente orientata alle attività interne di imprese e uffici, l'acquisizione del timbro avviene per mezzo di comuni scanner domestici. Il campo applicativo direttamente interessato è quello della *do-*

cument security e dunque della protezione contro la falsificazione e la copia di documenti ufficiali.

La seconda applicazione è sviluppata per dispositivi mobili e ha un variegato campo di utilizzo. Rispetto alla precedente gode della proprietà di autonomia come descritto nell'introduzione al capitolo 1 e ciò rende possibile al consumatore accertarsi di persona e prima dell'acquisto dell'originalità di un qualsiasi prodotto coperto dalla tecnologia ViDiTrust.

In questo capitolo verrà descritto il funzionamento del software per smartphone e ne saranno presentate le caratteristiche più importanti: è su tale applicativo che è stato svolto il lavoro di tesi.

3.1 Android application: ViDiApp

L'applicazione sviluppata per dispositivi mobili prende il nome di *ViDiApp* e dopo averne presentato i requisiti hardware e software e un elenco di smartphone ove l'app è stata sviluppata e ampiamente testata, si offrirà al lettore una breve guida, utile alla comprensione del suo funzionamento.

Prima di procedere, analizziamone l'architettura per mezzo della figura 3.1.

Per poter verificare l'autenticità di un prodotto protetto dal sistema ViDiTrust come per esempio la bottiglia di vino riportata in basso a sinistra, l'acquirente non dovrà fare altro che avviare l'applicazione sul proprio smartphone, posto al centro della figura. Una volta inquadrato il timbro con la fotocamera (ingrandimento in alto a sinistra), il software provvederà ad acquisire il codice bidimensionale (passo 1) e dopo una breve elaborazione locale al dispositivo (passo 2), una richiesta di verifica verrà inoltrata ai server ViDiTrust (passo 3). La comunicazione verrà intercettata tramite un web service (passo 4) e il messaggio ricevuto sarà elaborato in un paio di secondi (passo 5). Preparata la risposta, essa verrà inviata a ritroso via Internet (passo 6) e infine ricevuta dallo smartphone. A conclusione di queste procedure, il responso sarà visualizzato sul display del dispositivo (passo 7) e l'acquirente disporrà di alcune importanti informa-

Figura 3.1: Architettura di ViDiApp e del sistema di autenticazione

zioni prima di effettuare l'acquisto. La più rilevante riguarderà l'autenticità o meno del prodotto, poi un'insieme di altre, alcune basilari quali il nome del prodotto e i dati del produttore, altre specifiche e concordate con quest'ultimo.

Prima di osservare più da vicino la struttura modulare dell'applicazione, ne forniamo alcuni dettagli quali i requisiti di sistema e i principi di funzionamento.

3.1.1 Requisiti di sistema

Per poter utilizzare l'applicazione di verifica di autenticità ViDiApp, l'operatore deve disporre di uno smartphone (o tablet) dai seguenti requisiti hardware minimi:

- fotocamera da 5 Megapixel con flash e autofocus;
- accesso alla rete Internet (GSM/UMTS o Wi-Fi).

Per quanto riguarda i requisiti software, l'app è in grado di eseguire sulle seguenti piattaforme:

- Android versione 2.2, *Froyo* (API 8);
- Android versione 2.3-2.3.2, *Gingerbread* (API 9);
- Android versione 2.3.3-2.3.7, *Gingerbread* (API 10).

Per avere migliori tempi di risposta, sono consigliati dispositivi con buone capacità di elaborazione in termini di processore e memoria, nonché una connessione di rete prestante. Alcuni esempi di smartphone disponibili sul mercato in possesso delle predette caratteristiche sono presentati in tabella 3.1.

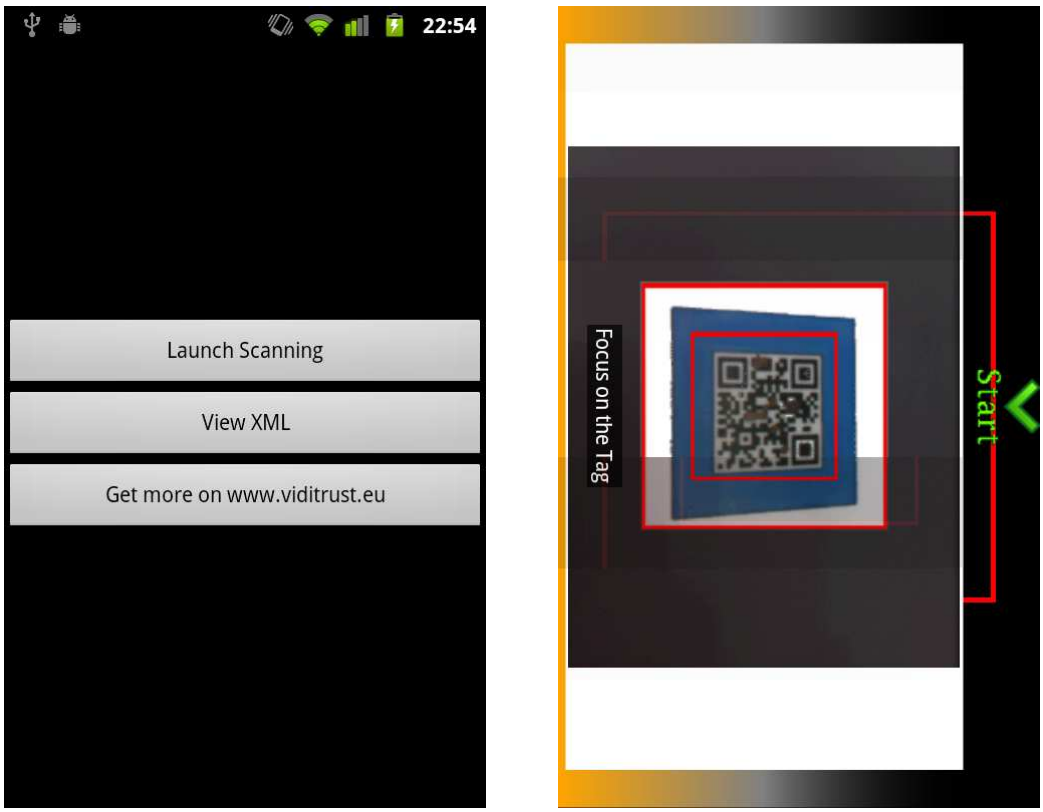
Modello	**CPU**	**RAM**	**Android**	**API**
HTC Wildfire	Qualcomm (528 MHz)	384 MB	v. 2.2	8
Samsung Galaxy ACE	Qualcomm (800 MHz)	278 MB	v. 2.3.6	10
Samsung Nexus S	ARM Cortex (1 GHz)	512 MB	v. 2.3.6	10
Samsung Galaxy W	Qualcomm (1.4 GHz)	512 MB	v. 2.3.5	10

Tabella 3.1: Modelli di smartphone compatibili con ViDiApp

Tutti gli smartphone elencati sono molto simili riguardo alla fotocamera incorporata e presentano identiche caratteristiche: sensore ottico da 5 Megapixel, autofocus e LED flash.

3.1.2 Principi di funzionamento

Installata l'applicazione ViDiApp sul proprio smartphone, l'utente potrà avviarla come una qualsiasi altra app, premendo sullo schermo touch la corrispondente icona. Entro pochi istanti verrà caricata una semplice GUI (*Graphical User Interface*) che presenterà tre pulsanti, come indicato in figura 3.2(a).



(a) Schermata iniziale con menù di scelta (b) Presentazione e help animato

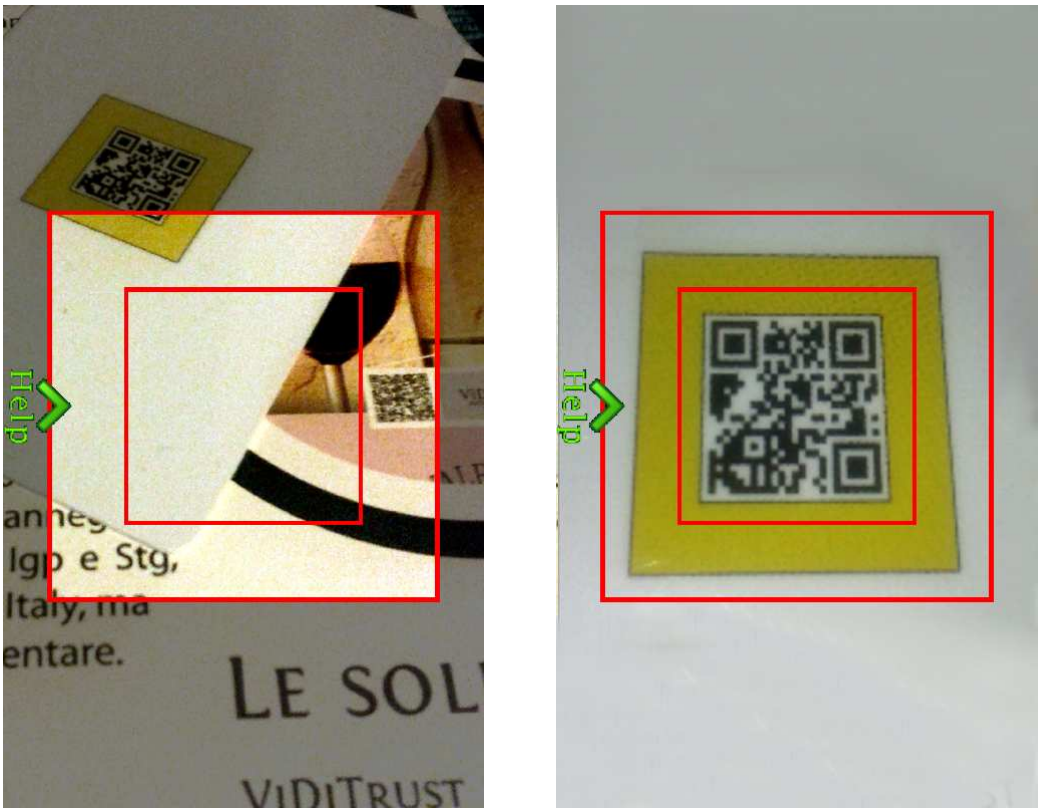
Figura 3.2: Schermate principali di ViDiApp

La funzionalità principale è associata al bottone *Launch Scanning*, mentre i restanti due permettono di visualizzare sul display rispettivamente l'esito dell'ultima validazione effettuata (*View XML*) e la pagina home del sito ViDiTrust (*Get more on* `www.viditrust.org`).

Una volta premuto il pulsante Launch Scanning, sullo schermo comparirà una presentazione, come indicato in figura 3.2(b): una breve ani-

mazione illustrerà all'utente l'uso dell'applicazione. La verifica di autenticità potrà essere effettuata semplicemente inquadrando in una maschera disegnata a schermo il timbro ViDiTrust.

Dopo aver premuto sulla scritta verde Start (o aver trascinato la presentazione a lato dello schermo), il display visualizzerà una maschera dai bordi rossi costituita da due quadrati innestati l'uno nell'altro (3.3).



(a) Maschera di scansione (bordi rossi) (b) Maschera di scansione in posizione

Figura 3.3: Schermate di scansione di ViDiApp

La fotocamera sarà attivata e mentre all'esterno della maschera l'immagine in trasparenza apparirà in ombra, al suo interno l'utente potrà inquadrare l'etichetta o il supporto stampato ove è presente il timbro di autenticazione. Da una distanza ravvicinata utile a realizzare scatti macro (circa sette/dieci centimetri), l'utente dovrà posizionarsi col quadrato interno dai bordi rossi sul *QR code* e contemporaneamente, con la ghiera

esterna individuata dai due quadrati, coprire la zona colorata, l'impronta digitale vIDfy. Quanto appena descritto può essere osservato in figura 3.3.

Se la distanza del dispositivo dal codice bidimensionale è corretta e l'inquadratura è tale che il timbro risulta essere tutto compreso nella maschera, in pochi istanti la fotocamera eseguirà l'autofocus e scatterà una foto. Tramite un apposito menù che è possibile richiamare a video con l'omonimo bottone dello smartphone, all'utente sarà comunque possibile abilitare e disabilitare il flash in modo manuale.

A conclusione di questa procedura automatica, verrà visualizzata una schermata fissa con lo scatto effettuato, come è possibile osservare in figura 3.4.

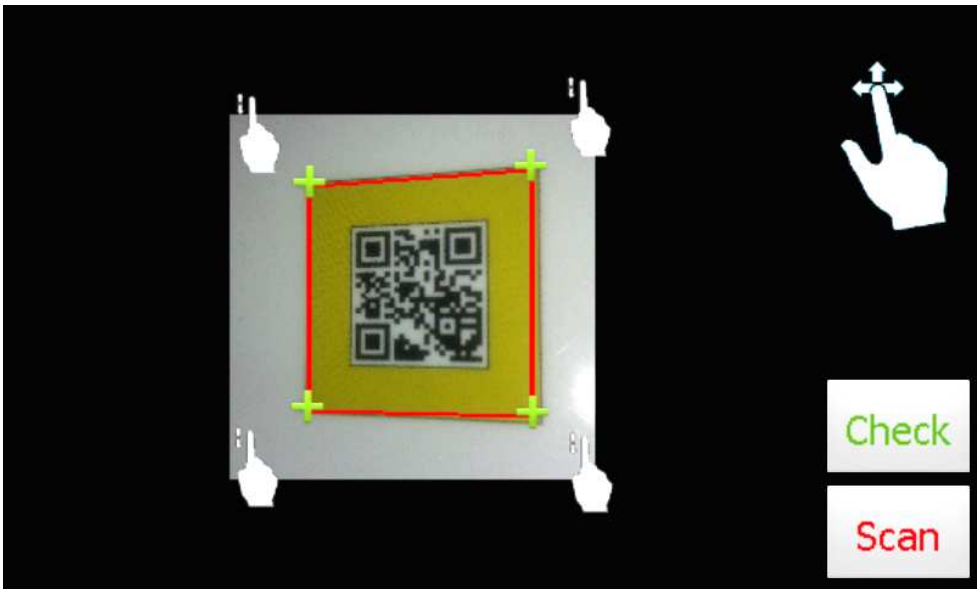


Figura 3.4: Immagine ottenuta dalla procedura automatica

A questo punto l'utente potrà intervenire per correggere eventuali non centrature del timbro trascinando i vertici della maschera rossa al fine di coprire tutta la superficie del codice bidimensionale (figura 3.5).

Dalla stessa schermata è possibile tornare alla maschera di acquisizione premendo il tasto *Scan*, oppure procedere con l'invio al server del timbro acquisito per mezzo del bottone *Check*.

L'immagine scattata e confermata dall'utente verrà dapprima processata dallo smartphone e poi inviata in rete dietro richiesta di un servizio

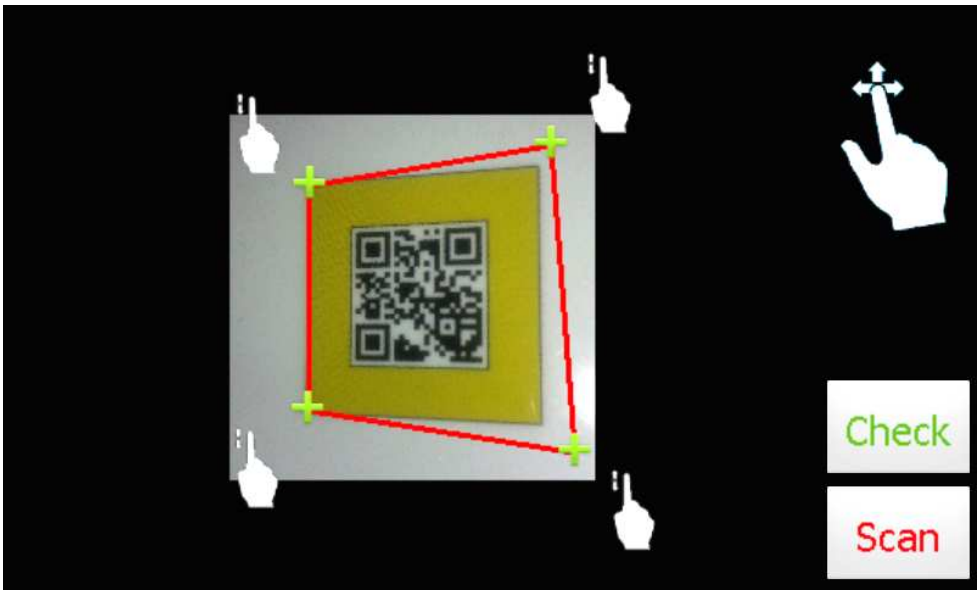
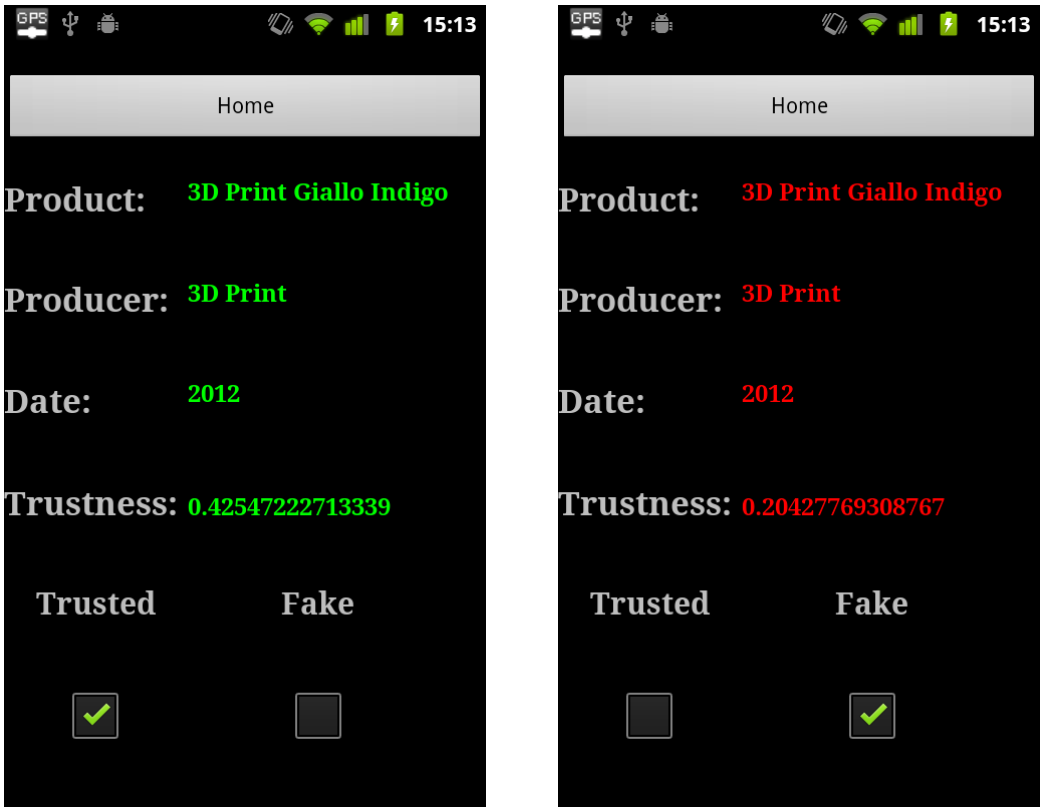


Figura 3.5: Spostamento manuale di due vertici della maschera

remoto, in modo molto simile a come illustrato nel paragrafo 4.1.2. Dopo l'attesa di alcuni secondi, il dispositivo mobile riceverà la risposta dal servitore e potrà visualizzarne il risultato, come mostrato in figura 3.6. L'informazione più importante fra quelle mostrate indicherà se il prodotto è autentico oppure è un falso, inoltre verranno riportati altri dati, come ad esempio il nome del prodotto, quello dell'azienda produttrice e dei suoi riferimenti e qualsiasi altro elemento con essa concordato.

I tempi necessari all'esecuzione dell'intera procedura sono variabili e dipendono da molti fattori: l'elaborazione della foto sullo smartphone, l'inoltro al servitore dei dati da essa ricavati, la preparazione della risposta e l'invio a ritroso della stessa. In condizioni di normale funzionamento possiamo affermare che l'utente dovrà attendere mediamente fra i sei e gli otto secondi dal momento in cui è stato eseguito e validato lo scatto fino alla ricezione dell'esito della verifica. Il servizio remoto realizzato è sincrono e bloccante e pertanto prevede attesa, ma l'utente può eseguire altre applicazioni sul proprio dispositivo in virtù delle capacità concorrenti offerte da Android, come descritto nel paragrafo 1.1.2.1.



(a) Prodotto autentico (b) Prodotto falso

Figura 3.6: Schermate di visualizzazione del risultato di ViDiApp

3.2 Architettura dell'applicazione

Per poter individuare agevolmente le aree di intervento interne all'applicazione, si vogliono illustrare per mezzo di una rappresentazione grafica i principali blocchi funzionali che costituiscono ViDiApp. La schematizzazione presentata in figura 3.7 è volutamente poco dettagliata e non ha pretese di congruenza con la realizzazione interna e a basso livello del software. Le motivazioni di tale scelta sono dettate dal voler offrire al lettore una visione sintetica e semplificata dell'argomento, ma anche per questioni legali associate al brevetto con cui è protetta la tecnologia in uso.

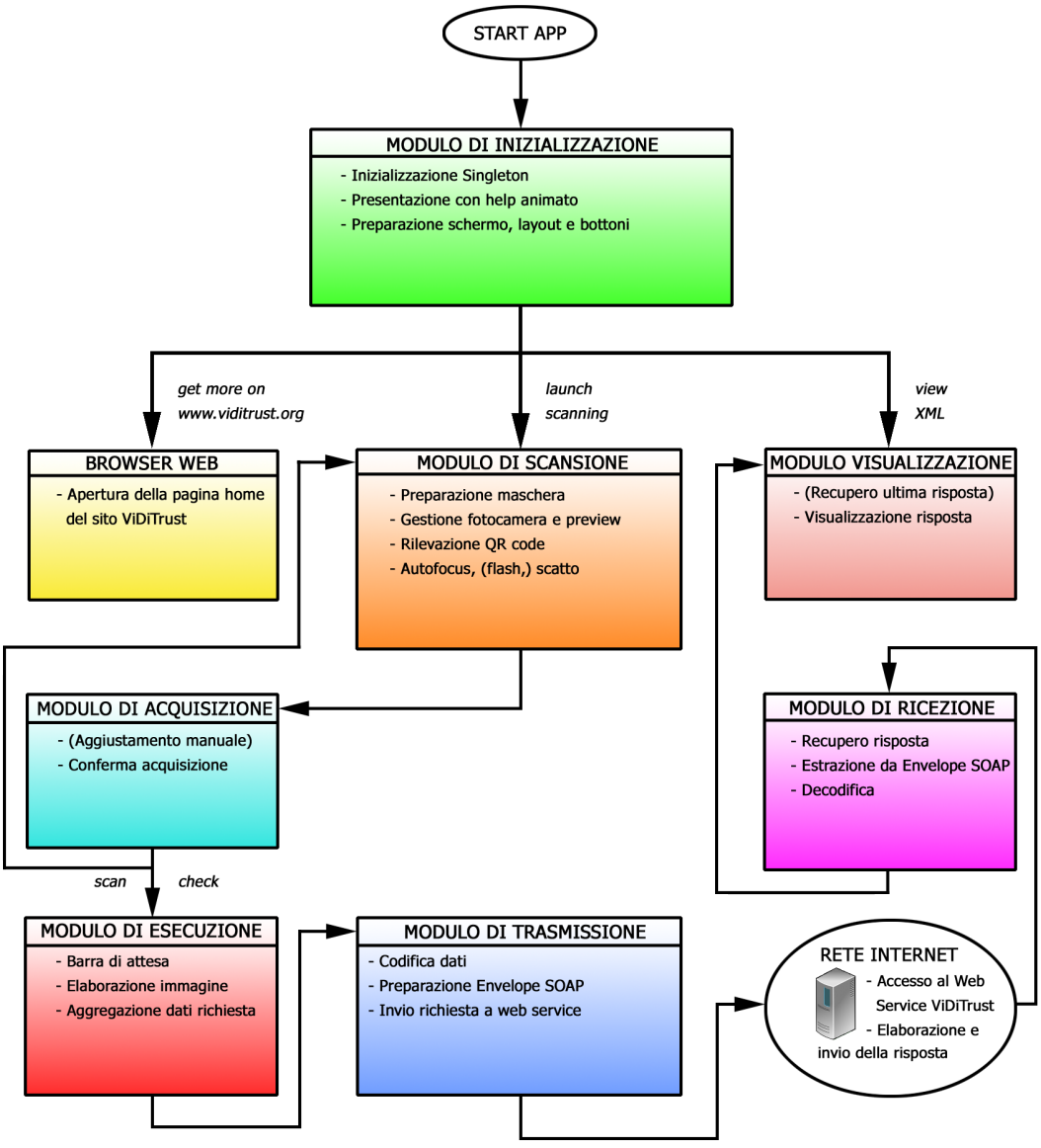


Figura 3.7: Blocchi funzionali della ViDiApp

Invocare l'esecuzione della app prevede l'accesso al primo elemento raffigurato, ovvero al *Modulo di Inizializzazione*. Tale modulo ha la capacità di creare un oggetto *singleton* utile per mantenere aggiornati, disponibili e integri i parametri di sistema e le grandezze d'uso di tutti gli altri blocchi funzionali. Un altro compito basilare del modulo è quello di presentare una breve animazione con lo scopo di illustrare le modalità d'uso dell'applicazione e per concludere, realizzerà anche l'interfaccia grafica, predisponendone i pulsanti descritti nel paragrafo precedente e rendendoli attivi. Selezionando uno dei tre bottoni, l'utente attiverà le funzionalità espletate da uno dei tre seguenti moduli: *Modulo di Scansione*, *Modulo di Visualizzazione* o il *Browser Internet*. In riferimento a quanto detto nel paragrafo 1.1.2.1, ciascun modulo appena indicato sarà realizzato per mezzo di specifici intent.

Qualora fossero richieste maggiori informazioni sull'impresa ViDiTrust o sui suoi prodotti e tecnologie, il modulo richiamato sarà il Browser Internet che aprirà una connessione col web server aziendale e visualizzerà sul display la pagina home del suo sito ufficiale.

Al contrario, se l'utente vorrà richiamare a video il risultato dell'ultima verifica effettuata, a essere invocato sarà il Modulo di Visualizzazione che caricherà dal dispositivo le informazioni di autenticità di un certo prodotto, presentandole come visto in figura 3.5 del precedente paragrafo. Ovviamente, se l'applicazione non è stata mai avviata o se per altri motivi la cache dati risulta essere vuota, il modulo sarà in grado di avvisare l'utente in modo adeguato della mancata visualizzazione.

Infine, il tipico caso d'uso sarà quello di eseguire una verifica di autenticità e pertanto verrà invocato il Modulo di Scansione. L'insieme dei suoi compiti prevede il cambiamento del layout grafico e la realizzazione a video della maschera di acquisizione del timbro, quella dai bordi rossi visibile nelle figure 3.3(a) e 3.3(b). Allo stesso tempo sarà inizializzata la fotocamera e gestita la *preview*, ovvero l'immagine inquadrata istante per istante. Intervenendo con l'autofocus ed eseguendo dei controlli sulla lettura del QR code, il modulo eseguirà automaticamente uno scatto e se necessario userà il LED flash del dispositivo.

Infine, realizzata la foto, il Modulo di Acquisizione individuerà il timbro e lo mostrerà all'utente, contornandolo con una maschera quadrata dagli angoli mobili. Il modulo implementa una procedura di acquisizione assistita e permette all'utilizzatore di:

- rieseguire la foto, magari per uno scatto fuori fuoco (rimando al Modulo di Scansione);
- distorcere manualmente la maschera operando sui suoi vertici al fine di coprire completamente il codice bidimensionale;
- trasmettere lo scatto e le informazioni relative al successivo Modulo di Esecuzione.

Il Modulo di Esecuzione eseguirà un insieme di operazioni che possono essere raggruppate in due categorie, quella comprendente le manipolazioni dell'immagine e quella relativa alla pacchettizzazione dei dati da inviare al server. Responsabilità del modulo sarà il ricavare le informazioni proprie dell'impronta digitale e accorparle a svariate altre, di sistema e non, utili al corretto funzionamento delle procedure di autenticazione. L'insieme di queste operazioni, come pure di quelle eseguite dai moduli seguenti, saranno tutte mascherate da una finestra di attesa animata.

Il pacchetto informativo così preparato verrà processato e codificato dal Modulo di Trasmissione che provvederà anche a produrre l'envelope SOAP e a inoltrare la richiesta di elaborazione a un servizio remoto realizzato per mezzo di un Web Service (vedere i paragrafi 1.2.4 e 1.2.4.3).

Instradata la richiesta su Internet, raggiungibile tramite Wi-Fi o connessione dati su rete cellulare, l'applicazione resterà in attesa del responso elaborato dai server ViDiTrust.

Il Modulo di Ricezione si occuperà di intercettare la risposta, estrarne i risultati dall'envelope SOAP e decodificarne il contenuto.

Per concludere, i dati ottenuti verranno inviati al Modulo di Visualizzazione che li proporrà a video all'utente, come descritto più sopra.

3.2.1 Identificazione delle aree d'intervento

L'applicazione ViDiApp per dispositivi mobili è in continua evoluzione e dallo stato embrionale in cui versava, oggi presenta un buon grado di maturità. Sviluppare un programma per smartphone presenta svariate difficoltà aggiuntive rispetto alla realizzazione di un software pensato per "postazione fissa" e ciò dipende da diversi fattori: dall'utilizzo di una piattaforma ricca e complessa come Android (dai repentini mutamenti in versioni successive), dall'implementazione di protocolli non ancora integrati nelle API, dalla conoscenza operativa e d'uso di sensori non previsti sui desktop, dalla disponibilità di risorse molto limitate.

Per questi motivi, il lavoro di tesi ha operato su una molteplicità di obiettivi, incentrati sulle necessità aziendali: dallo studio di fattibilità di certe funzionalità, alla ricerca e realizzazione di soluzioni applicative, dallo sviluppo e implementazione di controlli, al debug e alla valutazione delle prestazioni.

Qui di seguito verranno presentate le aree di intervento e le motivazioni operative, nei capitoli successivi ne saranno descritte le specifiche tecniche (aiutandosi ove opportuno col commento di alcuni stralci di codice), nonché i risultati raggiunti e la valutazione degli stessi.

La prima zona di intervento riguarda il Modulo di Trasmissione in cui si è voluto introdurre una gestione alternativa dell'invio dei dati. Le finalità dei cambiamenti introdotti mirano a una trasmissione più efficiente, ma al tempo stesso alternativa e dunque compatibile con la precedente realizzazione.

Un'altra area di alto interesse operativo è quella relativa al Modulo di Inizializzazione. In esso sono state introdotte nuove funzionalità capaci di identificare il dispositivo mobile e la sua posizione sulla Terra. Lo scopo di questi interventi sono da un lato la prevenzione degli abusi del servizio di autenticazione, mirati al danneggiamento a mezzo di attacchi informatici, dall'altro la realizzazione di una base per lo sviluppo di servizi ulteriori, quali la tracciabilità dei prodotti, l'individuazione e mappatura delle aree più a rischio contraffazione, la diffusione e l'uso del software ViDiTrust.

Sempre nel Modulo di Inizializzazione, come pure nel Modulo di Trasmissione, sono stati introdotti i necessari controlli di rete utili ad aumentare la qualità di servizio percepita dall'utente. L'applicazione deve verificare e comunicare all'utilizzatore eventuali problemi di connessione dipendenti dal dispositivo mobile oppure dalla non disponibilità del servizio di autenticazione a causa di disfunzioni dei servitori online.

Altri interventi sono stati effettuati sul Modulo di Esecuzione e nuovamente sul Modulo di Trasmissione relativamente all'elaborazione e gestione dei nuovi dati riguardanti l'identificazione e la geolocalizzazione.

Infine, numerose altre modifiche minori ma di una certa complessità strutturale sono state apportate alla maggior parte dei costituenti realizzativi di tutti i blocchi funzionali. Gli scopi di tali interventi sono fra i più vari, ma tutti accomunati da esigenze di integrazione, generalizzazione, semplificazione, modularizzazione e ingegnerizzazione del codice applicativo.

3.3 Conclusioni

In questo capitolo sono stati inizialmente presentati i tre stadi attraverso i quali si realizza l'autenticazione del prodotto per mezzo del sistema passivo ViDiTrust: acquisizione ottica del timbro, estrazione della vIDfy, verifica della stessa via servizio remoto.

Dopo aver brevemente presentato alcuni dei campi applicativi della tecnologia, sono state indicate due soluzioni software distinte, una orientata ad applicazioni che usano una postazione fissa, l'altra portabile e installabile sugli smartphone. Quest'ultima è stata maggiormente indagata, fornendo al lettore ulteriori elementi informativi quali la struttura generale dell'architettura di verifica, i requisiti minimi hardware e software necessari all'installazione e all'esecuzione dell'applicativo, nonché una guida utente che ne illustra il funzionamento.

Il lavoro di tesi svolto ha come oggetto ViDiApp, l'applicazione per dispositivi mobili, e pertanto ne sono stati presentati i principali blocchi funzionali sia in modo grafico che per mezzo di una breve descrizione.

A fine capitolo, in base ai moduli operativi sopra descritti, sono state identificate le principali aree di intervento e le finalità delle modifiche introdotte. Esse possono riassumersi in: ricerca, sviluppo e implementazione di nuove funzionalità, comprese quelle di controllo; integrazione, generalizzazione, semplificazione, modularizzazione e ingegnerizzazione del codice applicativo, nuovo e preesistente.

Capitolo 4

Ingegnerizzazione, ottimizzazione e studi di fattibilità

Nel precedente capitolo, nel paragrafo 3.2, sono state individuate le principali aree di intervento e le ragioni operative alla base delle modifiche introdotte nell'applicazione ViDiApp. In questa sede esporremo quanto realizzato durante il lavoro di tesi e riporteremo ove necessario alcuni dettagli implementativi. A tal proposito, si ricorda al lettore che il linguaggio di programmazione utilizzato è Java.

4.1 Modulo di Trasmissione

Uno degli interventi più significativi coinvolge il Modulo di Trasmissione e per semplificarne la comprensione, di seguito viene riportata una rappresentazione grafica maggiormente dettagliata rispetto a quanto visto a pagina 86. In figura 4.1 è riprodotta la struttura originaria di alto livello del Modulo di Trasmissione.

Il servizio da richiedere online, ovvero la verifica di autenticità del prodotto, prevede a questo punto del flusso di esecuzione l'invocazione di un servizio remoto a cui occorre fornire le informazioni raccolte ed elaborate

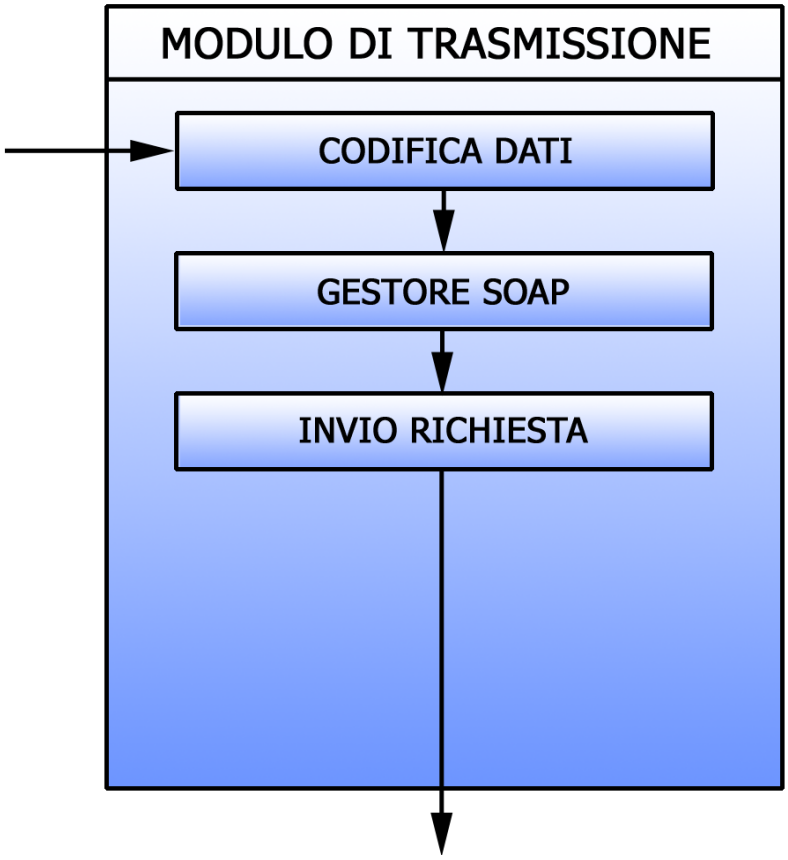


Figura 4.1: Modulo di Trasmissione originario

dal Modulo di Esecuzione. Implementato per mezzo di un web service, il servizio online prevede una procedura di comunicazione basata su quanto descritto nel paragrafo 1.2.4. Dapprima le informazioni da inviare verranno codificate per mezzo del blocco *Codifica Dati* interno al Modulo di Trasmissione, con lo scopo di ottenerne un messaggio XML di alto livello. Successivamente, il messaggio verrà trasferito al blocco *Gestore SOAP* che lo trasformerà in un "oggetto SOAP" che poi verrà incapsulato in una envelope del tipo visto in figura 1.14. Infine, tramite l'uso di alcuni metodi specifici operanti sulla envelope appena realizzata, il blocco *Invia Richiesta* invocherà il servizio remoto.

4.1.1 SOAP e libreria kSOAP2

Gli ultimi due blocchi descritti nel precedente paragrafo si occupano di realizzare l'oggetto SOAP, l'envelope e l'invio della richiesta al web server. Purtroppo, il sistema operativo mobile Android non prevede librerie dedicate alla gestione del protocollo SOAP, almeno nelle API di versione 8, 9 o 10. Per tale motivo, le importanti funzionalità sopra descritte vengono implementate nell'applicativo ViDiApp per mezzo di una libreria open source raccomandata da molti sviluppatori, ksoap2 [HS12]. Per ovvi motivi, il codice dei due blocchi sarà sviluppato in modo da utilizzarne i metodi la cui documentazione completa può essere rintracciata online, sul sito `http://ksoap2.sourceforge.net/doc/api/`.

Per avere un'idea del funzionamento base della libreria, riportiamo un piccolo esempio d'invocazione di un metodo remoto raggiungibile con un web service (WS). Nel codice in figura 4.2 viene presentato un WS realizzato per mezzo di Visual Studio.

```
1 // Web Service in esecuzione in locale: http://192.168.1.3
2 public class SimpleWebServices : System.Web.Services.WebService
3 {
4     // Remote Web Method
5       public int getSumOfTwoInts(int operand1, int operand2)
6       {
7         return operand1 + operand2;
8       }
9 }
```

Figura 4.2: Web Service realizzato in Visual Studio

La libreria kSOAP si basa fondamentalmente su un oggetto chiamato *SoapObject* e i tre parametri principali per poterlo utilizzare sono:

- il namespace del web service;
- il nome del metodo remoto;
- l'URL ove il web service è raggiungibile.

Esiste anche un'altra variabile molto importante comunemente nota come SOAP_ACTION, ma è semplicemente la concatenazione dei nomi del namespace e del metodo remoto.

Nel codice presentato in figura 4.3 sono riassunti i parametri appena discussi e nell'ultima riga verrà creata l'istanza di un SoapObject.

```
// Definizione e inizializzaz. dei parametri di un SoapObject
String NAMESPACE = "http://wstest.org/";
String METHOD_NAME = "getSumOfTwoInts";
String URL = "http://192.168.1.3/VipEvents/" +
         "Services/BasicServices.asmx";
String SOAP_ACTION = "http://wstest.org/getSumOfTwoInts";
// Creazione e inizializzazione del SoapObject
SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
```

Figura 4.3: Definizione e inizializzazione dei parametri utili al SoapObject

```
// Creazione oggetti p1 e p2 con i parametri interi
PropertyInfo pi1 = new PropertyInfo();
  pi.setName("operand1");
  pi.setValue(2); pi.setType(int.class); // Intero 2
PropertyInfo pi2 = new PropertyInfo();
  pi2.setName("operand2");
  pi2.setValue(5); pi2.setType(int.class); // Intero 5
// Aggiungiamo i parametri all'oggetto SoapObject
request.addProperty(pi1);
request.addProperty(pi2);
// Creazione della SOAP Envelope (SOAP v1.1)
SoapSerializationEnvelope envelope =
  new SoapSerializationEnvelope(SoapEnvelope.VER11);
// Imposto la compatibilita' con un Web Service .NET based
envelope.dotNet = true;
// Imposto l'object SOAP come messaggio d'uscita nell'env.
envelope.setOutputSoapObject(request);
```

Figura 4.4: Inserimento dei parametri nel SoapObject e incapsulamento nella SOAP envelope

Infine occorre aggiungere all'oggetto i parametri attesi dal metodo remoto, ma per farlo è necessario utilizzare elementi di appoggio realizzati per mezzo della classe *PropertyInfo.class*, come mostrato nel codice in figura 4.4.

Prima di procedere oltre occorre definire un oggetto di trasporto e infine agire sul SoapObject per ottenere il risultato, ovvero l'invocazione del metodo remoto capace di eseguire la somma dei parametri interi incapsulati nell'envelope. Quanto appena descritto è presentato nel codice in figura 4.5.

```
1  // Creazione del Transport Object
2  AndroidHttpTransport aHttpTransport =
3    new AndroidHttpTransport(URL);
4
5  try {
6    // Invocazione del metodo remoto via Web Service
7    aHttpTransport.call(SOAP_ACTION, envelope);
8    // Recupero della risposta
9    SoapObject response = (SoapObject)envelope.getResponse();
10   // Estrazione del risultato
11   int result =
12     Integer.parseInt(response.getProperty(0).toString());
13 }
14 catch(Exception e){
15   e.printStackTrace();
16 }
```

Figura 4.5: Definizione oggetto di trasporto e recupero del risultato

4.1.1.1 Limiti di kSOAP2

La libreria poc'anzi descritta funziona discretamente bene, almeno per l'uso previsto all'interno dell'applicazione ViDiApp, ma presenta alcuni difetti. Sorvolando sui dettagli tecnici, in questa sede si vuole indicare al lettore un elenco di limiti riscontrati nell'uso di kSOAP2, non tutti di carattere operativo.

Fra le mancanze più rilevanti della libreria open source kSOAP2 riportiamo le seguenti:

- gestisce solo la versione SOAP 1.1;
- presenta diversi bug documentati, alcuni riscontrati in modo diretto (protocollo https e opzione keepalive);
- non rende disponibili controlli sullo stato di progressione del trasferimento/richiesta;
- offre una documentazione incompleta per diversi scenari di utilizzo come il passaggio o la restituzione di oggetti complessi, lavorare con array di oggetti o con le date;
- non garantisce aggiornamenti futuri;
- prevede un certo grado di complessità architetturale.

Per i motivi appena elencati, ma anche per tentare un approccio più efficiente, una delle prime necessità manifestate dall'azienda è stata quella di realizzare uno studio di fattibilità sullo sviluppo di un protocollo di comunicazione alternativo, ma sempre di alto livello (applicativo).

4.1.2 Alternativa HTTP

Da una preventiva analisi della descrizione WDSL del web service presente sui server ViDiTrust si è notata la possibilità di operare per mezzo del protocollo HTTP tramite i *request method* GET e POST. Successivamente sono state effettuate ulteriori indagini lato client, volte a verificare se Android disponesse, a differenza del caso SOAP, delle librerie necessarie a realizzare e gestire delle comunicazioni tramite il protocollo HTTP.

La piattaforma si è dimostrata capace di operare direttamente con tale protocollo ed è dotata delle interfacce e delle classi utili allo scopo, raccolte in un package fondamentale *org.apache.http*, la cui documentazione può essere recuperata online [Dev12].

A termine di questa analisi preventiva, sono stati effettuati i primi passi operativi utili a trasformare il Modello di Trasmissione e renderlo simile a quanto indicato in figura 4.6.

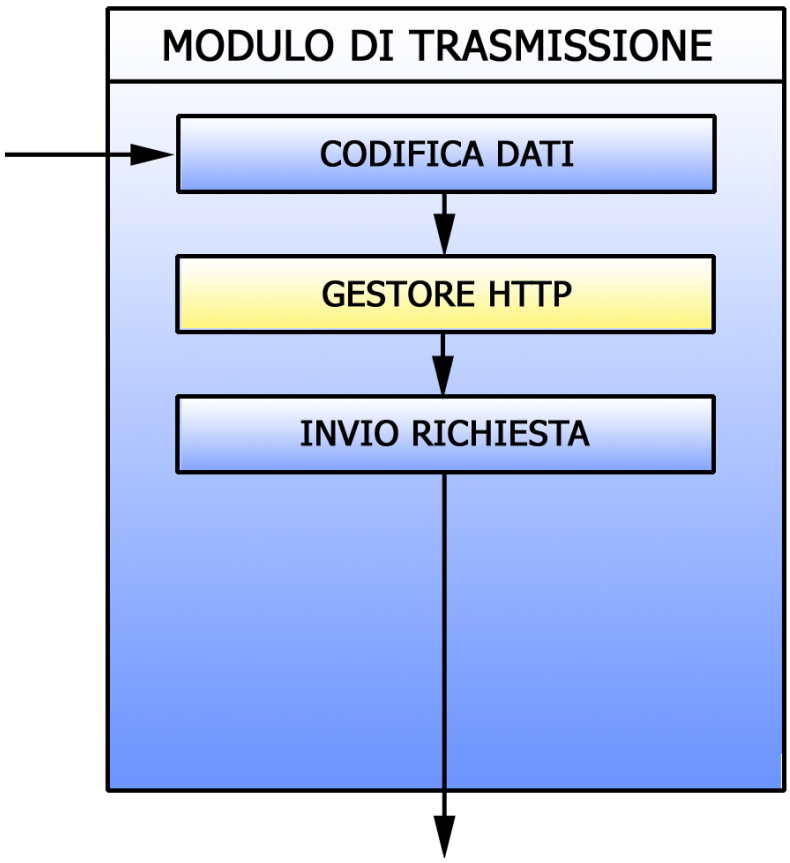


Figura 4.6: Sostituzione del Gestore SOAP con il Gestore HTTP

Il protocollo HTTP offre diversi metodi, ciascuno pensato per uno specifico scopo, ma i più noti e utilizzati sono due, il GET e il POST. Ambedue permettono di indicare una azione che il cliente richiede al server, il quale la eseguirà su una risorsa, identificata tramite un *Uniform Resource Identifier* (URI).

Il metodo GET è stato concepito per chiedere informazioni a un server, inviandogli pochi parametri tramite l'URL, specializzazione di URI, attraverso una stringa di query, ovvero una parte di testo che segue un punto interrogativo. Un semplice esempio è quello di richiedere informazioni a un sito web e visualizzarle nel browser, come già indicato nell'esempio del paragrafo , ma ora si desidera indicare al server il valore da usare per un certo parametro.

Esempi HTTP/GET:

1. `http://ekm.altervista.org/ekm_b.php?ekmn=1`
2. `http://ekm.altervista.org/ekm_b.php?ekmn=1&test=una+prova`

Nel primo caso, si può notare una richiesta GET ove viene specificato il solo parametro *ekmn* a cui viene assegnato il valore "1". Nell'esempio 2 è invece indicato come suggerire le impostazioni per più parametri. Ciò avviene per mezzo di coppie del tipo 'nome=valore', separate dal carattere '&' con l'aggiunta del simbolo '+' ove occorrano degli spazi.

Diversamente da quanto appena descritto, il metodo POST è stato pensato per inviare al server molte informazioni, in modo non visibile da URL, senza un limite sulla quantità di dati da trasmettere o sul loro tipo.

Realisticamente, il metodo POST sostituisce il GET quando la richiesta da effettuare superi il massimo numero dei caratteri consentiti dal server (ad esempio 256) e le informazioni da inviare non siano solo di tipo alfanumerico, ma prevedano immagini, dati complessi, etc.

Quest'ultimo caso incarna esattamente le esigenze della ViDiApp e pertanto è stato realizzato il blocco *Gestore HTTP* capace di operare tramite metodo POST. Inizialmente sviluppato in una applicazione *ad hoc* eseguibile su normali elaboratori dotati di Java Virtual Machine, raggiunta l'operatività prevista, il blocco è stato integrato in una app Android. Per diverso tempo sono state effettuate test e verifiche, attuate sia su dispositivi virtuali, sia sui modelli fisici presenti in tabella 80, principalmente sullo smartphone Nexus S. Come ultimo passo, il Modulo di Trasmissione aggiornato è stato sostituito nel core dall'applicazione ViDiApp.

Il Gestore HTTP è stato successivamente modificato per renderlo compatibile con il blocco preesistente (Gestore SOAP) e pertanto si preferisce rimandare il lettore al paragrafo successivo per ogni riferimento e commento al codice.

4.1.3 Integrazione dei componenti SOAP/HTTP

Come anticipato, il passo successivo alla realizzazione del Gestore HTTP è stato quello di integrarlo col blocco SOAP già sviluppato. Abbiamo anche esaminato nel paragrafo 4.1.1 come realizzare una richiesta SOAP tramite la libreria kSOAP2, ma ciò risulta essere profondamente diverso che operare via HTTP/POST. Ciò è dovuto in primo luogo alla necessità di eseguire una connessione tramite client HTTP, un oggetto Java realizzato con una struttura e che prevede metodi pensati per operare diversamente da quelli previsti per il SoapObject. Inoltre, l'invio della richiesta a mezzo del relativo blocco, deve tener conto della gestione o meno dell'envelope utile solamente nel caso SOAP.

Inizialmente si è tentato di realizzare una struttura con classi in eredità: definito un gestore base, le sue specializzazioni con le classi figlie lo avrebbero reso compatibile con un uso HTTP/POST piuttosto che uno SOAP. In realtà l'architettura realizzata presentava alcune incompatibilità che potevano essere risolte solo introducendo inutile complessità o riprogettando l'intero Modulo di Trasmissione, opzioni reputate inaccettabili.

L'approccio adottato nella realtà prevede la definizione e l'uso di una semplice interfaccia (*Interface_Transferable.java*) orientata a dichiarare i comportamenti comuni ai due gestori e al blocco di Invio Richiesta.

Prima di esaminare il codice dell'interfaccia, presentiamo l'evoluzione del Modulo di Trasmissione per mezzo di una rappresentazione grafica riportata in figura 4.7.

Eseguita la codifica dei dati, tramite l'interfaccia è possibile realizzare un gestore SOAP o HTTP a seconda delle necessità e soddisfacendo il vincolo aziendale, per poi utilizzare sull'oggetto prescelto le opportune operazioni di invio.

Le azioni più importati da definire sono le seguenti:

- inviare un messaggio;
- prelevare un messaggio ricevuto;
- recuperare informazioni sulle operazioni.

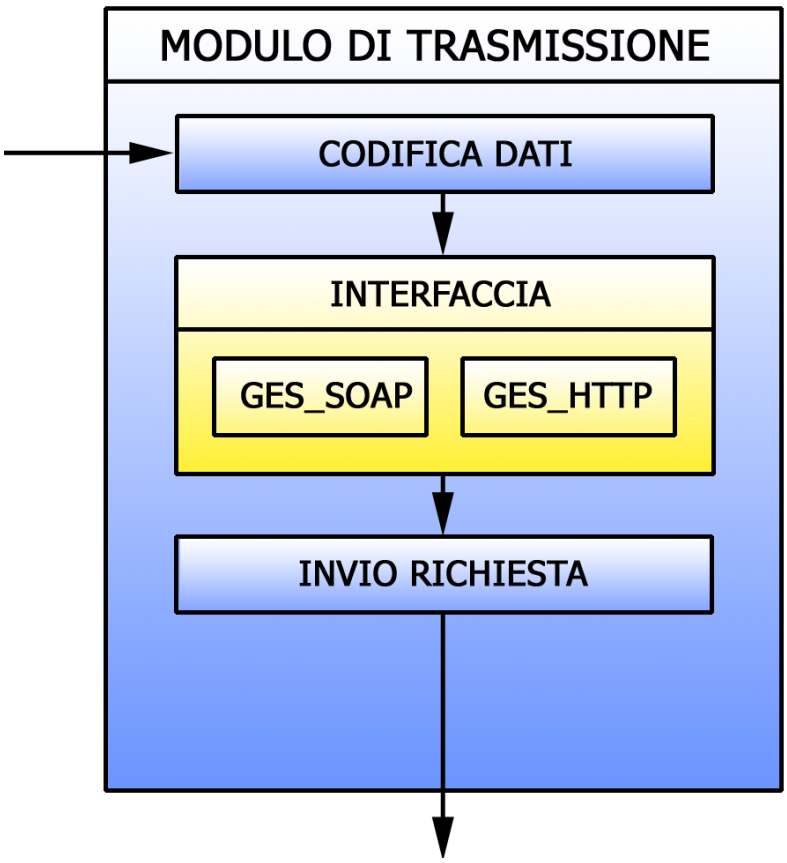


Figura 4.7: Integrazione dei Gestore SOAP e HTTP a mezzo di una interfaccia

```
1  public interface Interface_Transferable {
2    static final String NAMESPACE = "namespace";      // censurato
3    static final String SERVICE_NAME = "servicename"; // censurato
4    static final String METHOD_NAME = "methodname";   // censurato
5    /** Metodo di invio della richiesta */
6    abstract void sendMessage();
7    /** Metodo di prelievo della risposta */
8    abstract String getResponse();
9    /** Metodo di recupero informazioni */
10   abstract String getInfo();
11 }
```

Figura 4.8: Interfaccia *Interface_Transferable.java* utile ai gestori SOAP/HTTP

Come indicato nel listato in figura 4.8, l'invio e la ricezione dei messaggi possono essere realizzati implementando rispettivamente i metodi *sendMessage()* e *getResponse()*, infine, l'implementazione del metodo di debug *getInfo()* permette di realizzare la funzionalità descritta nel terzo punto della precedente lista. Un ulteriore dettaglio che si vuole sottolineare è la definizione nell'interfaccia di alcuni campi costanti tutti utilizzabili dal gestore SOAP e in parte da quello HTTP. Per motivi di sicurezza, in questa sede i loro valori veri sono stati mascherati, ma il significato è identico a quanto riportato nel paragrafo 4.1.1.

```
1  public class HandlerHTTP implements Interface_Transferable {
2    private static String myURL = null;
3    private String myProxy = null;
4    private int myPort = 0;
5    HttpPost httpPostRequest = null;
6    HttpClient httpclient = null;
7    HttpResponse hRes = null;
8
9    // Costruttore
10   public HandlerHTTP(String mURL, String data){...}
11   // Corpo della classe...
12
13   // Implementazione metodi d'interfaccia
14   public void sendMessage() {...}
15   public String getResponse() {...}
16   public String getInfo() {...}
17 }
```

Figura 4.9: Struttura base del Gestore HTTP

Esaminando l'organizzazione del Gestore HTTP rappresentata in figura 4.9, è possibile notare la presenza di campi privati ove verranno mantenuti l'URL di riferimento al servizio remoto erogato via web service (*myURL*)e le informazioni relative all'eventuale proxy di rete(*myProxy* e *myPort*). Le restanti tre variabili locali verranno usate rispettivamente per ospitare gli oggetti necessari a realizzare una richiesta tramite metodo POST (*httpPostRequest*), costituire un client HTTP (*httpclient*) e un conte-

nitore per la risposta ottenuta (*hRes*). Oltre il costruttore (descritto successivamente) e il corpo principale della classe, sarà presentato a titolo di esempio l'implementazione di un metodo dell'interfaccia per la classe *HandlerHTTP.class*.

Il costruttore riportato nel listato presente in figura 4.10 eseguirà alcune operazioni e inizializzerà i campi privati nonché quasi tutte le variabili locali principali della classe. Nel caso venga rilevata la presenza di un proxy di rete, i suoi dati verranno settati nell'oggetto *httpclient*. Immediatamente dopo (riga 14) verrà inizializzato l'oggetto di richiesta POST nel quale sarà inserita l'entità contenente i dati da trasmettere al server.

```
1  public HandlerHTTP(String mURL, String data)
2    {// Ricavo le informazioni del proxy
3    myProxy = android.net.Proxy.getDefaultHost();
4    myPort = android.net.Proxy.getDefaultPort();
5    myURL = mURL + SERVICE_NAME; // Realizzo l'url ove c'e' il servizio
6    httpclient = new DefaultHttpClient(); // Creo il client HTTP
7    // Gestisco l'eventuale Proxy sul client
8    if (myProxy != null) {
9      HttpHost proxy = new HttpHost(myProxy, myPort);
10     httpclient.getParams().setParameter(ConnRoutePNames.DEFAULT_PROXY,
11         proxy);
12   }
13   try { // Creo la richiesta Http/POST
14       httpPostRequest = new HttpPost(myURL + "/" + METHOD_NAME);
15   } catch (IllegalArgumentException iae) {iae.printStackTrace();}
16   try { // Aggiungo i dati da inviare
17       List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(1);
18       nameValuePairs.add(new BasicNameValuePair("str", data));
19       httpPostRequest.setEntity(new UrlEncodedFormEntity(nameValuePairs));
20   } catch (UnsupportedEncodingException uee) {uee.printStackTrace();}
21 }
```

Figura 4.10: Costruttore del Gestore HTTP

Per concludere, le implementazioni dei metodi dell'interfaccia relativi all'invio e alla ricezione dei messaggi non saranno indicati in questa sede,

```
1  public String getInfo() {
2    String str = "";
3    str += "HTTP Handler - INFO\n";
4    str += " PROXY [";
5    if (myProxy != null)
6      str += myProxy + ":" + Integer.toString(myPort) + "]";
7    else
8      str += "X:X]\n";
9    str += " STATUS [" + hRes.getStatusLine() + "]";
10   return str;
11 }
```

Figura 4.11: Implementazione del metodo *getInfo()* per il Gestore HTTP

ma si commenterà brevemente il metodo *getInfo()* riportato in figura 4.11.

Il metodo permette di ricavare alcune utili informazioni mantenute nel Gestore HTTP come il proxy, presentato nel formato *nome:porta*, nonché sullo stato HTTP della risposta ricevuta dal server. Ciò è possibile indagando l'oggetto che nel codice è indicato col nome *hRes*, ottenendo informazioni sullo stato di ritorno (tabella 4.1), come descritto nella RFC2616, sezione 6.1 che cito testualmente [For99]:

«The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:»

1xx	Informational	Request received, continuing process
2xx	Success	The action was successfully received, understood, and accepted
3xx	Redirection	Further action must be taken in order to complete the request
4xx	Client Error	The request contains bad syntax or cannot be fulfilled
5xx	Server Error	The server failed to fulfill an apparently valid request

Tabella 4.1: Status Code and Reason Phrase

4.2 Identificazione del dispositivo

Una nuova funzionalità che è stata introdotta nella ViDiApp riguarda l'identificazione del dispositivo mobile. La facoltà di conoscere quale sia lo smartphone che ha eseguito una certa richiesta di verifica di autenticità non è tanto necessaria alla realizzazione del servizio remoto (come non lo è per un modello base di Cliente/Servitore, paragrafo 4.1.2), quanto piuttosto alla sua salvaguardia. Per evitare, o meglio prevenire abusi intenzionali del servizio volti al suo danneggiamento, è utile conoscere l'identità del richiedente. Dato che al momento non sono previsti limiti d'uso del servizio, qualora ci si dovesse accorgere che un certo dispositivo esegue un numero di accessi giornalieri troppo elevato o con una concentrazione sospetta, diviene necessario intervenire, per esempio bloccandone le richieste.

L'argomento è stato molto dibattuto in azienda, sia per questioni tecniche e realizzative, sia per quanto riguarda la tutela della privacy dell'utente. Vari elementi potrebbero essere usati come discriminante, utili a realizzare un'identità digitale del dispositivo; fra questi è possibile elencare i seguenti: il numero telefonico del cellulare, l'IMSI o l'IMEI (paragrafo 2.1.3), il *MAC address* della scheda di rete.

Alcuni sono più lesivi della privacy di altri, ad esempio il numero telefonico è sicuramente più personale che non l'IMEI o il MAC address; alcuni sono più generici, altri prevedono contesti applicativi limitati. Per chiarire meglio quest'ultimo punto, è utile osservare che mentre tutti gli smartphone hanno una scheda SIM e dispongono di un numero cellulare poiché da essi discendono (paragrafo 1.1), non è detto che tutti posseggano una connettività di rete Wi-Fi e dunque di un MAC address. Viceversa, usare l'applicativo ViDiApp su un tablet, significa disporre di una piattaforma ove la scheda di rete sarà sicuramente presente, non è invece scontata la possibilità di accedere alla rete cellulare e dunque l'esistenza di un IMEI/IMSI.

Effettuando ulteriori ricerche, è stato individuato anche un altro parametro utile all'identificazione del dispositivo: ANDROID_ID. Tale ele-

mento è un valore a 64 bit generato in modo random al primo avvio e resta sempre lo stesso per tutto il tempo di vita del dispositivo, salvo operazioni di reset delle impostazioni o altre operazioni critiche effettuate sul sistema. Dopo aver eseguito alcuni esperimenti molto incoraggianti con ANDROID_ID si è deciso di abbandonare questo sistema di identificazione per un insieme di motivi. Innanzi tutto, il valore a 64 bit generato è casuale e non vi sono garanzie sulla sua unicità: anche se ciò non costituisce un problema stringente per quanto discusso in ViDiTrust, è sicuramente un difetto inaccettabile per la realizzazione di una identità digitale. Inoltre, il vantaggio di essere un parametro molto generale (è definito per tutte le piattaforme Android) è minato da alcuni bug noti, ad esempio per sistemi che usano la versione 2.2, e pertanto non è usabile.

In conclusione, si è optato per una soluzione mista capace di garantire un buon grado di generalità cercando di preservare il più possibile la privacy. Scartando il numero telefonico e l'ANDROID_ID, per identificare il dispositivo si sono usati contemporaneamente l'IMEI, l'IMSI e il MAC address. Come già osservato, non è detto che tutti questi parametri siano ricavabili dai vari device in uso, ma è possibile dare loro una priorità e in base a essa riferire il primo elemento fra quelli individuati.

Prima di esaminare parte del codice relativo allo specifico intervento, in figura 4.12 viene riportata una visione più dettagliata del Modulo di Inizializzazione ove è stato introdotto il blocco di *Identificazione del Dispositivo* (in giallo).

Le modifiche al codice Java faranno riferimento a svariate classi, alcune afferenti anche ad altri moduli principali, come ad esempio il Modulo di Trasmissione per quanto riguarda il blocco di *Codifica Dati*, paragrafo 4.1. Gli spezzoni di codice presentati al lettore favoriranno pertanto un percorso logico utile alla comprensione dell'intervento senza la pretesa di offrire una descrizione puntuale e completa della soluzione tecnica realizzata.

Iniziamo l'esposizione dell'intervento riportando la struttura base della classe che si occupa di realizzare un singleton, creato e inizializzato nel blocco *Init Singleton*. Come descritto nel paragrafo 3.2, lo scopo principale del blocco e dell'oggetto che restituisce è quello di costituire un nucleo

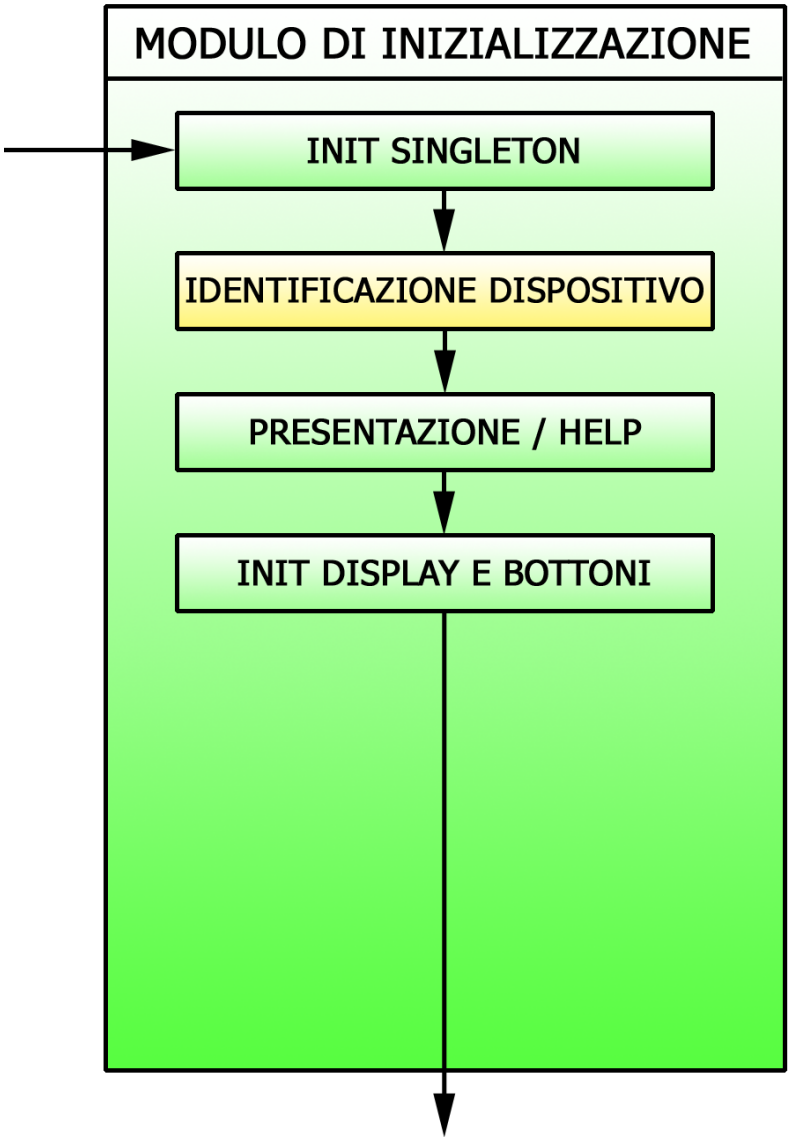


Figura 4.12: Modulo di Inizializzazione con blocco di *Identificazione del Dispositivo*

utile a mantenere aggiornati, disponibili e integri i parametri di sistema e le grandezze d'uso di tutti gli altri blocchi funzionali.

Nel codice riportato in figura 4.13 è possibile notare l'implementazione Java del *pattern* relativo a un Singleton. Caratteristiche inconfondibili sono il costruttore privato (che descriveremo in parte in seguito), il campo statico ove conservare l'istanza dell'oggetto (*mIstance*) e il metodo "getter"

```
public class InitSingleton
{
  static InitSingleton mInstance=null; // Istanza del singleton
  // Campi di gestione/mantenimento dei parametri
  // utili agli altri Moduli/Classi
  private String _manufacturer; // Costruttore del dispositivo
  private String _model;        // Modello del dispositivo
  private String _osVersion;    // Versione di Android
  private String _imei; // IMEI
  private String _imsi; // IMSI
  private String _maca; // Mac Address

  private InitSingleton() {...} // Costruttore privato

  // Metodo "getter" statico: ritorna sempre la stessa istanza
  public static InitSingleton getAttivation()
  {
    synchronized (mLock)
    {if (mInstance == null) mInstance = new InitSingleton();
     return mInstance;
    }
  }
}
```

Figura 4.13: Struttura base della classe principale relativa al blocco *Init Singleton*

(*getAttivation()*) anch'esso statico utile a creare e inizializzare il singleton, unico metodo capace di accedere al costruttore.

L'aspetto più interessante riguarda l'introduzione di alcuni campi specifici, utili a conservare le informazioni relative all'identificazione del dispositivo: *_imei*, *_imsi* e *_maca*, rispettivamente usati per l'IMEI, l'IMSI e il MAC address.

I campi sono definiti come stringhe, come la maggior parte di quelli chiamati in causa durante l'interazione fra la ViDiApp e il server di autenticazione. Come vedremo, ciò è dovuto al fatto che la comunicazione dei parametri fra le parti è realizzata ad alto livello, usando XML e non tramite formati binari. A titolo di esempio sono stati riportatati anche altri

campi inclusi nel singleton, utilizzati a fini statistici: *_manufacturer*, *_model* e *_osVersion*. Con essi è possibile mantenere e trasmettere informazioni relative al costruttore e al modello del dispositivo, nonché la versione del sistema operativo Android installato.

```
1  import android.os.Build;
2  public class InitSingleton
3  {
4    private InitSingleton()  // Costruttore privato (frazione)
5    {// Inizializzazione diretta dei parametri sempre ricavabili
6      _manufacturer = Build.MANUFACTURER;
7      _model = Build.MODEL;
8      _osVersion = Build.VERSION.RELEASE;
9      _imei = ""; // IMEI
10     _imsi = ""; // IMSI
11     _maca = ""; // Mac Address
12   }
13 }
```

Figura 4.14: Costruttore privato del singleton (frazione)

In figura 4.14 è invece riportato una frazione del codice relativo al costruttore della classe *InitSingleton.java* ove è possibile notare l'inizializzazione dei parametri discussi in precedenza. Alcuni di essi possono essere ricavati immediatamente accedendo alle librerie di sistema offerte da Android come per esempio quelle appartenenti al package *android.os*, gli altri dovranno essere determinati e impostati in modo opportuno nell'oggetto singleton. Allo scopo sono previsti i necessari metodi di "set" come ad esempio *setIMEI()*, ma anche tutti i metodi di "get" utili a ricavare dall'oggetto le grandezze determinate e che si vogliono condividere fra le classi e i moduli del progetto: *getManufacturer()* e *getIMEI()* sono degli esempi (figura 4.15).

Dopo aver descritto come realizzare l'identificazione del dispositivo, i parametri utili allo scopo e come vengono gestiti nella classe *InitSingleton.java*, analizziamo il codice relativo a una delle attività principali dell'applicazione Android. Del concetto di Activity abbiamo discusso nel pa-

```
1 // Alcuni dei metodi di get/set della classe InitSingleton.java
2 public String getManufacturer() {
3   return _manufacturer;}
4
5 public String getIMEI() {
6   return _imei;}
7
8 public void setIMEI(String imei){
9   _imei=imei;}
```

Figura 4.15: Alcuni metodi di set/get estratti della classe *InitSingleton.java*

ragrafo 1.1.2.1 del primo capitolo e ciò ci consente di comprendere meglio il codice riportato in figura 4.16, ove viene presentato un piccolo estratto del file *Activity_Main.java*.

Dall'analisi del listato è possibile notare la presenza di alcuni degli elementi tipici di un codice sviluppato per piattaforma Android. In effetti, la classe oggetto di analisi estende Activity, utile a realizzare uno dei principali componenti Android; esiste poi il metodo di callback onCreate() che contiene la maggior parte delle operazioni di inizializzazione e che dispone in questo caso di un parametro di tipo Bundle utile a gestire lo stato dell'activity. Infine, nei commenti riportati nelle righe 14 e 15, si legge di impostare un particolare permesso nel file *AndroidManifest.xml* dell'applicazione: nel mondo Android, per poter usare alcune risorse, occorre esplicitamente indicarlo attraverso i relativi permessi.

Esaminando il corpo del metodo onCreate(), dopo aver invocato il costruttore della classe Activity, viene definito e inizializzato un oggetto capace di realizzare un'istanza del gestore telefonico (*tm*). Ciò è necessario per poter ricavare i valori di IMEI e IMSI per mezzo dell'invocazione dei metodi *getDeviceId()* e *getSubscriberId()*. Per necessità aziendali, in caso uno degli elementi non potesse essere ricavato, il relativo parametro-stringa verrà impostato a stringa vuota.

Le ultime operazioni indicate permetteranno di salvare i valori di questi parametri nell'oggetto *init*, l'istanza dell'oggetto singleton, per mezzo

```java
public class Activity_Main extends Activity{ // Estratto

  @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Definisco il gestore telefonico
      TelephonyManager tm = (TelephonyManager)
           getSystemService(Context.TELEPHONY_SERVICE);
      String imei = tm.getDeviceId();        // Ricavo IMEI
      if (imei==null) imei=""; // Impongo stringa vuota se null
      String imsi = tm.getSubscriberId();    // Ricavo IMSI
      if (imsi==null) imsi=""; // Impongo stringa vuota se null

      // Definisco il gestore wifi - Occorre il permesso
      // android.permission.ACCESS_WIFI_STATE in AndroidManifest.xml
      WifiManager wifim = (WifiManager)
           getSystemService(Context.WIFI_SERVICE);
      WifiInfo info = wifim.getConnectionInfo();
      String maca = info.getMacAddress(); // Ricavo Mac Address
      if (maca==null) maca=""; // Impongo stringa vuota se null

      // Setto i parametri ricavati in init
      // NB 'init' e' l'istanza del singleton
      init.setIMEI(imei);
      init.setIMSI(imsi);
      init.setMACA(maca);
  }
}
```

Figura 4.16: Metodo di callback onCreate() dell'attività Activity_Main

degli opportuni metodi di set in esso definiti, come indicato in figura 4.15.

Per maggiore chiarezza aggiungiamo due piccole note, la prima di carattere generale. Spesso nel codice si userà l'oggetto *init* che per motivi di spazio non sarà sempre esplicitamente dichiarato e inizializzato nelle figure relative ai listati: da ora in avanti può essere considerato un'istanza della classe *InitSingleton.java* precedentemente esaminata. L'altra osserva-

zione che si vuole sottoporre al lettore riguarda l'invocazione e l'uso dei gestori, nello specifico di quello telefonico e di quello Wi-Fi appena trattati. Tali gestori sono messi a disposizione da Android e ne abbiamo discusso nell'omonimo paragrafo (1.1.2), ove è possibile rintracciarli nello strato Application Framework presente in figura 1.3.

Per concludere, commenteremo due ulteriori spezzoni di codice appartenenti alla classe adibita alla pacchettizzazione e alla codifica dei dati in formato XML.

```
1  public class XMLcs {
2    String _manufacturer; // Costruttore del dispositivo
3    String _model; // Modello del dispositivo
4    String _osVersion; // Versione Android OS
5    String _idDevice; // IDENTITA' DEL DISPOSITIVO
6
7    public Obj_XMLClientStream(...){
8      _manufacturer=init.getManufacturer(); //
9      _model=init.getModel();
10     _osVersion=init.getOsVersion();
11   // Priorita' per l'IDENTITA' DEL DISPOSITIVO:
12   // 1. IMEI - 2. IMSI - 3. Mac Address - 4. Stringa vuota
13   if(!(init.getIMEI()== "")) _idDevice="IMEI-" + init.getIMEI();
14   else if (!(init.getIMSI() == "")) _idDevice="IMSI-"+init.getIMSI();
15   else if (!(init.getMACA() == ""))_idDevice="MACA-"+init.getMACA();
16   else _idDevice="";
17   }
18   // Metodo capace di ricavare da un oggetto XMLcs, una stringa
19   // formattata secondo un preciso formato XML
20   public String stringToXml() {...}
21 }
```

Figura 4.17: Struttura base del "codificatore XML"

In figura 4.17 è riportato un frammento di codice relativo alla classe *XMLcs.java* in cui si possono notare tre sezioni principali. La parte iniziale prevede la definizione di campi simili a quanto già presentato nella classe *InitSingleton.java*: essi sono un sottoinsieme limitato di quelli mantenuti

nell'oggetto *init*, più precisamente sono i soli campi utili alle finalità della classe ed è bene disporne localmente.

La seconda parte presenta una frazione del costruttore di classe: i campi vengono inizializzati prelevandoli dal Singleton, ma è in questa sede che viene deciso tramite una specifica politica quale parametro identificativo utilizzare. Come è possibile osservare nel codice, il campo *_idDevice* sarà impostato col valore IMEI, se non disponibile con l'IMSI, infine col MAC address. Se non è possibile determinare alcuno di questi elementi, allora il campo *_idDevice* verrà impostato a stringa vuota. Si segnala una piccola nota tecnica: per evitare problemi di compatibilità con le versioni più vecchie della piattaforma Android, si evita di usare il metodo *isEmpty()* per verificare la presenza di stringhe vuote, ma si ricorre a dei confronti diretti con "".

```
1  // Costruisco e restituisco una stringa con un preciso formato XML
2  public String stringToXml() {
3    String xml = "<cs>\n"
4      + " <Manufacturer>" + _manufacturer + "</Manufacturer>\n"
5    + " <Model>" + _model + "</Model>\n"
6    + " <OsType>" + _osType + "</OsType>\n"
7    + " <OsVersion>" + _osVersion + "</OsVersion>\n"
8    + " <IDdevice>" + _idDevice + "</IDdevice>\n"
9    + "</cs>";
10   return xml;
11 }
```

Figura 4.18: Frammento del metodo *stringToXml()* della classe *XMLcs.java*

Infine, la terza parte presenta il metodo *stringToXml()* il cui codice è in parte incluso in figura 4.18. Utile a organizzare il contenuto della classe in un formato XML da restituire come stringa, il metodo inserisce i vari campi determinati in precedenza fra *tag* dall'ovvio significato.

4.3 Localizzazione del dispositivo

Un'altra importante funzionalità aggiunta all'applicativo ViDiApp riguarda la capacità di determinare la posizione del dispositivo e comunicarla ai server ViDiTrust. Come indicato nel paragrafo 3.2.1, disporre di tali informazioni è utile per realizzare importanti servizi quali la tracciabilità dei prodotti, una mappatura delle aree a maggiore contraffazione, il controllo della diffusione di ViDiApp.

Ancora una volta si ha a che fare con un duplice problema di ordine tecnico/implementativo e di tutela della privacy. Nel paragrafo 1.3 sono state analizzate le varie possibilità di localizzazione per un dispositivo mobile, fondamentalmente tramite GPS, via rete cellulare o grazie a Internet, per mezzo della mappatura degli Access Point (AP). Osservando che uno degli scopi principali di ViDiApp è quello di autenticare dei prodotti in loco, in autonomia e prima dell'acquisto, è plausibile immaginare come comune scenario applicativo quello in cui un acquirente si trovi in un negozio, al chiuso. Operare per mezzo del GPS risulterebbe essere difficoltoso (non si è a vista cielo), dispendioso per quanto concerne i consumi di batteria, probabilmente lento nell'acquisizione della posizione e l'utente potrebbe dimostrarsi diffidente o pigro nell'attivare tale funzionalità. Inoltre, il grado di precisione che si desidera ottenere vuole essere deliberatamente basso (dell'ordine dei centinaia di metri), proprio per non risultare troppo invadenti e salvaguardare il più possibile la privacy dell'utente.

In merito a quanto osservato nel paragrafo 1.3.3, sempre a causa di problemi legati alla privacy, alle possibili incompatibilità fra le legislazioni di Stati differenti, nonché per l'incognita sulla corretta manutenzione della mappatura degli AP, anche una localizzazione basata sulla rete Internet non è sembrata essere la soluzione ideale.

Da quanto fin'ora illustrato, l'approccio più indicato per realizzare il sistema di localizzazione da inserire nella ViDiApp è parso essere quello della rete cellulare e anche l'azienda si è dimostrata concorde nell'adottare questa linea di sviluppo.

Ristretto il campo d'indagine, l'analisi si è concentrata ancora una vol-

ta sulle capacità offerte dalla piattaforma Android in merito alla localizzazione. Questa sezione delle API è ben curata e opera principalmente tramite la classe *LocationManager.java* del package *android.location*, anch'essa basata sull'uso di un gestore che rimanda allo strato Application Framework dell'architettura Android (1.3). Senza introdurre i dettagli, ci limitiamo a segnalare che tramite l'uso di questa risorsa è possibile scegliere una localizzazione basata sul GPS o sulla rete impostando un opportuno *provider*, rispettivamente il GPS_PROVIDER e il NETWORK_PROVIDER.

Come discusso nel paragrafo 1.3.2, operare via rete cellulare significa adottare le tecniche di localizzazione Cell-based o Triangulation-based. A causa della scarsa precisione necessaria e desiderata, si è scelto di operare con la prima modalità, supportata in "buona parte" dalla piattaforma Android. Anticipando alcuni dettagli che vedremo approfonditamente nei commenti al codice, determinare la posizione del dispositivo in base alla cella a cui è connesso è possibile, ma non si ottengono direttamente le informazioni attese, ovvero la sua posizione espressa come coppia di coordinate latitudine e longitudine. È pertanto necessario ricorrere a un intermediario che traduca opportunamente i dati ricavati dalla cella telefonica.

In figura 4.19 è riportata la nuova forma assunta dal Modulo di Inizializzazione dopo l'introduzione dei blocchi dedicati al recupero delle informazioni LAC e Cell-ID e alla loro traduzione in coordinate geografiche: *Localizzazione GSM (Lac/Cell-ID)* e *Risoluzione Localizzazione*.

Prima di commentare il codice sviluppato, si avvisa il lettore che la maggior parte dei listati di seguito riportati saranno introdotti in classi e moduli già ampiamente commentati nel paragrafo precedente, pertanto il livello di dettaglio descrittivo verrà adeguatamente ridotto.

Innanzi tutto esaminiamo il codice relativo alla classe *InitSingleton.java* riportato in figura 4.20.

Nella classe sono stati aggiunti i campi stringa *_lat* e *_lon* utili per ospitare rispettivamente i dati relativi alle coordinate di latitudine e longitudine. Come precedentemente osservato, anche in questo caso i campi che non possono essere inizializzati direttamente verranno impostati con

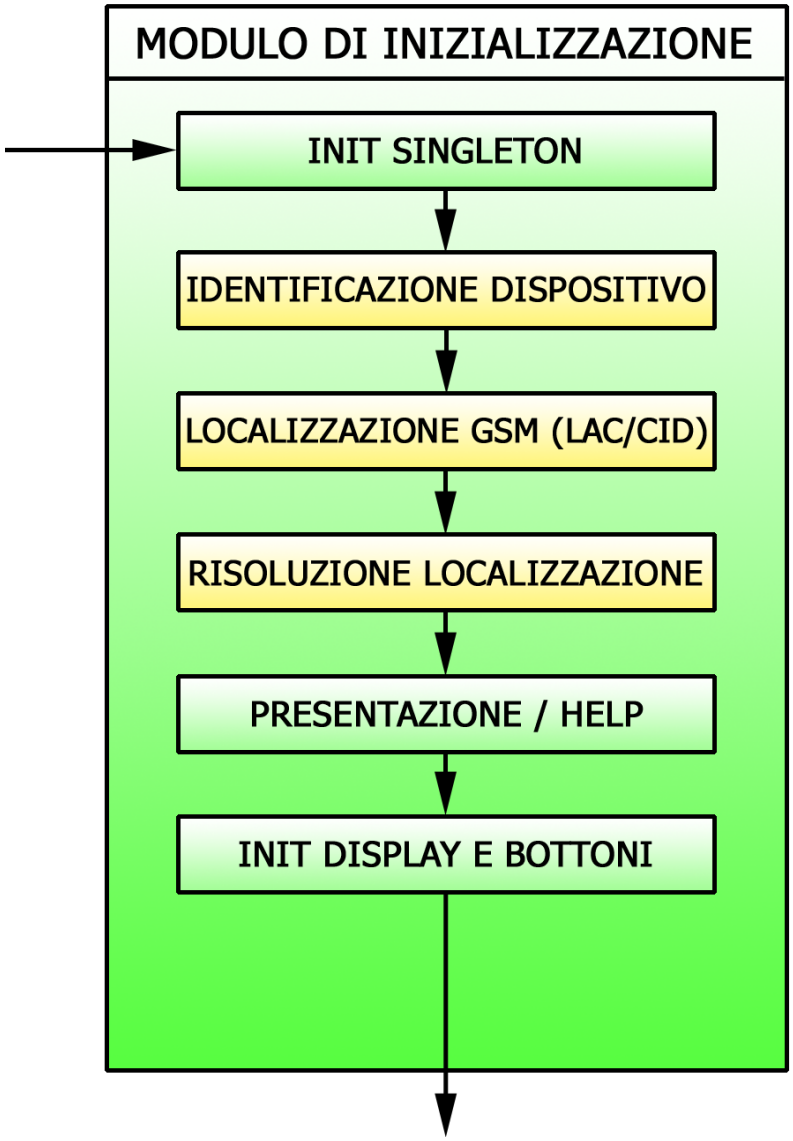


Figura 4.19: Blocchi *Localizzazione GSM (Lac/Cell-ID)* e *Risoluzione Localizzazione* inseriti nel Modulo di Inizializzazione

stringhe vuote dal costruttore, come indicato in figura 4.21. Sempre nella classe *InitSingleton.java* sono stati introdotti i relativi metodi di get/set: *getLat()*, *getLon()*, *setLat()* e *setLon()*, ma non sono stati riportati in figura.

Passando a esaminare gli interventi eseguiti nell'attività Activity_Main (figura 4.22), si può notare che nel metodo onCreate() sono stati

```
1 public class InitSingleton
2 {
3   static InitSingleton mInstance=null; // Istanza del singleton
4   // Campi di gestione/mantenimento dei parametri
5   // utili agli altri Moduli/Classi
6   private String _lat; // Latitudine
7   private String _lon; // Longitudine
8   // Altro codice...
9 }
```

Figura 4.20: Modifiche alla classe *InitSingleton.java* finalizzate alla localizzazione

```
1 // Costruttore privato della classe InitSingleton.java (frazione)
2 private InitSingleton()
3   {
4    // Inizializzazione dei parametri
5    _lat = ""; // Latitudine
6    _lon = ""; // Longitudine
7    // Altro codice...
8   }
```

Figura 4.21: Modifiche al costruttore privato della classe *InitSingleton.java*

introdotti due parametri interi inizializzati al valore "-1" (righe 10 e 11).

Ricordando brevemente quanto riportato nel paragrafo 1.3.2, si ha che ciascuna rete di un qualsiasi operatore è divisa in aree che prevedono un codice LAC (Local Area Code), inoltre, ciascuna stazione base posta all'interno di una certa area (cella) può essere identificata attraverso un ulteriore codice, il Cell-ID. I parametri interi *_lac* e *_cellID* riportati nel listato servono per mantenere tali informazioni che possono essere ricavate operando sull'oggetto *gsm_location*. L'oggetto in questione discende da un'istanza del gestore telefonico *tm* su cui è stato usato il metodo *getCellLocation()*. È utile notare che per poter realizzare e lavorare con un oggetto di classe *GsmCellLocation* è necessario fornire il permesso ACCESS_COARSE_LOCATION, sempre da specificare nel file *AndroidManifest.xml*. Infine, se l'oggetto *gsm_location* non può essere realizzato perché

```
1  public class Activity_Main extends Activity{ // Estratto
2    @Override
3    public void onCreate(Bundle savedInstanceState) {
4      super.onCreate(savedInstanceState);
5      // Definisco il gestore telefonico
6      TelephonyManager tm = (TelephonyManager)
7           getSystemService(Context.TELEPHONY_SERVICE);
8      // La prima fase della localizzazione via rete cllulare GSM
9      // prevede la determinazione degli interi cellID e lac
10     int cellID = -1; // Inizializzo cellID
11     int lac = -1;    // Inizializzo lac: Location Area Code
12     GsmCellLocation gsm_location;
13     // Occorre il permesso
14     // android.permission.ACCESS_COARSE_LOCATION
15     gsm_location = (GsmCellLocation) tm.getCellLocation();
16     if (gsm_location != null) {
17       cellID = gsm_location.getCid(); // Ricavo cellID
18       lac = gsm_location.getLac();    // Ricavo lac
19     }
20   }
21 }
```

Figura 4.22: Modifiche al metodo onCreate() di Activity_Main

ad esempio lo smartphone non ha campo oppure è in "modalità aereo" per la quale tutte le connettività vengono spente, il controllo effettuato alla riga 17 permetterà di mantenere i parametri interi fissati al valore di inizializzazione "-1".

Con questo si conclude la descrizione del blocco Localizzazione GSM (Lac/Cell-ID).

Scopo del blocco di Risoluzione Localizzazione è quello di tradurre i valori di LAC e Cell-ID in longitudine e latitudine e per farlo è necessario ricorrere a un intermediario. Sarebbe possibile realizzare una funzionalità di conversione che operi localmente al dispositivo mobile, ma è più conveniente utilizzare servizi presenti in rete, visto che l'applicazione ViDiApp ha comunque bisogno di accedervi per realizzare le proprie funzionali-

tà di autenticazione. Per raggiungere l'obiettivo esistono varie soluzioni, ad esempio è possibile interrogare il sito `http://www.cellid.eu/` sfruttandone le API fornite, oppure inoltrando una particolare richiesta a `http://www.google.com/glm/mmap`, ovvero a Google Maps. Nell'applicazione è stato implementato questo secondo metodo e mentre in origine la gran parte del codice era inglobata in Activity_Main, successivamente il maggior numero delle funzionalità di rete è stato trasferito in una libreria dedicata di cui parleremo in parte in questo stesso paragrafo.

```
1  if (gsm_location != null) {
2    cellID = gsm_location.getCid(); // Ricavo cellID
3    lac = gsm_location.getLac();    // Ricavo lac
4    // Risolvo e imposto nell'oggetto 'init' i valori
5    // di latitudine e longitudine
6    if (LibConnect.LocationRequest(cellID, lac)) {
7      init.setLat(LibConnect.latitudeSTR);
8      init.setLon(LibConnect.longitudeSTR);
9    }
10 }
```

Figura 4.23: Integrazione nel metodo onCreate() di Activity_Main

Al momento riportiamo un nuovo spezzone di codice che completa quello presentato in figura 4.22 e rende possibile la traduzione dei parametri LAC e Cell-ID nelle coordinate latitudine e longitudine. In figura 4.23 si può notare come ciò venga realizzato per mezzo dell'invocazione del metodo pubblico e statico *LibConnect.LocationRequest()* a cui vengono passati i parametri determinati poco sopra. Se la risoluzione va a buon fine, i valori di longitudine e latitudine saranno salvati in campi pubblici all'interno della classe *LibConnect.java*, quindi recuperati e impostati nei campi loro dedicati presenti nell'oggetto *init*.

Per comprendere meglio quanto descritto, in figura 4.23 proponiamo parte del codice presente nella libreria di connessione utile a realizzare le principali funzionalità di rete (4.24). La stessa libreria verrà nuovamente commentata nel paragrafo 4.4, relativo alle funzionalità di controllo del-

la disponibilità della connessione e della presenza in rete del servizio di autenticazione.

```
1  public class LibConnect {
2    public static int latitudeINT;     // Intero per la latitudine
3    public static int longitudeINT;    // Intero per la longitudine
4    public static String latitudeSTR;  // Stringa per la latitudine
5    public static String longitudeSTR; // Stringa per la longitudine
6    // Traduce cellID e lac in latitudine e longitudine
7    public static boolean LocationRequest(int cellID, int lac) {...}
8    // Scrive la richiesta di risoluzione in Google Maps
9    private static void writeRequest(OutputStream out, int cellID,
10       int lac) {...}
11   // Converte i campi interi in stringhe e che poi li salva
12   private static void convert() {
13     latitudeSTR = String.valueOf((float) latitudeINT / 1000000);
14     longitudeSTR = String.valueOf((float) longitudeINT / 1000000);
15     return;
16   }
17 }
```

Figura 4.24: Struttura base della libreria di connessione *LibConnect.java* (frazione)

Come già fatto notare, all'interno della libreria sono definiti quattro campi pubblici e statici (righe 2-5) così da poter essere acceduti senza necessità di definire i metodi get. Due campi sono dedicati alla rappresentazione intera delle coordinate di latitudine e longitudine, gli altri due sono usati per disporre di tali informazioni in forma di stringa.

L'unico metodo pubblico esposto è *LocationRequest()* invocato in precedenza nel modo indicato in figura 4.23. Internamente al metodo verranno chiamati quelli privati di *writeRequest()* utile a interrogare il servizio Google Maps e *convert()*, capace di trasformare i valori interi delle coordinate geografiche ottenute, successivamente salvate nei campi loro dedicati.

Per concludere la trattazione della libreria, descriviamo in modo conciso il metodo *LocationRequest()* riportato in figura 4.25. Innanzi tutto verrà inizializzato il risultato di ritorno a falso, così se si dovesse verificare un qualsiasi problema, il metodo può comunicarlo a chi lo invoca. Fissata

```
1  public static boolean LocationRequest(int cellID, int lac) {
2
3    boolean result = false;
4    String urlmmap = "http://www.google.com/glm/mmap";
5
6    try {
7      URL url = new URL(urlmmap);
8      URLConnection conn = url.openConnection();
9      HttpURLConnection httpConn = (HttpURLConnection) conn;
10     httpConn.setRequestMethod("POST"); httpConn.setDoOutput(true);
11     httpConn.setDoInput(true); httpConn.connect();
12     // Apro uno stream di output sulla connessione HTTP
13     OutputStream outputStream = httpConn.getOutputStream();
14     writeRequest(outputStream, cellID, lac); // Scrivo sull'out-stream
15     // Apro uno stream di input sulla connessione HTTP
16     InputStream inputStream = httpConn.getInputStream();
17     DataInputStream dataInputStream = new DataInputStream(inputStream);
18
19     dataInputStream.readShort(); // scarto...
20     dataInputStream.readByte();  // scarto...
21     int code = dataInputStream.readInt(); // Leggo un intero
22     if (code == 0) {
23     // Leggo due interi, rispettivamente lat. e lon.
24     latitudeINT = dataInputStream.readInt();
25     longitudeINT = dataInputStream.readInt();
26     convert(); // Imposto i campi stringa della classe
27     result = true;
28     }
29   } catch (IOException e) {e.printStackTrace();}
30   return result;
31 }
```

Figura 4.25: Metodo *LocationRequest()* della classe *LibConnect.java*

la stinga *urlmmap* all'url di riferimento, con un insieme di istruzioni viene realizzata e aperta una connessione HTTP che lavora con il metodo di request POST. Su tale connessione verranno aperti gli stream di output e input, rispettivamente alle righe 11 e 14.

Dopo aver scritto i parametri *cellID* e *lac* sull'output per mezzo del metodo *writeRequest()*, si inizia a leggere la risposta attesa sullo stream dati costruito su quello di input. Conoscendo il formato della risposta, scartati alcuni valori inutili, si riesce a prelevare i valori interi che rappresentano la latitudine e la longitudine. Infine viene invocato il metodo *convert()* descritto in precedenza e restituito il risultato vero, a significare che la procedura si è conclusa correttamente.

```
1  public class XMLcs {
2    // Coordinate geografiche
3    String _geoLat; // Latitudine
4    String _geoLon; // Longitudine
5
6    public Obj_XMLClientStream(...){
7    // Imposto i valori stringa di latitudine e longitudine
8    // prelevandoli da 'init'
9      _geoLat=init.getLat();
10     _geoLon=init.getLon();
11   }
12 }
```

Figura 4.26: Modifiche ulteriori al "codificatore XML"

Analogamente a quanto descritto nel paragrafo precedente, anche in questo caso occorre trasferire i valori di latitudine e longitudine ricavati alla classe adibita alla pacchettizzazione e alla codifica dei dati in formato XML.

In figura 4.26 è presentato parte del codice da aggiungere alla classe *XMLcs.java*. La parte iniziale prevede la definizione di campi stringa *_geoLat* e *_geoLon*, successivamente è indicato un aggiornamento del costruttore di classe che permette di inizializzare tali campi prelevandone i valori dall'oggetto *init*, espressione del Singleton.

Infine è stato aggiornato anche il metodo *stringToXml()* come indicato in figura 4.27: i campi relativi alle coordinate geografiche vengono anche in questo caso incapsulati in opportuni *tag*.

```
// Costruisco e restituisco una stringa con un preciso formato XML
public String stringToXml() {
  String xml = "<cs>\n"
  // Altro codice...
  + " <GeoLat>" + _geoLat + "</GeoLat>\n"
  + " <GeoLon>" + _geoLon + "</GeoLon>\n"
  + "</cs>";
  return xml;
   }
```

Figura 4.27: Modifiche ulteriori al metodo *stringToXml()* della classe *XMLcs.java*

4.4 Funzionalità di rete

L'accesso al servizio remoto di autenticazione da parte del software ViDiApp prevede necessariamente l'accesso alla rete Internet, come specificato nei requisiti 3.1.1 e come si è detto nei precedenti paragrafi, in particolare in quello dedicato al Modulo di Trasmissione (4.1) e in quello relativo alla localizzazione del dispositivo, nel blocco di Risoluzione Localizzazione (4.3).

In questa sezione verranno presentate alcune funzionalità di controllo che possono essere categorizzate in "locali" e "remote". Le prime prevedono la verifica della disponibilità di connessione da parte del dispositivo e dunque si limitano a verificare la presenza delle risorse hardware necessarie e del loro stato di funzionamento. Della categoria remota fanno invece parte l'insieme dei metodi che devono accedere a Internet, in particolare al Web, per indagare sulla disponibilità del servizio di autenticazione.

L'insieme di queste funzionalità sono raccolte nella libreria *LibConnect.java* che accoglie tutti gli elementi che intervengono in rete, come pure i metodi utili alla risoluzione della localizzazione descritti nel paragrafo precedente.

Introdotte le funzioni di controllo, per mezzo di alcune figure verrà mostrata l'evoluzione dei moduli in cui sono state inserite e grazie ad alcuni brani di codice si illustrerà al lettore il loro specifico utilizzo.

A termine del paragrafo verranno discussi e commentati ulteriori metodi della libreria realizzati con lo scopo di rintracciare il web service o il web server a cui indirizzare la richiesta di autenticazione.

4.4.1 Controllo della connessione

Avere la possibilità di accedere a Internet con un dispositivo mobile può ridursi alla seguente semplificazione: disporre di una connessione di tipo Wi-Fi oppure abilitare un traffico dati sulla rete telefonica offerta dal proprio gestore. Non tutti gli smartphone prevedono entrambe le possibilità, ma è nostro interesse verificare la disponibilità di almeno una di esse. Una volta determinato questo aspetto, occorre stabilire anche che la risorsa utile alla realizzazione della connessione sia in effetti attiva e pronta a essere utilizzata da ViDiApp. Se l'utente possiede un cellulare avanzato che offre ambedue i tipi di connettività ma imposta la "modalità aereo" oppure, per esempio, ha disabilitato il traffico dati GSM per questioni economiche e non vi sono connettività wireless in zona, il fatto di disporre delle risorse hardware non è sufficiente a stabilire la connessione richiesta.

Allo scopo verranno presentate due funzionalità molto simili, una dedicata al *controllo di connessione Wi-Fi*, l'altra relativa al *traffico dati su rete cellulare*. Fra gli elementi comuni è lecito osservare che entrambe ritornano un valore booleano in base al quale è possibile stabilire la disponibilità e lo stato della connettività, ma di questo parleremo nei commenti al codice. Infine, proprio per la loro grande somiglianza, verrà presentato il listato di uno solo dei due metodi (quello Wi-Fi, in figura 4.28), l'altro verrà esposto per differenza, illustrando le piccole variazioni in cui si distingue.

Il metodo *wifiIsOk()* prevede il passaggio del parametro *ctx* contenente le informazioni di ambiente relative all'applicazione e ciò è necessario per poter posizionare il metodo nella libreria *LibConnect.java*. Gli elementi interni che commenteremo hanno infatti bisogno di accedere al contesto per poter essere definiti e realizzati. Il metodo è pubblico e statico così da poter essere invocato dove necessario senza dover istanziare oggetti dedi-

```
1  public static boolean wifiIsOk(Context ctx) {
2    // Occorre permesso: android.permission.ACCESS_NETWORK_STATE
3    ConnectivityManager cm = (ConnectivityManager)
4       ctx.getSystemService(Context.CONNECTIVITY_SERVICE);
5    if (cm == null)
6      return false; // Non riesco a interrogare il dispositivo
7    if (!cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI)
8        .isAvailable())
9        return false; // Tipo di connettivita' non 'presente'
10     else
11       return cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI)
12           .isConnected();
13 }
```

Figura 4.28: Metodo *wifiIsOk()* della classe *LibConnect.java*

cati. Come detto precedentemente, si ha un ritorno di tipo booleano, di valore vero solo se la connessione Wi-Fi è disponibile e abilitata.

Istanziare l'oggetto *cm* prevede l'uso del permesso ACCESS_NETWORK_STATE che dovrà essere dichiarato nell'apposita sezione del file *AndroidManifest.xml*. Se il gestore della connettività viene correttamente preparato, lo si interroga dapprima col metodo *isAvailable()* e se il responso è positivo, si ritorna il valore restituito da *isConnected()*. Il primo metodo verifica che la risorsa sia disponibile (installata e attiva), ovvero che l'antenna e la circuiteria dedicata siano presenti e alimentate, il metodo successivo si accerta che la connessione sia pronta all'uso e dunque si possegga un indirizzo IP valido e tutto il necessario per poter accedere a Internet.

Esiste poi un metodo analogo che si occupa di indagare sullo stato della connessione mobile e cioè della possibilità di instaurare un traffico dati attraverso la linea telefonica cellulare. Il nome di tale funzionalità è *mobileIsOk()* e opera allo stesso modo di quella appena descritta. La principale differenza fra i due metodi consiste nella costante fornita al metodo *getNetworkInfo()* riportato alle righe 7 e 10: al posto di ConnectivityManager.TYPE_WIFI deve essere usato il valore

ConnectivityManager.TYPE_MOBILE.

Un caso d'uso pratico di questi metodi verrà fornito a breve.

4.4.2 Controllo della disponibilità di servizio

Poter accedere a Internet per mezzo di una connessione Wi-Fi o cellulare è condizione necessaria per richiedere il servizio remoto di autenticazione, ma non è sufficiente. È importante che l'applicazione ViDiApp sia in grado di stabilire che il servizio remoto sia attivo e funzionante, il che richiede una verifica remota della risorsa, a differenza dei controlli della connessione che sono locali al dispositivo.

Inizialmente è stato pensato di realizzare una funzionalità molto simile al *ping*, capace di rispondere se la macchina remota è raggiungibile oppure no. Ben presto però si è deciso di potenziare questo tipo di verifica poiché stabilire che il server è presente in rete non vuol dire che sia anche accessibile il servizio che offre. La prima soluzione escogitata prevedeva la realizzazione e l'interrogazione di un ulteriore web service, minimale e che non prevedesse trasferimenti di dati fra le parti per avere un *overhead* minimo. Successivamente si è giunti alla conclusione che la funzionalità desiderata poteva essere ottenuta più semplicemente eseguendo una richiesta HTTP GET sul web server. Controllando lo *status code* di ritorno (vedi tabella 4.1) sarebbe stato possibile determinare quanto desiderato sapere. Se non si riceve alcun codice di ritorno, il server è offline, se invece si ottiene un valore numerico, la macchina è online e il numero esprimerà la condizione operativa del web server, e dunque del web service che vi esegue.

In figura 4.29 è riportato il codice di buona parte del metodo *checkService()* della libreria *LibConnect.java* previsto per lo scopo.

Come è possibile osservare, il metodo ritorna un valore intero, negativo se si sono avuti problemi di connessione, di risposta o se è occorsa una qualche eccezione, positivo quando si è ricevuto uno status code valido. Il metodo è stato semplificato per questioni di spazio e dai commenti è possibile apprendere che è gestita la possibilità di operare con connessioni

```
1  public static int checkService() {
2    // Determino se c'e' un Proxy e salvo le caratteristiche...
3    final HttpGet httpGetRequest; // Richiesta HTTP/GET
4    final HttpClient httpclient;  // Client HTTP
5    final HttpResponse hRes;      // risposta HTTP
6
7    httpclient = new DefaultHttpClient(); // Creo il client HTTP
8    // Gestisco l'eventuale Proxy sul client...
9    // Creo la richiesta Http/GET all'url specificato in SERVER_URL
10   httpGetRequest = new HttpGet(SERVER_URL);
11   // Eseguo la richiesta Http/GET
12   try {
13     hRes = httpclient.execute(httpGetRequest);
14     if (hRes == null) return -1;
15     else return hRes.getStatusLine().getStatusCode();
16   }
17   catch (ClientProtocolException cpe)
18         {cpe.printStackTrace(); return -2;}
19   catch (IOException ioe) {ioe.printStackTrace(); return -3;}
20   finally {httpclient.getConnectionManager().shutdown();}
21 }
```

Figura 4.29: Metodo *checkService()* della classe *LibConnect.java*

mediate da un proxy. Nelle prime righe di codice vengono definite delle variabili ove contenere rispettivamente il tipo di richiesta HTTP da effettuare (*httpGetRequest*), l'oggetto client HTTP (*httpclient*) e la risposta ottenuta (*hRes*). Successivamente viene creato il client HTTP sul quale saranno impostati gli eventuali riferimenti al proxy e preparato l'oggetto con la richiesta GET da effettuare al server espresso dalla stringa SERVER_URL (volutamente omesso in figura). Alla riga 10 si esegue la richiesta HTTP/GET e si tenta si salvare il risultato in *hRes*. Come già osservato, se si incorre in un qualsiasi problema, il valore restituito sarà un intero negativo, altrimenti lo status code che viene estratto attraverso la concatenazione dei metodi *.getStatusLine().getStatusCode()*. Il valore atteso capace di indicarci il buon funzionamento del server e del servizio è "200".

4.4.3 Controlli di rete e moduli operativi

Realizzati i controlli delle funzionalità di rete (locali e remoti) riportiamo per mezzo di rappresentazioni grafiche l'inserimento delle stesse all'interno dei moduli che costituiscono ViDiApp. In figura 4.31 è possibile notare il Modulo di Inizializzazione in cui è stato inserito il blocco *Controlli di Rete*. La posizione occupata dai controlli di rete all'interno del modulo ha un preciso significato: si vuole controllare la disponibilità della connessione prima di accedere a Internet, al servizio di Google Maps, utile a tradurre le informazioni di LAC e Cell-ID nelle coordinate di latitudine e longitudine, come descritto nel paragrafo 4.3.

```
1  // Nel metodo onCreate() di Activity_Main
2   // Altro codice...
3   GsmCellLocation gsm_location;
4   gsm_location = (GsmCellLocation) tm.getCellLocation();
5   if (gsm_location != null) {
6     cellID = gsm_location.getCid(); // Ricavo cellID
7     lac = gsm_location.getLac();    // Ricavo lac
8     // Se ho una connessione valida eseguo LocationRequest()
9     if (LibConnect.mobileIsOk(MainCtx) ||
10      LibConnect.wifiIsOk(MainCtx)) {
11      // Risolvo e imposto nell'oggetto 'init' i valori lat. e lon.
12      if (LibConnect.LocationRequest(cellID, lac)) {
13        init.setLat(LibConnect.latitudeSTR);
14        init.setLon(LibConnect.longitudeSTR);
15      } // End if - LocationRequest()
16    } // End if - verifica connessione
17  } // End if - gsm_location
```

Figura 4.30: Metodo onCreate() di Activity_Main con i controlli di connessione

Per comprendere meglio quanto osservato, si rimanda al listato 4.30 ove sono stati accorpati i codici delle figure 4.22 e 4.23 e introdotti i controlli di rete relativi alla connessione. Una volta ricavati i parametri *_cellID* e *_lac*, prima di avviare la traduzione per mezzo del metodo *LibConnect.LocationRequest()*, viene eseguita la verifica di connessione tramite le

funzionalità di rete precedentemente descritte (*wifiIsOk()* e *mobileIsOk()*): se almeno una delle due ha esito positivo, la traduzione delle coordinate può essere messa in esecuzione.

Figura 4.31: Blocco *Controlli di Rete* inserito nel Modulo di Inizializzazione

Un altro modulo in cui vengono introdotti i controlli di rete, sia quelli relativi alla verifica di disponibilità di connessione, sia quelli progettati per

accertare la presenza del servizio remoto di autenticazione, è il Modulo di Trasmissione (figura 4.32).

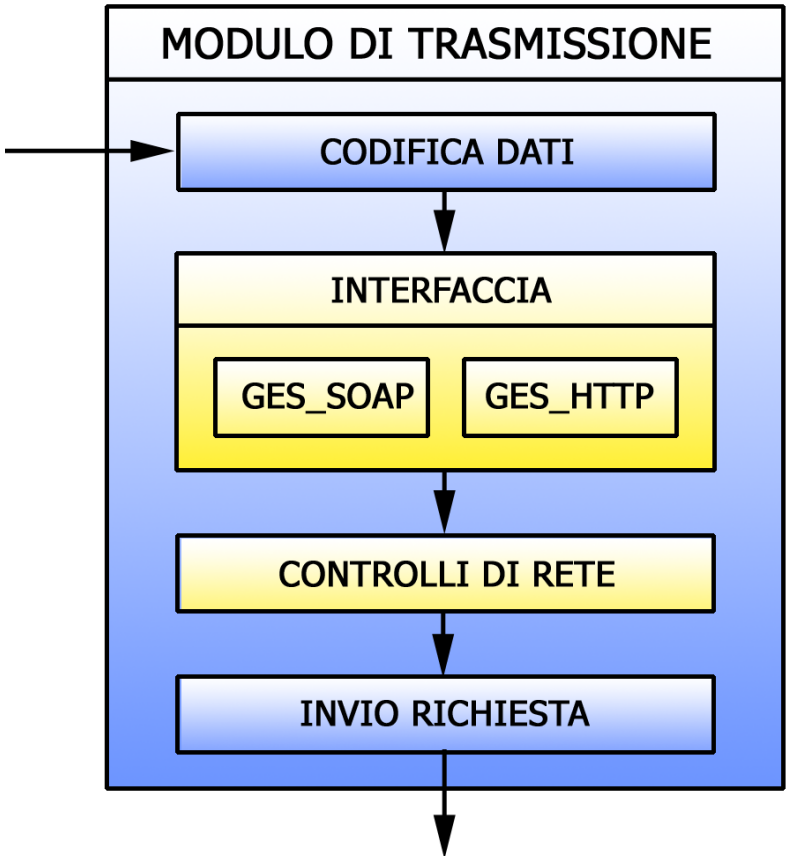


Figura 4.32: Blocco *Controlli di Rete* inserito nel Modulo di Trasmissione

Commentare il codice relativo a questo specifico intervento è abbastanza difficoltoso a causa dei moltissimi rimandi interni, pertanto si presenta un listato reinventato e semplificato (figura 4.33) che utilizza le funzionalità descritte nel blocco Gestore HTTP discusso nel paragrafo 4.1.3.

Definite due variabili stringa rispettivamente utilizzate per salvare la risposta in formato XML del server (*response*) e le informazioni relative alla comunicazione (*info*), viene creata un'istanza del Gestore HTTP. I parametri forniti al costruttore non sono indicati esplicitamente, ma speci-

ficano l'url del server web ove è possibile eseguire una richiesta POST (*SERVER_URL*) e i dati da inviargli (*data*).

```
String response; // Risposta del server di autenticazione
String info;     // Informazioni sulla comunicazione

// Determino SERVER_URL e data...
// SERVER_URL: url del server web ove e' presente il web service
// data      : stringa in formato XML con i dati da inviare
// Creo l'istanza del Gestore HTTP
hHTTP = new HandlerHTTP(SERVER_URL, data);
// Verifico la disponibilita' della connessione
boolean connOk = (LibConnect.mobileIsOk(MainCtx) ||
            LibConnect.wifiIsOk(MainCtx)));
if (connOk){
    int statusCode = checkService(); // Ricavo lo satus code
    if (statusCode == 200) {
         // Ho il servizio, invio i dati
         hHTTP.sendMessage();
         // Prelevo la risposta
         response = hHTTP.getResponse();
    }
   else {Log.e("MT.ERRORE   ", "Servizio down");
         info = hHTTP.getInfo();
         Log.e("MT.hHTTP INFO", info);}
} else {...}
```

Figura 4.33: Codice illustrativo per l'uso dei controlli di rete (locali e remoti) nel Modulo di Trasmissione

Verificato lo stato della connessione e salvato il risultato in una variabile booleana di comodo (*connOk*), solo se si ha modo di accedere a Internet verrà invocato il metodo *checkService()* capace di appurare la presenza del servizio di autenticazione. Se il server web risponde correttamente, verrà eseguito dapprima l'invio dei dati, poi il prelevamento della risposta: ambedue le operazioni sono possibili intervenendo sull'oggetto *hTTTP* rispettivamente con i metodi *sendMessage()* e *getResponse()*. Se al contrario si è verificato un errore per disservizio e non imputabile alla connessione

del dispositivo mobile, nel log verrà salvata un'apposita segnalazione e i dati restituiti dall'esito della comunicazione.

4.4.4 Ricerca del servizio di autenticazione

Utilizzando la funzionalità offerta da *checkService()* descritta nel paragrafo 4.4.2 è possibile verificare se il servizio remoto di autenticazione è disponibile oppure no, ma relativamente a uno specifico server indicato per mezzo di una stringa costante: *SERVER_URL* (figura 4.29). Nel momento in cui si è deciso si parametrizzare il metodo al fine di poter interrogare servitori diversi, è nata una nuova idea poi sviluppata e in parte realizzata per mezzo dei metodi che commenteremo a breve. Invece di interrogare direttamente un server al fine di sapere se è presente in rete e se è disponibile a offrire il servizio di autenticazione, si potrebbe interrogare una macchina adeguatamente istruita e capace di restituire l'url del "miglior server" utilizzabile al momento. Questa speciale macchina riceverà periodicamente dai vari servitori numerose informazioni sul loro stato di funzionamento, sulle loro capacità di esecuzione e sul volume di richieste servite. In base a tali dati, un'opportuna politica di elezione determinerà il servitore più indicato da consigliare ai clienti e renderà disponibile l'url della macchina salvandolo in una pagina web dinamica. Nel codice che esamineremo, relativo a due metodi fra loro collegati, vedremo come l'applicazione ViDiApp sia in grado di ricavare questa informazione.

In figura 4.35 è riportato il codice del metodo *findService()* che sarà invocato da *findService2()* presentato in figura 4.34.

Il metodo *findService()* riceve in ingresso una stringa che ha il significato di un url e ne viene immediatamente verificata la sua correttezza. Se la stringa è conforme, similmente al modo di operare di *checkService()*, il metodo crea le risorse necessarie per eseguire una richiesta HTTP/GET sull'url specificato e se necessario terrà conto della presenza di eventuali proxy (codice non riportato nel listato). Qualora si verifichi un qualsiasi problema, dovuto al sollevarsi di una eccezione oppure per una risposta HTTP fallita (*hRes* resta un'istanza vuota), il metodo ritorna la stringa nul-

la. Al contrario, se dalla risposta è possibile estrarne lo status code e il suo stato assume un valore diverso da "200", quello stesso valore verrà immediatamente restituito in forma di stringa.

```
public static String findService2(Context ctx,
    String urlPageInfo) {
  // Se ho almeno una connettivita' alla rete...
  if (LibConnect.mobileIsOk(ctx) || LibConnect.wifiIsOk(ctx)) {
   // Ricavo il servizio dal ritorno di findService()
   String service = LibConnect.findService(urlPageInfo);
   if (service == null) {return null;}
   else if (service.startsWith("http://"))
     {service += "/"; // Rendo omogeneo
      return service;
     }
   else
     { // Caso di stringa pari a uno status code diverso da '200'
       return null;
     }
   }
   else return null; // Caso in cui non ho connettivita'
}
```

Figura 4.34: Metodo *findService2()*

Infine, nel caso in cui si riscontri un codice di corretta esecuzione del metodo GET, dall'oggetto *hRes* verrà estratta e restituita la risposta vera e propria, ovvero quanto letto dalla pagina web interrogata.

Il metodo *findService2()* (figura 4.34) ha due scopi: invocare *findService()* con la sicurezza di disporre di una connessione di rete, manipolarne la risposta ottenuta.

La chiamata a *findService2()* prevede il passaggio di due parametri, uno relativo al contesto dell'ambiente Android (*ctx*), uno di tipo stringa in cui si specifica l'url della pagina web del "server informativo" da consultare (*urlPageInfo*).

```
1  public static String findService(String url) {
2    // Eseguo una piccola verifica sulla stringa passata come url
3    try {new URL(url);}
4    catch (MalformedURLException mURLe) {
5      mURLe.printStackTrace(); return "Invalid url: " + url;
6    }
7    // Determino se c'e' un Proxy e salvo le caratteristiche...
8    final HttpGet httpGetRequest; // Richiesta HTTP/GET
9    final HttpClient httpclient;  // Client HTTP
10   final HttpResponse hRes;      // risposta HTTP
11   httpclient = new DefaultHttpClient(); // Creo il client HTTP
12   // Gestisco l'eventuale Proxy sul client...
13   // Creo la richiesta Http/GET: url e' stato verificato
14   httpGetRequest = new HttpGet(url);
15   // Eseguo la richiesta Http/GET
16   try {
17        hRes = httpclient.execute(httpGetRequest);
18        if (hRes == null) return null;
19        else {
20        int code = hRes.getStatusLine().getStatusCode();
21        switch (code) {
22          case 200: {if (hRes.getEntity() == null) return null;
23         else
24            try {return EntityUtils.toString(hRes.getEntity());}
25            catch (ParseException pe) {pe.printStackTrace();}
26            catch (IOException ioe) {ioe.printStackTrace();}
27        } // End case 200
28          default: {return Integer.toString(code);}
29        } // End switch
30        }
31   }
32   catch (ClientProtocolException cpe) {cpe.printStackTrace();
33                                        return null;}
34   catch (IOException e) {e.printStackTrace(); return null;}
35   finally {httpclient.getConnectionManager().shutdown();}
36 }
```

Figura 4.35: Metodo *findService()*

Verificata la possibilità di connessione, controllo effettuato alla riga 4, viene invocato il metodo *findService()* precedentemente descritto a cui è passato il valore *urlPageInfo*. Una volta che il suo risultato è stato salvato nella stringa *service*, solo se il suo valore è una stringa rappresentativa (diversa dal valore nullo o da uno status code), vengono effettuate due ultime operazioni. La prima esegue un controllo sulla stringa ricevuta e vuole accertarsi che sia un url che riferisca un server web. La realizzazione di questa funzionalità non è molto elegante, ma è efficace e di fatto esprime più un'esigenza di debug che una verifica operativa. L'altra operazione svolta aggiunge in coda un "/" poiché per questioni di sicurezza l'url ricevuto ne è sprovvisto.

In caso di eccezioni, mancata connessione, o per ritorni indesiderati di *findService()*, il metodo *findService2()* restituisce un valore nullo, altrimenti si avrà a disposizione una stringa verificata relativa a un url che individua un server web presente in rete e capace di offrire un servizio di autenticazione funzionante.

4.5 Ulteriori interventi

Diverse altre modifiche sono state eseguite su vari blocchi funzionali, anche in questi casi con finalità molto differenti. A conclusione di questo capitolo si vogliono offrire al lettore due ultimi esempi di intervento, ma senza ricorrere a dettagli implementativi.

Nel primo caso si è intervenuti nel Modulo di Inizializzazione ove i controlli di rete (locali e remoti) sono stati resi concorrenti per mezzo dell'uso di un thread dedicato.

L'altro esempio riportato introduce una modifica apportata al Modulo di Scansione che punta a ottenere una maschera di acquisizione capace di adattarsi alle dimensioni del display del dispositivo mobile in uso.

4.5.1 Gestione dei controlli di rete via thread

Una volta realizzate le diverse funzionalità di rete e averle integrate nei Moduli di Inizializzazione e Trasmissione come indicato nel paragrafo 4.4.4, si è notato che in alcuni casi potevano verificarsi latenze dell'ordine di qualche secondo, in particolare durante il controllo del servizio remoto in presenza di proxy. Il problema è raramente riscontrabile nel Modulo di Trasmissione e comunque non causa disagi all'utente poiché mascherato dalla barra di attesa. Nel Modulo di Inizializzazione è più probabile incorrere in questo ritardo, soprattutto se la ViDiApp è la prima attività a essere avviata fra quelle con richiesta di accesso a Internet. Il problema non è tanto l'attesa, aspetto comunque da indagare adeguatamene e risolvere, quanto piuttosto la visualizzazione di un display nero che disorienta l'utente.

Invece di introdurre una barra di attesa animata, avendo a disposizione il blocco di *Presentazione Help*, si è deciso di rendere a esso parallelo quello dei *Controlli di Rete*, come indicato in figura 4.36. Come è possibile osservare, dopo la fase iniziale, l'identificazione del dispositivo e la determinazione dei parametri LAC e Cell-ID, il Modulo di Inizializzazione avvia in modo concorrente la presentazione animata e i controlli di rete (locali e remoti). Durante i pochi secondi di riproduzione della mini-guida utente, i controlli di rete possono essere eseguiti efficacemente, inoltre l'utente dovrà agire manualmente per accedere alla fase di scansione, come descritto nel paragrafo 3.1.2, fornendo ulteriore tempo al sistema di verifica.

Fra gli ulteriori accorgimenti utilizzati è previsto che l'animazione sia eseguita in modo continuo e inamovibile dal display finché non siano stati effettuati i controlli di rete. Nel caso in cui i controlli falliscano e l'utente tenti di passare alla fase di scansione, verrà immediatamente avvisato del problema tramite un messaggio a video. Se si è verificata una mancanza di connettività imputabile al dispositivo, l'utilizzatore sarà invitato ad attivare la connessione Internet.

Tornando al flusso di esecuzione del Modulo di Inizializzazione, ter-

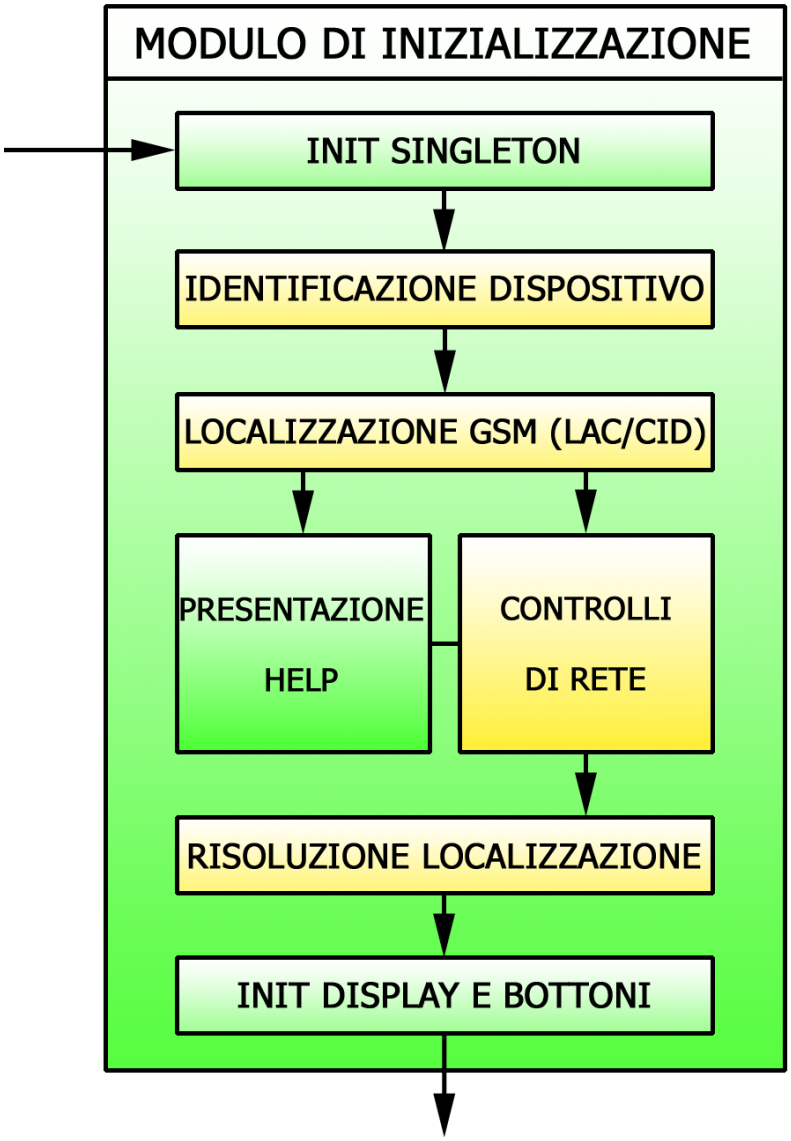


Figura 4.36: Modifica al Modulo di Inizializzazione: esecuzione concorrente dei blocchi *Presentazione Help* e *Controlli di Rete*

minati i controlli di rete con esito positivo, il controllo viene passato al blocco di *Risoluzione Localizzazione* descritto nei paragrafi 4.3 e 4.4.4.

Il thread è realizzato estendendo la classe *Thread.java* e prevede campi e parametri necessari a realizzare i controlli di rete, come ad esempio quello relativo al contesto utile a *wifiIsOk()* e *mobileIsOk()*. Prevede inol-

tre la possibilità di mantenere in una variabile booleana interna il risultato dei controlli eseguiti. Esistono vari modi per poter trasmettere questa informazione, ad esempio impostandola in un campo dedicato nell'oggetto *init*, istanza del Singleton oppure per mezzo della notifica di un messaggio costruito e gestito tramite le classi *Message*, *Handler* e *Bundle* offerte da Android.

4.5.2 Display, preview e maschere di acquisizione

In figura 4.37 è riportato il Modulo di Scansione presentato con un dettaglio maggiore rispetto a quanto visto in precedenza (figura 3.7). L'area di intervento è quella colorata in giallo e relativa al blocco di *Preparazione Maschera*.

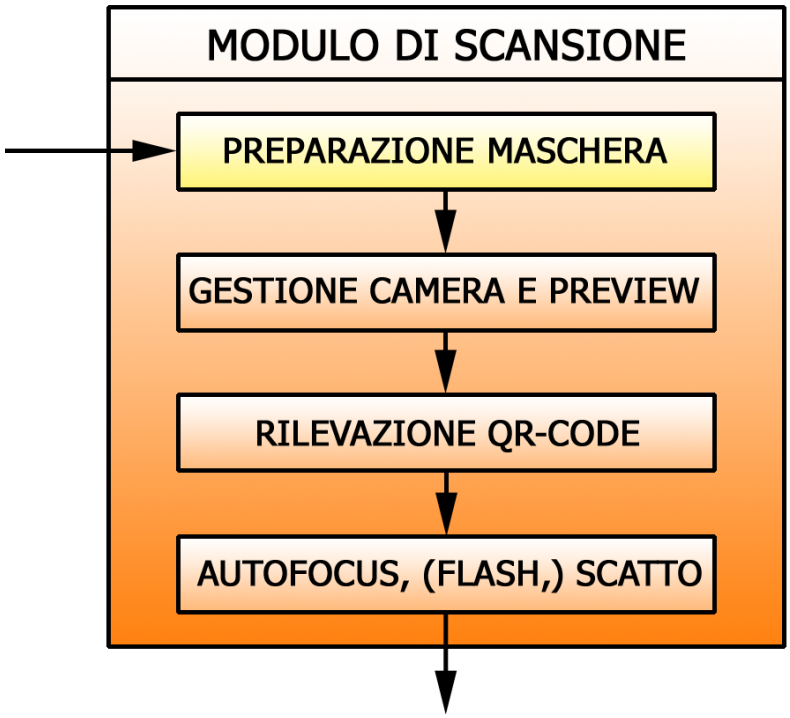


Figura 4.37: Modifica al blocco *Preparazione Maschera* del Modulo di Scansione

Come affermato nel paragrafo 3.1.2, si desidera effettuare una foto macro, ovvero da una distanza ravvicinata di al massimo una decina di centi-

metri. Il modo per suggerire all'utilizzatore tale distanza utile a effettuare lo scatto macro è quello di disegnare sul display una maschera di acquisizione ben dimensionata. Inquadrando il timbro fra i contorni rossi come mostrato in figura 3.3(a) si avrà la sicurezza che il dispositivo mobile è stato posizionato alla giusta distanza.

Disegnare in modo appropriato i contorni rossi della maschera è possibile eseguendo dei calcoli su alcuni parametri che caratterizzano il display, ad esempio le sue dimensioni in pixel, lato lungo e lato corto.

Prima dell'intervento nel blocco di Preparazione Maschera, le grandezze di riferimento del video erano prefissate poiché riferite a un unico dispositivo fisico usato per i test. Non appena il parco macchine si è ampliato (tabella 3.1), l'utilizzo dell'applicazione ha palesato una rappresentazione incoerente dei bordi della maschera di acquisizione e ciò suggeriva all'utente una distanza errata da cui effettuare gli scatti. Il problema riscontrato dipendeva dalle eterogenee caratteristiche video dei display, dimensionalmente diversi da modello a modello.

Le modifiche apportato al blocco hanno fatto sì di generalizzare i parametri relativi allo schermo, principalmente alle sue dimensioni espresse in altezza/lunghezza. Non più considerati valori numerici fissi, vengono determinati a tempo di esecuzione interrogando il sistema Android.

4.6 Conclusioni

In questo sezione tecnica, in base alle aree operative individuate nel capitolo precedente, sono stati riportati gli interventi principali eseguiti sull'applicazione ViDiApp.

Innanzi tutto è stato analizzato il Modulo di Trasmissione: dopo aver esaminato la soluzione SOAP adottata per la comunicazione dei dati fra il dispositivo mobile e i server di autenticazione, si sono elencate le principali limitazioni inerenti la libreria open source kSOAP2 utilizzata. Dopo aver presentato l'alternativa HTTP, protocollo supportato da Android, avvalendosi del commento al codice si è illustrato al lettore come il Gestore HTTP realizzato sia stato messo a disposizione in alternativa al già pre-

sente Gestore SOAP. Ciò è stato possibile per mezzo dell'introduzione di un'opportuna interfaccia capace di avvicinare modi molto diversi di operare ma accomunati dalle stesse finalità: instaurare una comunicazione di alto livello (applicativo).

Successivamente è stato sviluppato l'argomento relativo all'identificazione del dispositivo mobile. Affrontata una analisi sull'insieme dei parametri utili allo scopo, si è cercato di ottenere una soluzione capace di non ledere la privacy dell'utilizzatore pur mantenendo un buon grado di generalità. Il codice introdotto a supporto della descrizione risolutiva e che prevede l'uso con priorità di certi parametri identificativi (IMEI / IMSI / Mac Address) ha introdotto in modo particolareggiato alcuni moduli e classi base utilizzati anche nei successivi paragrafi.

Un'altra importante funzionalità sviluppata e integrata in ViDiApp si occupa di localizzare il dispositivo mobile. Dopo un breve richiamo ai concetti espressi nel paragrafo 1.3.2, sono stati illustrati i motivi che hanno portato ad adottare un approccio alla localizzazione basato su rete cellulare, nel particolare il metodo Cell-based. Prima di commentare il codice realizzato, si è fatto cenno alle API Android che si occupano della localizzazione evidenziando come quelle relative alla tecnica prescelta non restituiscano direttamente le coordinate in forma di latitudine e longitudine, ma due codici numerici, LAC e Cell-ID. Questa coppia di valori che esprimono rispettivamente la Local Area Code (codice di area) e l'identificatore di cella, permettono sì di determinare la posizione del dispositivo, ma devono essere tradotti nel formato atteso. Elencati i possibili metodi con cui raggiungere lo scopo, la restante parte del paragrafo è stata dedicata all'esplorazione della soluzione implementata che prevede l'accesso alla rete Internet e lo sfruttamento del servizio di mappe Google Maps.

Successivamente è stato descritto e commentato un gruppo di metodi raccolti nella libreria *LibConnect.java* capaci di realizzare le più importanti funzionalità di rete utilizzate da ViDiApp. In essa sono implementati metodi di verifica locale al dispositivo che permettono di controllarne le capacità di connessione a Internet (via Wi-Fi o rete cellulare), come pure un metodo di accertamento remoto in grado di stabilire se il web server

con cui interagire è raggiungibile e il suo servizio di autenticazione funzionante. A seguire, il paragrafo descrive attraverso commenti e figure l'inserimento dei controlli di rete (riassunti nell'omonimo blocco) nelle sezioni relative al Modulo di Inizializzazione e al Modulo di Trasmissione. Infine sono stati presentati due metodi annidati capaci di aggregare le funzionalità di controllo di rete locali e remote, nonché di eseguire una ricerca presso un server informativo dedicato. Il fine di tale indagine è quello di ottenere come risultato l'url del web server al momento più indicato a cui richiedere il servizio remoto di autenticazione.

Il capitolo si conclude presentando al lettore altri due interventi eseguiti sull'applicazione ViDiApp, ma senza riportare i dettagli tecnici implementativi. Il primo di essi consiste nel rendere concorrenti i controlli di rete effettuati nel Modulo di Inizializzazione per mezzo dell'uso di un thread. La seconda modifica coinvolge il Modulo di Scansione e più precisamente il blocco di Preparazione Maschera adibito a realizzarne i contorni. Dopo l'intervento, le dimensioni della maschera disegnata sul display non sono più calcolate ai parametri dello schermo considerati come valori fissi, ma determinati a tempo di esecuzione, specifici del particolare dispositivo in uso.

Capitolo 5

Valutazioni finali e sviluppi futuri

In quest'ultimo capitolo verrà presentato al lettore un esame del Modulo di Trasmissione mettendo a confronto l'uso del Gestore SOAP e del Gestore HTTP. Dopo aver riportato i tipi di test realizzati, le modalità di misurazione adottate e i risultati ottenuti, si esporranno alcune osservazioni sui comportamenti rilevati.

Infine si esporrà una breve panoramica su ViDiApp al fine di indicarne le migliorie da introdurre a breve termine, la realizzazione di nuovi test utili a completare quanto analizzato nel prossimo paragrafo e, in generale, gli sviluppi futuri più probabili.

5.1 Gestore SOAP vs Gestore HTTP

I test effettuati sui gestori SOAP e HTTP riguardano soprattutto l'efficienza temporale valutata durante lo scambio dei dati fra l'applicazione ViDiApp in esecuzione sul dispositivo mobile e il servizio remoto attivo sui server ViDiTrust.

La soluzione alternativa HTTP è stata sviluppata per ottenere l'indipendenza dalla libreria kSOAP2 e dalle sue limitazioni, ma anche per verificare se rinunciando all'envelope SOAP e ad altri orpelli introdotti dal protocollo, fosse possibile aumentare la velocità dei trasferimenti.

L'insieme delle casistiche da sottoporre ad analisi è molto ampio, poiché molte sono le condizioni e i parametri da tenere in considerazione.

Verificata l'impossibilità di effettuare tutte le prove del caso, l'attenzione è stata rivolta a quelle più significative, cercando di individuare i parametri e le grandezze utili a caratterizzarle.

Viceversa, durante l'esecuzione dei test, si è potuto verificare sperimentalmente che alcuni degli elementi reputati essere discriminanti si sono dimostrati ininfluenti. Per offrire al lettore un esempio concreto di quanto appena affermato, basti considerare che sono stati eseguiti set di test distinti in base alle diverse versioni di Android per cui l'applicazione è stata compilata: API 8, 9, 10. Considerando l'insieme delle prove eseguite per il blocco Gestore HTTP, i risultati ottenuti da compilazione differenti non mostravano sostanziali differenze e questa "insensibilità alle API" è stata riscontrata anche all'interno dei campioni di studio relativi al Gestore SOAP. Per questo motivo le misurazioni effettuate sono state accorpate, distinguendole unicamente in base al gruppo SOAP o HTTP di appartenenza.

Prima di procedere oltre, è già possibile affermare in questa fase introduttiva che le prestazioni dei due blocchi si sono dimostrate indipendenti dalle diverse versioni del sistema operativo mobile usato.

5.1.1 Piattaforme di esecuzione

La maggior parte dei test eseguiti durante il periodo di analisi e sviluppo della ViDiApp al fine di risolvere bug, avere indicazioni di massima di funzionamento e per svariati altri scopi, sono stati tutti realizzati per mezzo di dispositivi virtuali e fisici su cui era in esecuzione una certa versione del sistema operativo mobile Android.

Nel caso del Modulo di Trasmissione e dei blocchi relativi alla gestione SOAP e HTTP, i test sono stati effettuati anche su un personal computer portatile poiché il codice relativo alla trasmissione e ricezione può essere agilmente scorporato dall'applicazione mobile per realizzarne una Java. L'unica osservazione da fare è che gli esperimenti eseguiti sul computer hanno sfruttato un bytecode compilato per la Java Virtual Machine di Oracle, viceversa, l'emulatore e gli smartphone hanno utilizzato i com-

pilati *dex* per la Dalvik VM (paragrafo 1.1.2). Come ulteriore precisazione, si segnala che le rappresentazioni virtuali dei device utilizzati sono state create e messe in esecuzione sullo stesso computer portatile utilizzato per i test in ambito Java.

Per quanto esposto nel precedente paragrafo relativamente alla "insensibilità alle API", e anche per altri motivi che addurremo in seguito, non si ritiene necessario riportare dettagliatamente le caratteristiche delle macchine su cui sono stati eseguiti i test. Le proprietà base degli smartphone utilizzati sono indicate in tabella 3.1, in particolare si osservino le caratteristiche del modello Nexus S, il più usato durante l'esecuzione delle prove. Gli emulatori sono stati realizzati in modo da ricalcare il più possibile le caratteristiche del device fisico appena indicato, ma in più versioni, ciascuna con una diversa installazione Android.

5.1.2 Linee guida, contesto e parametri dei test

Come illustrato, i test sono stati effettuati su diverse piattaforme che possono essere raggruppate nelle seguenti categorie: elaboratore, dispositivo virtuale, smartphone. Pertanto, per eseguire delle misurazioni, è stato necessario sviluppare un software dedicato e capace di operare con ciascuna di esse; in realtà sono state realizzate solamente due applicazioni, una dedicata agli elaboratori e che si poggia sulla Java VM e un'altra capace di operare sui dispositivi virtuali e reali per mezzo della Dalvik VM.

Le due versioni eseguono lo stesso nucleo di codice per quanto riguarda la realizzazione delle misurazioni, ma differiscono in alcuni elementi "esterni". Sviluppato inizialmente per operare in un progetto Java, al momento di portare il sistema di verifica su piattaforme Android è stato introdotto il minimo indispensabile per renderlo operativo anche sui sistemi mobili: ad esempio, si è dovuta realizzare l'activity principale, il suo metodo onCreate() ove viene invocato il codice relativo alle misurazioni e poco altro.

Per quanto riguarda il contesto di esecuzione delle prove, iniziamo col dire che per poter avviare delle prove temporali relative ai gestori SOAP

e HTTP occorre accedere a Internet e ciò è possibile per mezzo di una connessione Wi-Fi o tramite la rete cellulare. I test svolti utilizzeranno esclusivamente un tipo di connessione Wi-Fi, l'unica comune a tutte le piattaforme considerate.

Un altro elemento da specificare è la dimensione del pacchetto di informazioni da scambiare fra il gestore HTTP/SOAP presente sul client e il servizio remoto invocato sul server. Una volta fissata, tale grandezza sarà mantenuta costante per tutti i test. Allo scopo è stata utilizzata una stringa XML d'uso reale e cioè scambiata durante un normale processo di verifica di autenticità. Le sue dimensioni sono di circa 186 kB, per la precisione 1904091 byte.

Riguardo all'esecuzione dei test temporali, si vogliono specificare immediatamente due ulteriori aspetti: uno riguarda la convenzione usata per le misurazioni, l'altro esprime la natura multipla di ciascuna prova.

Per convenzione andremo a considerare come tempo di riferimento per lo scambio dati fra le parti la somma del tempo di andata della richiesta (dalla piattaforma cliente al server), del tempo di esecuzione sul server e del tempo di ritorno della risposta (dal server al dispositivo/elaboratore). D'ora in avanti indicheremo questo tempo col nome di *TAER*: Tempo di Andata, Esecuzione (sul server) e Ritorno. Volendo considerare il solo tempo di volo (*TAR*), ovvero senza conteggiare il tempo di esecuzione, sarà sufficiente sottrarre al TAER la quantità di 1,7 secondi, valor medio misurato direttamente sul server e praticamente costante per normali condizioni operative. A tal proposito, si osserva che il server utilizzato per elaborare le richieste e fornire le risposte è sempre stato lo stesso.

Infine, ciascuna prova effettuata sarà caratterizzata da diversi aspetti che illustreremo in seguito, ma si vuole sottolinearne fin da subito la sua natura multipla. Ciò significa che il TAER relativo a una certa prova non è determinato in base a un'unica operazione di andata-esecuzione-ritorno, ma è un valor medio calcolato su un gruppo di operazioni che nelle prove effettuate è stato fissato a 20.

Ulteriori aspetti da tenere in considerazione per comprendere meglio le modalità di esecuzione dei test saranno presentati e commentati nel

prossimo paragrafo, dedicato al software sviluppato per realizzarli.

5.1.3 Software per eseguire i test

Prima di presentare i risultati dei test temporali effettuati e commentarne i valori misurati, occorre descrivere il funzionamento del sviluppato utile per ottenerli.

```
1  public class MiniVidiJava {
2    private final static String SERVER = ""; // Server omesso
3    private final static int N_GIRI = 20; // Operazioni previste
4
5    // Stima temporale (in ms) del tempo di esecuzione del server
6    private final static int EXEC_TIME_MS = 1700;
7    // Limite temporale (in ms) SOTTO il quale la prova e' scartata
8    private final static long LIMITE_INF = 2150; // 2,15 secondi
9    // Limite temporale (in ms) SOPRA il quale la prova e' scartata
10   private final static long LIMITE_SUP = 90000; // 90 secondi
11   // Array utili a collezionare i TAER per POST e SOAP
12   private static long tempiAER_POST[]= new long[N_GIRI];
13   private static long tempiAER_SOAP[]= new long[N_GIRI];
14
15   public static void main(String[] args) {...}
16
17   // Il metodo opera sugli array-temporali ottenuti dalle prove.
18   // I valori in essi contenuti sono al minimo zero.
19   // Il valore restituito e' il numero piu' piccolo e diverso da zero
20   private static long trovaMin(long array[]) {
21     long min = -1;
22     Arrays.sort(array);
23     for (int i = 0; i < array.length; i++) {
24       if (array[i] != 0) {min = array[i]; break;}
25     }
26     return min;
27   }
28 }
```

Figura 5.1: Struttura base dell'applicazione Java utile a eseguire i test temporali

Seguendo una linea cronologica di sviluppo, verrà dapprima presentato il programma realizzato per ambiente Java: la sua struttura base è presentata in figura 5.1. Si possono immediatamente notare la definizione di alcune utili grandezze: il valore costante (omesso) di tipo stringa conterrà l'url del server ViDiTrust da raggiungere e la costante intera fissata al valore 20 individuerà il grado di molteplicità desiderato per la prova, come illustrato in precedenza. I tre valori numerici seguenti (*EXEC_TIME_MS*, *LIMITE_INF*, *LIMITE_SUP*) sono costanti che permettono rispettivamente di indicare il numero di millisecondi stimato per l'esecuzione del server e i limiti temporali inferiore e superiore per cui una singola fase della prova è da considerarsi non valida.

Successivamente sono definiti due array capaci di contenere un numero di valori temporali pari al numero di operazioni prefissate: un array è usato per le prove (HTTP/) POST, l'altro per SOAP.

Infine sono indicati il metodo *main()* che verrà descritto separatamente e il metodo *trovaMin()* capace di operare sugli array temporali e restituire il minimo valore diverso da zero in essi contenuti.

Esaminiamo ora la prima parte del metodo *main()* del software MiniVidiJava presentato in figura 5.2. Nel metodo principale verranno definite e inizializzate due stringhe per ospitare il messaggio da inviare e quello da ricevere (*mess* e *ress*) e due oggetti per i gestori HTTP e SOAP.

Prelevando da un file XML i dati da inviare al server, il suo contenuto sarà assegnato a *mess* in forma di stringa e poi verranno visualizzate a video alcune informazioni come ad esempio la sua dimensione in byte (righe 15-16).

Infine saranno definiti e inizializzati diversi parametri numerici utili alla realizzazione delle prove temporali: gli interi *succPOST* e *succSOAP* hanno lo scopo di tener traccia del numero di prove singole eseguite con successo rispetto alle 20 previste e dunque ricavare una percentuale di affidabilità da associare alla prova multipla.

Ora esaminiamo nel listato in figura 5.3 il corpo principale adibito alla realizzazione dei test temporali.

Tramite una struttura iterativa, dopo aver raddoppiato il numero di

```
1  public static void main(String[] args) {
2    String mess="";  // Stringa del messaggio da inviare
3    String res = ""; // Stringa da usare per la risposta
4    // Oggetti dei gestori SOAP e HTTP/POST
5    HandlerSOAP oS; HandlerHTTP oP;
6
7    // Preparo il messaggio leggendo dal file
8    File xmlFile = new File("./xml_invio"); //Dimensioni 190/200 kB
9    DataInputStream dis = null; // Creo 'dis' e lo collego a xmlFile
10   ByteArrayOutputStream baos = null; // Creo 'baos'
11   // Leggo da 'dis' e scrivo in 'baos', infine ne ottengo 'mess'
12   mess += baos.toString();
13
14   // Preambolo ai TEST Temporali
15   System.out.println("* Lunghezza str invio: "
16       +mess.length()+" byte"); // Altre informazioni...
17   // Grandezze utili alle misurazioni temporali
18   long inizio=0; // Tempo di inizio misura
19   long fine=0;   // Tempo di fine misura
20   long tempoAER=0; // Tempo di misura TAER (fine - inizio)
21   long totaleTAER_POST=0; // Somma dei tempi TAER via POST
22   long totaleTAER_SOAP=0; // Somma dei tempi TAER via SOAP
23   // Grandezze utili a misurare la qualita' del test
24   int succPOST=0; int succSOAP=0;
25 } // End main()
```

Figura 5.2: Parte del metodo *main()* dell'applicazione Java MiniVidiJava

giri preventivato, per mezzo di un contatore si inizierà a realizzare i singoli test temporali. Quando il contatore assume un valore dispari, verrà azionato un test sul gestore HTTP, per un valore pari sarà avviato un test sul gestore SOAP.

Questo modo di operare prevede l'alternanza dei singoli test e pertanto verrà riferito come modalità di "Esecuzione Alternata" (ALT). In origine, le operazioni erano iterate, ma svolte in modo sequenziale (SEQ) ovvero prima venivano eseguiti 20 invii HTTP e successivamente tutti quelli SOAP. Il codice per effettuare test sequenziali sarà omesso, ma del suo

```
// Ciclo HTTP/POST e SOAP intercalati - 'Esecuzione Alternata'
  int totGiri=N_GIRI*2;
  for (int i = 1; i <= totGiri; i++) {
    // Se dispari lavoro con HTTP/POST
    if (i%2!=0) {
      oP = null; oP = new HandlerHTTP(SERVER, mess);
      inizio = System.currentTimeMillis();
      oP.sendMessage();
      fine = System.currentTimeMillis();
        // res=oP.getResponse(); // Prelevo e stampo risposta...
      tempoAER = fine - inizio;

      // Se ho 'tempoAER' ok, incremento succPOST,
      // altrimenti impongo 'tempoAER' a zero
      if ((tempoAER > LIMITE_INF) && (tempoAER <= LIMITE_SUP))
      succPOST++;
      else
      tempoAER = 0;
      tempiAER_POST[((i+1)/2) - 1] = tempoAER;
      System.out.println("  POST - Giro n." + ((i+1)/2)
          + " - Tempo AER: " + tempoAER + " ms"
          + " - Tempo AR (stima): "
          + (tempoAER-EXEC_TIME_MS) + " ms");
       totaleTAER_POST+=tempoAER;
      } // End POST
      // Se 'i' e' dispari, lavoro con SOAP
      else {// Codice analogo al precedente...
      }
  } // End for
```

Figura 5.3: Parte di codice del metodo *main()* utile ai test temporali
Esecuzione alternata

significato si discuterà in seguito.

Tornando a esaminare il listato, è possibile notare che nel caso di invio HTTP, l'oggetto *oP* verrà di volta in volta reso nullo e poi creato ex novo; subito dopo verrà eseguita l'operazione di invio (*sendMessage()*) che se andrà a buon fine includerà anche il processo di esecuzione sul server e la

ricezione della risposta, salvata internamente al gestore HTTP. Per questi motivi è sufficiente misurare il tempo agli estremi di questa "unica" operazione per ottenere il tempo TAER della singola fase. Se il tempo misurato è compatibile con i limiti temporali precedentemente definiti, il numero di successi ottenuti per le operazioni effettuate dal gestore HTTP verrà incrementato, altrimenti il tempo misurato sarà reso pari a zero (righe 15-18). A prescindere dal valore assunto, il tempo verrà salvato nell'array dedicato allo scopo mentre a video saranno stampate alcune informazioni sulla prova appena eseguita.

Analogamente si procederà per effettuare le singole prove relative al gestore SOAP, il cui codice è stato omesso in figura.

Per concludere la valutazione delle misurazioni temporali eseguite, analizzando gli array ove sono state salvate, il sistema è in grado di determinare il tempo migliore (minimo), il tempo peggiore (massimo) e il tempo medio delle operazioni di "andata, elaborazione e ritorno". Il codice relativo a questa parte è indicato in figura 5.4.

```
1  // Elaboro i dati temporali relativi al gestore HTTP/POST
2  Arrays.sort(tempiAER_POST);
3  System.out.println("\n* Ciclo POST  - Affidabilita' del test: "
4      +(float)((100*succPOST)/N_GIRI)+"%");
5  System.out.println("  Tempo AER (Migliore/Peggiore/Medio) in ms: "
6      + trovaMin(tempiAER_POST)+ " / " + tempiAER_POST[N_GIRI - 1]
7      + " / "+ (totaleTAER_POST / succPOST));
8  System.out.println("  Tempo  AR (Migliore/Peggiore/Medio) in ms: "
9      + (trovaMin(tempiAER_POST) - EXEC_TIME_MS) + " / "
10     + (tempiAER_POST[N_GIRI - 1] - EXEC_TIME_MS) + " / "
11     + ((totaleTAER_POST - (succPOST * EXEC_TIME_MS)) / succPOST));
12 // Elaboro dati SOAP; analogo la codice precedente...
```

Figura 5.4: Parte di codice del metodo *main()* utile all'elaborazione finale e stampa a video dei test temporali

È utile notare che oltre le informazioni appena illustrate, a video verrà anche stampata l'affidabilità percentuale del test (intesa come prova

multipla), ovvero quante prove singole sono andate a buon fine su quelle previste (20).

Infine, il valor medio calcolato e proposto è valutato in base alle sole prove concluse con successo e di ciò se ne terrà conto durante le valutazioni finali per mezzo del peso fornito dalla percentuale di affidabilità.

Il software sviluppato per essere eseguito sui dispositivi virtuali e reali prevede al suo interno lo stesso nucleo di codice che abbiamo fin qui discusso e quindi non verrà presentato. Come precedentemente affermato, le uniche differenze previste riguardano la definizione dell'activity, la gestione del metodo di callback onCreate() e il prelevamento dal file delle informazioni XML da inviare al server, operazione eseguita in maniera leggermente diversa su Android.

Esaminata la realizzazione del software usato per effettuare le misurazioni temporali, concludiamo con due note.

L'originario sistema di test eseguiti in modo sequenziale (SEQ) è stato abbandonato poiché nel caso di singole operazioni di "andata-esecuzione-ritorno" molto lunghe (30/40 secondi ciascuna), l'avvio di quelle relative all'altro gestore venivano enormemente ritardate (anche di 10 o 15 minuti). Il problema di questo approccio alla misurazione è che, seppur lentamente, le caratteristiche prestazionali di certe connessioni Wi-Fi possono variare sensibilmente, falsando i risultati delle misurazioni. Ciò accade quando buona parte delle verifiche relative a un gestore vengono eseguite con certe prestazioni di rete e quelle relative all'altro gestore sono realizzate sulla stessa connessione che nel mentre ha assunto caratteristiche differenti.

Operare con verifiche alternate (ALT) permette di mediare la variazione di prestazioni relativa alla rete wireless e dunque consente di fornire confronti più equi fra i due gestori.

In merito a quanto detto, si potrebbe pensare di migliorare ulteriormente i confronti rendendo le misurazioni concorrenti, ad esempio sfruttando dei thread. Questo approccio è stato discusso in azienda, ma alla fine è stato accantonato non volendo introdurre *overhead* di difficile valutazione dovuti agli inevitabili *context switch*.

5.1.4 Risultati dei test temporali

Le misurazioni temporali effettuate sono state riportate in alcune tabelle presentate nelle prossime pagine.

Le tabelle prevedono un titolo col quale riferiscono l'applicazione e/o la piattaforma ove sono state eseguite, nonché una categoria con la quale sono state classificate le reti wireless con cui si era connessi al momento delle prove.

Le categorie previste sono due:

- STD (rete standard).
- RNA (rete non affidabile).

La prima voce indica l'insieme delle reti che presentano mediamente buone prestazioni in termini di potenza di segnale ricevuto, latenze piccole per stabilire i canali di connessione, banda di trasmissione elevata, etc.

La seconda voce include le reti che presentano problemi anche gravi in una o più delle caratteristiche sopra indicate.

A seguire, nelle tabelle sono presentati due macro-blocchi, a sinistra si trova quello relativo al gestore HTTP, la parte destra è dedicata a SOAP. Nelle intestazioni dei blocchi sono riportati alcuni parametri discussi precedentemente a formare una legenda utile come promemoria. È possibile leggere che ciascuna prova multipla è costituita da 20 prove singole, che la dimensione dei messaggi inviati è di circa 186kB e che le misurazioni effettuate riguardano i TAER, in tabella indicati come "Tempi AER".

Su ciascuna riga sono presenti, da sinistra a destra, alcune caratteristiche relative alla misurazione, come la data e l'ora, il nome della rete e se possibile la potenza del segnale ricevuto espressa come una percentuale. Di fianco è indicata l'affidabilità percentuale determinata come descritto nel paragrafo 5.1.3 e infine sono presentati i tempi misurati ed espressi in millisecondi: il tempo migliore, il peggiore e il valor medio, calcolato sul numero di successi ottenuti.

Come ultime due osservazioni utili alla comprensione della tabella occorre indicare che fra i due blocchi è possibile leggere una sigla, SEQ aut

ALT, che indica rispettivamente se il software di misurazione ha eseguito delle prove singole in modo sequenziale o alternato.

Infine, nell'ultima riga compaiono le medie pesate dei tre valori temporali misurati, calcolati su tutte le misurazioni riportate e relative alla propria categoria: le medie vengono pesate col parametro di affidabilità. Più il suo valore percentuale è elevato, più la prova è significativa e maggiore sarà il suo peso nel calcolo dei valor medi.

Le tabelle presentate sono le seguenti:

- le tabelle 5.3 e 5.4 sono relative all'applicazione Java MiniVidiJava eseguita sul computer portatile, rispettivamente per una rete STD e per una RNA;
- la tabella 5.5 riguarda l'applicazione TransmissionVidi per piattaforma Android ed è eseguita su dispositivo virtuale; la rete di riferimento è solamente quella STD;
- le tabelle 5.6 e 5.7 sono state realizzate con i risultati ottenuti dall'app TransmissionVidi eseguita sul dispositivo fisico Nexus S, rispettivamente usato con una rete STD e con una RNA.

Prima di esaminare i risultati, si fa notare al lettore che in tabella 5.6 alcune delle prove sono state effettuate in sede ViDiTrust con il server di riferimento a poche decine di metri (rete Silver). In questo caso i tempi sono risultati molto bassi, tanto da dover rimuovere momentaneamente i limiti temporali inferiori descritti in precedenza.

5.1.4.1 Valutazioni finali

Per comprendere meglio le valutazioni effettuate sui risultati ottenuti e presentate qui di seguito, nelle tabelle 5.1 e 5.2 sono stati riassunti i dettagli più importanti delle misurazioni effettuate per i gestori HTTP e SOAP.

Dal confronto diretto dei risultati ottenuti per il gestore HTTP e per quello SOAP si nota che i tempi riscontrati sono abbastanza simili. In prima battuta non sembra esserci un miglioramento o peggioramento sensibile delle prestazioni usando un metodo di comunicazione piuttosto che un altro.

Piattaforma	Rete	TAER Migl.	TAER Pegg.	TAER Medio
Java	Standard	4080 ms	13173 ms	5824 ms
Java	Non Affidabile	25419 ms	46445 ms	38252 ms
Emulatore	Standard	4525 ms	16157 ms	7880 ms
Nexus S	Standard	2494 ms	5555 ms	3298 ms
Nexus S	Non Affidabile	18988 ms	36199 ms	26879 ms

Tabella 5.1: Tabella riassuntiva dei risultati temporali per il **Gestore HTTP**

Piattaforma	Rete	TAER Migl.	TAER Pegg.	TAER Medio
Java	Standard	3961 ms	12245 ms	6161 ms
Java	Non Affidabile	26134 ms	47322 ms	36226 ms
Emulatore	Standard	6240 ms	16420 ms	8406 ms
Nexus S	Standard	2497 ms	7423 ms	3482 ms
Nexus S	Non Affidabile	17875 ms	36116 ms	25016 ms

Tabella 5.2: Tabella riassuntiva dei risultati temporali per il **Gestore SOAP**

Prima di procede con l'analisi si vuole sottolineare che i tempi misurati per le reti standard sono migliori quando misurati sul dispositivo fisico e peggiori se si prendono in considerazione l'emulatore o il computer portatile con piattaforma Java e ciò si verifica anche nelle misurazioni per le reti non affidabili. I protocolli usati sono gli stessi come anche il nucleo del codice che viene eseguito, inoltre abbiamo verificato che le misurazioni temporali sono indipendenti dalla versione del sistema operativo mobile usato. Alla luce di tutto ciò, sembrerebbe che i risultati siano influenzati da altri fattori, probabilmente più a basso livello e direttamente dipendenti dall'hardware. Ciò sembra essere confermato da alcune altre verifiche effettuate su altri dispositivi fisici, ma qui non riportate in quanto incomplete.

Controllando i risultati ottenuti nelle tabelle estese, sembra essere confermato quanto esposto precedentemente: misurazioni alternate mediano la variazione di prestazioni delle reti wireless e pertanto i valori medi TAER ottenuti per i due gestori sono più vicini fra loro rispetto a quelli mi-

surati con interrogazioni sequenziali. L'ultima osservazione è forse la più importate fra quelle esposte fin'ora: mentre il gestore SOAP ha prestazioni migliori di quello HTTP quando le reti considerate sono inaffidabili, si riscontra il viceversa quando le reti sono standard. Il protocollo SOAP appare dunque leggermente meno efficiente di quello HTTP, ma più robusto in caso di problemi di rete.

Java MiniVidiJava - Rete STD

Gestore HTTP

Numero di invii per ciascuna prova						20	
Dimensione XML inviato						190491 byte	
NB TEMPI AER: Andata + Esecuzione sul server + Ritorno							
				TEMPI AER (ms)			
Data	Ora	Rete	Affidabilità	Migliore	Peggiore	Media	
16/11/2011	18.00	Alma Wifi 25%	90,00%	3483	18000	6014	SEQ
17/11/2011	12.20	Alma Wifi 25%	100,00%	2328	10437	3263	SEQ
17/11/2011	15.05	Alma Wifi 25%	100,00%	3837	17056	6212	SEQ
18/11/2011	10.00	Alma Wifi 25%	100,00%	3492	16664	6949	SEQ
18/11/2011	15.00	CIPI Wifi 80%	100,00%	6263	14744	7272	SEQ
18/11/2011	16.33	CIPI Wifi 80%	100,00%	6148	15571	7104	SEQ
21/11/2011	11.55	Alma Wifi 25%	100,00%	6278	15573	7926	SEQ
23/11/2011	08.50	Alma Wifi 25%	100,00%	2751	5175	3270	SEQ
28/11/2011	14.00	Alma Wifi 25%	100,00%	4043	15564	6220	ALT
28/11/2011	14.50	Alma Wifi 25%	100,00%	3586	10074	5893	ALT
07/12/2011	12.30	Alma Wifi 70%	100,00%	4107	13228	6410	ALT
07/12/2011	14.55	Alma Wifi 70%	100,00%	2582	6473	3377	ALT
MEDIE PESATE				4080	13173	5824	

Gestore SOAP

Numero di invii per ciascuna prova						20	
Dimensione XML inviato						190491 byte	
NB TEMPI AER: Andata + Esecuzione sul server + Ritorno							
				TEMPI AER (ms)			
Data	Ora	Rete	Affidabilità	Migliore	Peggiore	Media	
16/11/2011	18.00	Alma Wifi 25%	100,00%	3776	13059	5587	
17/11/2011	12.20	Alma Wifi 25%	100,00%	2220	5620	2834	
17/11/2011	15.05	Alma Wifi 25%	100,00%	3306	11805	5624	
18/11/2011	10.00	Alma Wifi 25%	100,00%	4374	17597	8645	
18/11/2011	15.00	CIPI Wifi 80%	100,00%	5865	9424	6879	
18/11/2011	16.33	CIPI Wifi 80%	100,00%	5796	12407	7532	
21/11/2011	11.55	Alma Wifi 25%	100,00%	5743	14054	8439	
23/11/2011	08.50	Alma Wifi 25%	100,00%	2588	3737	3038	
28/11/2011	14.00	Alma Wifi 25%	100,00%	3566	16913	6878	
28/11/2011	14.50	Alma Wifi 25%	100,00%	3723	14127	6557	
07/12/2011	12.30	Alma Wifi 70%	100,00%	3975	21026	8091	
07/12/2011	14.55	Alma Wifi 70%	100,00%	2603	7174	3828	
MEDIE PESATE				3961	12245	6161	

Tabella 5.3: Risultati TAER per MiniVidiJava su rete Wi-Fi STD

Java MiniVidiJava - Rete Non Affidabile

Gestore HTTP

Numero di invii per ciascuna prova						20	
Dimensione XML inviato						190491 byte	
NB TEMPI AER: Andata + Esecuzione sul server + Ritorno					TEMPI AER (ms)		
Data	Ora	Rete	Affidabilità	Migliore	Peggiore	Media	
17/11/2011	18.30	SDA Wifi	100,00%	22048	33490	27436	SEQ
17/11/2011	21.00	SDA Wifi	100,00%	22518	29532	25186	SEQ
22/11/2011	23.10	SDA Wifi	100,00%	26836	52839	43697	SEQ
28/11/2011	18.00	SDA Wifi	100,00%	28152	58454	47630	ALT
28/11/2011	19.50	SDA Wifi	100,00%	27541	57912	47312	ALT
MEDIE PESATE				25419	46445	38252	

Gestore SOAP

Numero di invii per ciascuna prova						20	
Dimensione XML inviato						190491 byte	
NB TEMPI AER: Andata + Esecuzione sul server + Ritorno					TEMPI AER (ms)		
Data	Ora	Rete	Affidabilità	Migliore	Peggiore	Media	
17/11/2011	18.30	SDA Wifi	100,00%	21711	30713	24493	
17/11/2011	21.00	SDA Wifi	100,00%	21600	39121	27171	
22/11/2011	23.10	SDA Wifi	100,00%	34749	55003	44439	
28/11/2011	18.00	SDA Wifi	100,00%	26240	55782	43670	
28/11/2011	19.50	SDA Wifi	100,00%	26372	55993	41356	
MEDIE PESATE				26134	47322	36226	

Tabella 5.4: Risultati TAER per MiniVidiJava su rete Wi-Fi RNA

Android TransmissionVIDI su EMU - Rete STD

Gestore HTTP

Numero di invii per ciascuna prova	20
Dimensione XML inviato	190491 byte

NB TEMPI AER: Andata + Esecuzione sul server + Ritorno

Data	Ora	Rete	Affidabilità	TEMPI AER (ms) Migliore	Peggiore	Media	
18/11/2011	10.20	Alma Wifi 25%	100,00%	3912	12389	8428	SEQ
18/11/2011	11.10	Alma Wifi 25%	100,00%	3397	14890	5273	SEQ
18/11/2011	16.30	CIPI Wifi 80%	100,00%	6200	10181	6855	SEQ
23/11/2011	15.00	Alma Wifi 25%	100,00%	4534	22885	9459	SEQ
23/11/2011	17.25	Alma Wifi 25%	100,00%	3833	10622	6810	SEQ
24/11/2011	09.55	Alma Wifi 25%	100,00%	3357	18854	6343	SEQ
24/11/2011	15.00	CIPI Wifi 80%	100,00%	6442	23275	11992	SEQ
MEDIE PESATE				4525	16157	7880	

Gestore SOAP

Numero di invii per ciascuna prova	20
Dimensione XML inviato	190491 byte

NB TEMPI AER: Andata + Esecuzione sul server + Ritorno

Data	Ora	Rete	Affidabilità	TEMPI AER (ms) Migliore	Peggiore	Media
18/11/2011	10.20	Alma Wifi 25%	100,00%	6269	15314	9075
18/11/2011	11.10	Alma Wifi 25%	100,00%	5155	9231	6815
18/11/2011	16.30	CIPI Wifi 80%	100,00%	7191	10974	7715
23/11/2011	15.00	Alma Wifi 25%	100,00%	5079	18962	8196
23/11/2011	17.25	Alma Wifi 25%	100,00%	6009	17309	8696
24/11/2011	09.55	Alma Wifi 25%	100,00%	4800	18639	6170
24/11/2011	15.00	CIPI Wifi 80%	100,00%	9178	24509	12177
MEDIE PESATE				6240	16420	8406

Tabella 5.5: Risultati TAER per Tr.Vidi EMU su rete Wi-Fi STD

Android TransmissionVidi su NEXUS S - Rete STD

Gestore HTTP										Gestore SOAP									
Numero di invii per ciascuna prova						20				Numero di invii per ciascuna prova						20			
Dimensione XML inviato						190491 byte				Dimensione XML inviato						190491 byte			
NB TEMPI AER: Andata + Esecuzione sul server + Ritorno										NB TEMPI AER: Andata + Esecuzione sul server + Ritorno									
				TEMPI AER (ms)										TEMPI AER (ms)					
Data	Ora	Rete	Affidabilità	Migliore	Peggiore	Media				Data	Ora	Rete	Affidabilità	Migliore	Peggiore	Media			
01/12/2011	09.10	Alma Wifi 70%	100,00%	2151	5407	2995	ALT			01/12/2011	09.10	Alma Wifi 70%	100,00%	2265	5402	2964			
01/12/2011	09.40	Alma Wifi 70%	100,00%	2323	5709	2871	ALT			01/12/2011	09.40	Alma Wifi 70%	100,00%	2324	3636	2826			
01/12/2011	09.50	Alma Wifi 70%	100,00%	2542	5974	3521	ALT			01/12/2011	09.50	Alma Wifi 70%	100,00%	2355	7137	3606			
01/12/2011	11.20	Alma Wifi 70%	100,00%	3082	7385	3957	ALT			01/12/2011	11.20	Alma Wifi 70%	100,00%	2803	24315	4891			
01/12/2011	11.35	Alma Wifi 70%	100,00%	3175	8343	4648	ALT			01/12/2011	11.35	Alma Wifi 70%	100,00%	2967	10619	4827			
01/12/2011	17.30	Alma Wifi 70%	100,00%	2697	6677	4089	ALT			01/12/2011	17.30	Alma Wifi 70%	100,00%	2954	6757	4102			
01/12/2011	17.40	Alma Wifi 70%	100,00%	2431	5679	3043	ALT			01/12/2011	17.40	Alma Wifi 70%	100,00%	2448	5964	3356			
06/12/2011	17.45	Alma Wifi 70%	100,00%	2151	5407	2895	ALT			06/12/2011	17.45	Alma Wifi 70%	100,00%	2265	5402	2964			
07/12/2011	15.30	Alma Wifi 70%	100,00%	2729	5740	3469	ALT			07/12/2011	15.30	Alma Wifi 70%	100,00%	2705	6486	3626			
07/12/2011	15.35	Alma Wifi 70%	100,00%	2903	6311	4230	ALT			07/12/2011	15.35	Alma Wifi 70%	100,00%	2810	8499	4427			
13/12/2011	10.00	SilverWifi 80%	100,00%	1880	2034	1921	ALT			13/12/2011	10.00	SilverWifi 80%	100,00%	2055	2470	2106			
13/12/2011	10.05	SilverWifi 80%	100,00%	1859	1991	1932	ALT			13/12/2011	10.05	SilverWifi 80%	100,00%	2011	2393	2086			
MEDIE PESATE				2494	5555	3298				MEDIE PESATE				2497	7423	3482			

Tabella 5.6: Risultati TAER per Tr.Vidi NEXUS S su rete Wi-Fi STD

Android TransmissionVidi su NEXUS S - Rete Non Affidabile

Gestore HTTP

Numero di invii per ciascuna prova								20
Dimensione XML inviato								190491 byte
NB TEMPI AER: Andata + Esecuzione sul server + Ritorno								
					TEMPI AER (ms)			
Data	Ora	Rete	Affidabilità		Migliore	Peggiore	Media	
01/12/2011	00.10	SDA Wifi	100,00%		22800	41498	31353	ALT
19/12/2011	12.20	Alma Wifi 40%	45,00%		20570	40385	25167	ALT
19/12/2011	12.20	Alma Wifi 40%	80,00%		2785	7530	4529	ALT
06/12/2011	21.45	SDA Wifi	100,00%		21340	40344	31232	ALT
07/12/2011	21.30	SDA Wifi	100,00%		22174	41673	31578	ALT
07/12/2011	22.35	SDA Wifi	100,00%		21886	42333	32003	ALT
MEDIE PESATE					18988	36199	26879	

Gestore SOAP

Numero di invii per ciascuna prova							20
Dimensione XML inviato							190491 byte
NB TEMPI AER: Andata + Esecuzione sul server + Ritorno							
				TEMPI AER (ms)			
Data	Ora	Rete	Affidabilità	Migliore	Peggiore	Media	
01/12/2011	00.10	SDA Wifi	100,00%	21934	41499	28738	
19/12/2011	12.20	Alma Wifi 40%	45,00%	15399	38464	23236	
19/12/2011	12.20	Alma Wifi 40%	75,00%	3214	14007	5606	
06/12/2011	21.45	SDA Wifi	100,00%	21556	38923	28756	
07/12/2011	21.30	SDA Wifi	100,00%	20234	40235	27897	
07/12/2011	22.35	SDA Wifi	100,00%	19885	39332	30034	
MEDIE PESATE				17875	36116	25016	

Tabella 5.7: Risultati TAER per Tr.Vidi NEXUS S su rete Wi-Fi RNA

5.2 Sviluppi futuri

ViDiApp e le altre attività interne a ViDiTrust sono in continua evoluzione e molti sono i cambiamenti che si stanno apportando all'applicazione come pure ai server predisposti a fornire il servizio remoto di autenticazione.

Limitatamente al lavoro svolto e discusso nel quarto e quinto capitolo, si vuole innanzi tutto indagare e risolvere il problema di latenza che a volte si verifica durante i controlli di rete eseguiti nel Modulo di Inizializzazione e nel Modulo di Trasmissione, e relativi alla verifica della disponibilità del servizio remoto.

Si vuole inoltre eseguire un maggior numero di test, temporali e non, finalizzati a verificare la veridicità dell'affermazione riportata nel paragrafo 5.1.4.1, ovvero che il protocollo SOAP è meno efficiente ma più robusto del protocollo HTTP. In verità sono già in corso ulteriori sperimentazioni atte a misurare se esistono overhead significativi introdotti dai protocolli sugli oggetti da movimentare e si è anche cercato di effettuare invii di messaggi di dimensioni maggiori (2 MB) per valutarne tempi e risposte.

Un altro obiettivo è quello di indagare ulteriori protocolli di comunicazione, ad esempio quelli ispirati all'architettura REST ovvero *Representional State Transfer* e relativi a servizi leggeri e distribuiti.

Un'altra importante area da sottoporre a test temporali è quella relativa alle trasmissione via rete cellulare per poter osservare come si comportano i due gestori sviluppati e se in questo caso le prestazioni risultino essere compatibili come quelle verificate per le reti wireless oppure no.

In base ai risultati di queste ed altre indagini sarebbe anche interessante tentare di sviluppare un insieme di metodi che a runtime siano in grado di stabilire quale gestore sia da preferire in base alle condizioni della rete rilevate: mirare a un trasferimento efficiente quando si dispone di una connessione standard, realizzare una comunicazione robusta quando si verificano dei problemi.

Un altro aspetto su cui si intende operare riguarda l'estensione della compatibilità dell'applicazione: in varie occasioni si è verificato che il co-

dice funzionante su un certo dispositivo andasse in crash su altri, ma non a causa di bug o mancanze di risorse, quanto piuttosto per una loro diversa gestione, non sempre conforme o attesa rispetto a quanto promesso, ad esempio, dalle API Android. Secondo questa linea di pensiero sarebbe opportuno iniziare a testare l'applicazione anche sulla nuova versione della piattaforma Android, la 4.0.x, che promette una migliore gestione delle connettività di rete e nuovi controlli e funzionalità per la fotocamera [Dev12].

Infine, il software ViDiTrust nato per postazioni fissa e poi migrato su dispositivi mobili, raggiunta la maturità necessaria vedrà la luce anche su piattaforme differenti da Android, come IOS di Apple o BlackBerry OS di RIM.

5.3 Conclusioni

Quest'ultimo capitolo ha esaminato il comportamento dei gestori HTTP e SOAP per mezzo dell'introduzione di alcuni test temporali. Prima di analizzare i dati relativi alle misurazioni effettuate, si è voluto offrire al lettore una migliore comprensione su come i test siano stati ideati, secondo quali linee guida operino e quali parametri operativi occorresse tenere in considerazione e quali no. Dopo aver indicato le piattaforme su cui sono stati effettuati i test, si è presentato il software sviluppato per eseguire le misurazioni e se ne sono discusse le principali caratteristiche per mezzo di alcuni commenti al codice. Successivamente sono stati riportati i risultati ottenuti classificandoli per piattaforma (Java, Android emulato, Android su Nexus S) e per tipo di rete wireless disponibile (standard o non affidabile). Dopo aver introdotto le tabelle relative ai risultati, alcune di esse realizzate a scopi riassuntivi, e dopo aver illustrato la differenza fra test sequenziali e alternati, si è tentato di interpretarne i valori ottenuti. L'osservazione più importate che è stata dedotta da quanto misurato sperimentalmente è che seppure i due modi di comunicare (via HTTP o via SOAP) abbiano fornito prestazioni molto simili, il gestore HTTP sembra essere più efficiente rispetto a quello SOAP quando la connessione Wi-Fi è

"buona" e cioè non manifesta latenze nel realizzare le connessioni, ha una banda larga, il segnale è potente etc, viceversa il SOAP è più robusto per connessioni problematiche.

La conclusione del capitolo è stata dedicata agli sviluppi futuri previsti per l'applicazione Android. Relativamente a quanto sviluppato in questo lavoro di tesi, si vorrebbe indagare e correggere alcuni problemi lasciati aperti come le rare e anomale latenze introdotte dai controlli di rete, oppure introdurre nuovi protocolli di comunicazione e verificarne con test intensivi e mirati le prestazioni, anche su rete cellulare. Le ricerche appena introdotte servirebbero per sviluppare un modulo che a tempo di esecuzione sia capace di invocare il miglior gestore possibile fra quelli disponibili in base alla rete in uso e alla sue caratteristiche. Per concludere si vorrebbe iniziare a testare e adattare l'applicazione su nuove piattaforme come Android 4.0.x, IOS e BlackBarry OS.

Conclusioni

Il lavoro di tesi presentato su un'applicazione per dispositivi mobili basati su sistema Android ha realizzato gli obiettivi prefissati. A partire dalla tecnologia innovativa nel campo dei sistemi di autenticazione passiva proposta dall'azienda ViDiTrust, i vari interventi eseguiti lato client del processo di verifica hanno contribuito alla sua ingegnerizzazione.

La realizzazione di nuove funzionalità quali l'identificazione e la localizzazione del dispositivo mobile, l'inserimento dei controlli di rete necessari a verificare la disponibilità della connessione e del servizio di autenticazione, nonché l'implementazione di un sistema di comunicazione remoto alternativo a quello originariamente in uso, hanno da un lato arricchito le potenzialità del software, dall'altro ottenuto un miglioramento della qualità del servizio in rispetto dell'utente e della sua privacy.

Lo sviluppo delle varie parti ha reso necessario un impegno profuso e di ampio spettro rivolto a una interessante ricerca di metodi e tecnologie utili a raggiungere gli scopi previsti, ma anche fondato sull'implementazione delle soluzioni escogitate su una piattaforma innovativa e in continua evoluzione quale è Android. Seppur in modo limitato, per questioni legali e di spazio, si è tentato di offrire al lettore uno sguardo d'insieme e con un certo livello di astrazione sugli argomenti da comprendere e affrontare, ma al tempo stesso anche un "supporto pratico" realizzato per mezzo di codice e descrizioni tecniche.

Il lavoro di tesi è stato in buona parte rivolto al confronto delle prestazioni di due moduli di comunicazione alternativi. Il primo, già sviluppato dall'azienda ViDiTrust, è basato sul protocollo SOAP e implementato attraverso una libreria open source (kSOAP2), il secondo è stato realizza-

to durante la fase operativa della tesi e si avvale del protocollo HTTP, in particolare del suo metodo di request POST.

Consci del fatto che il software sviluppato per eseguire i test temporali debba essere migliorato e specializzato, che il numero delle prove aumenti in quantità e qualità e che occorra effettuarle anche su rete cellulare oltre che su quella wireless, i risultati ottenuti sono incoraggianti e inducono a pensare che il protocollo HTTP sia più efficiente di quello SOAP, ma anche meno robusto qualora si usino connessioni poco stabili o dalle prestazioni altalenanti.

Quest'ultimo aspetto assieme alla risoluzione di alcuni problemi verificatisi in corso d'opera, rappresentano parte degli sviluppi futuri che si pensa di indagare nel breve termine. Fra i nuovi obiettivi di più ampio respiro si annoverano il *porting* dell'applicativo su piattaforme più evolute come Android 4.0.x o su diversi sistemi operativi mobili quali IOS e BlackBerry OS.

L'obiettivo ViDiTrust di combattere la piaga della contraffazione per mezzo di tecnologie innovative è incoraggiato dagli approcci usati verso il produttore che a basso costo riesce a tutelare i proprio prodotti, come pure verso l'utente finale che può accertarsi della loro legalità in autonomia, semplicemente con un'*app* e prima dell'acquisto. Il lavoro di tesi ha operato con queste finalità, introducendo nuove funzioni e ottenendo risultati misurabili.

Bibliografia

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services: concepts, architectures and applications*. Springer Verlag, 2004.

[Age12] ESA: European Space Agency. What is egnos? `http://www.esa.int/esaNA/egnos.html`, 2012. [Online; in data 01-marzo-2012].

[BCC+02] R. J. Brunner, F. Cohen, F. Curbera, D. Govoni, S. Haines, M. Kloppmann, B. Marchal, K.S. Morrison, A. Ryman, J. Weber, and M. Wutka. *Java Web Service*. Apogeo Editore Srl, 2002.

[Bia09] Ludovico Biagi. I fondamentali del gps. *Geomatics Workbooks*, 8, 2009.

[BP09] O. Babaoglu and P. Paladino. La steganografia e i suoi molteplici usi. *Università degli Studi di Bologna*, 2009.

[Bra02] Tania Branigan. The key to the mobile phone theft epidemic. `http://www.guardian.co.uk/mobile/article/0,2763,643752,00.html`, Febbraio 2002. [Online; in data 17-febbraio-2012].

[Bur09] Ed Burnette. *Hello, Android: Introducing Google's Mobile Development Platform*. Pragmatic Bookshelf, 2nd edition, 2009.

[Car10] Massimo Carli. *Android: guida per lo sviluppatore*. Apogeo Editore Srl, 2010.

[Cor10] Antonio Corradi. Corso di reti di calcolatori. *Università degli Studi di Bologna*, 2010.

[Cor12] Microsoft Corporation. Market place. `http://www.windowsphone.com/it-IT/marketplace`, Febbraio 2012. [Online; in data 17-febbraio-2012].

[Dev12] Android Developers. Android developers home page. `http://developer.android.com/`, Febbraio 2012. [Online; in data 10-febbraio-2012].

[dI12] Ministro degli Interni. Carta d'identità elettronica - cie. `http://www.interno.it/mininterno/export/sites/default/it/temi/servizi_demografici/scheda_006.html`, Febbraio 2012. [Online; in data 15-febbraio-2012].

[DoD05] USA Department of Defense. Federal radionavigation plan. 2005.

[DoD08] USA Department of Defense. Global positioning system, standard positioning service, performance standard. 2008.

[DSG09] Alfredo De Santis and Daniele Giardino. Corso di sicurezza su reti 2 / digital watermarking. *Università degli Studi di Salerno*, 2009.

[dU09] URP degli URP. Carta d'indentità elettronica. `http://www.urp.it/Sezione.jsp?idSezione=1719`, Settembre 2009. [Online; in data 16-febbraio-2012].

[For99] IETF: Internet Engineering Task Force. Hypertext transfer protocol – http/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, 1999. [Online; in data 05-marzo-2012].

[Fou12] IDF: The Internationl DOI Foundation. The doi® system. `http://www.doi.org/`, Febbraio 2012. [Online; in data 17-febbraio-2012].

[Gar07] Maurizio Garatti. Identificazione della rete. `http://digilander.libero.it/garats/gsm/data/Net_monitor_info/netmon11.html`, Dicembre 2007. [Online; in data 15-febbraio-2012].

[GL01] Y. Gao and Z. Liu. Differential gps positioning over internet. *Journal of Geospatial Engineering*, 3(1):1–8, 2001.

[HB01] Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *IEEE*, 2001.

[HS12] Stefan Haustein and James Seigel. ksoap 2 home page. `ksoap2.sourceforge.net/`, Febbraio 2012. [Online; in data 29-febbraio-2012].

[Inc02] Ekahau Inc. Ekahau positioning engine 2.0 - technology white paper, 2002.

[Inc12a] Apple Inc. App store. `http://store.apple.com/it`, Febbraio 2012. [Online; in data 10-febbraio-2012].

[Inc12b] Google Inc. Andoroid market. `https://market.android.com/`, Febbraio 2012. [Online; in data 17-febbraio-2012].

[Lei10] Raphael Leiteritz. Google submission dpa's wifi collection. `http://www.google.com/googleblogs/pdfs/google_submission_dpas_wifi_collection.pdf`, Aprile 2010. [Online; in data 10-febbraio-2012].

[MOJ95] G.J. Morgan-Owen and G.T. Johnston. Differential gps positioning. *ELECTRONICS & COMMUNICATION ENGINEERING JOURNAL*, 1995.

[Mon05] G. Montemanari. Steganografia, l'arte della scrittura nascosta. *Università degli Studi di Bologna*, 2005.

[PD08] L.L. Peterson and B.S. Davie. *Reti di calcolatori*. Apogeo Editore Srl, 2nd edition, 2008.

[Pel09] Carlo Pelliccia. Programmazione android. `www.informatica.uniroma2.it/upload/2009/LIS/`, 2009. [Online; in data 10-febbraio-2012].

[Srl11] Altroconsumo Nuove Edizioni Srl. Smartphone: la parola ai consumatori. `http://www.altroconsumo.it/hi-tech/telefoni-cellulari/speciali/smartphone-la-parola-ai-consumatori`, Marzo 2011. [Online; in data 17-febbraio-2012].

[Sys12] Handle System. Handle system home page. `www.handle.net/`, Febbraio 2012. [Online; in data 17-febbraio-2012].

[Taj05] Zeno Tajoli. Doi: uno strumento per costruire la biblioteca digitale. *Bollettino AIB*, 2005.

[VD09] F.S.T. Van Diggelen. *A-GPS: assisted GPS, GNSS, and SBAS*. Artech House Publishers, 2009.

[ViD10] ViDiTrust. Viditrust home page. `http://www.viditrust.com/`, 2010. [Online; in data 05-marzo-2012].

[ViD11] ViDiTrust. Procedimento di marcatura anticontraffazione di prodotti a stampa e relativo sistema, Settembre 2011. Depositato alla CCIAA di Firenze il 27/09/2011, col numero di domanda FI2011A000207.

[Wik12] Wikipedia. Banconote euro — wikipedia, l'enciclopedia libera. `http://it.wikipedia.org/w/index.php?title=Banconote_euro&oldid=46489007`, 2012. [Online; in data 20-febbraio-2012].

Elenco delle figure

Elenco delle tabelle