Università degli Studi di Pisa

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

PROGETTO DI UN'APPLICAZIONE PER IL PAGAMENTO E IL CONTROLLO DELLE AREE DI SOSTA MEDIANTE TELEFONO CELLULARE

Candidato: Mancini Alessandro .....

Relatori: Avvenuti Marco .....

Anastasi Giuseppe .....

Vecchio Alessio .....

Anno Accademico 2006/2007

Ai miei genitori

Sommario

La tesi tratta lo studio di applicazioni basate su marcatori *visualtag* (simboli bidimensionali stampabili su superfici) in cui è possibile codificare dati di varia natura. Nell'ambito di dispositivi mobili è possibile identificare un oggetto reale acquisendo l'immagine del *visualtag* con la fotocamera incorporata nel dispositivo stesso, senza ricorrere a strumenti di lettura dedicati. L'informazione ricavata dalla decodifica dell'immagine permette di sfruttare le funzionalità associate a quell'oggetto. È stata realizzata un'applicazione per telefoni cellulari che dimostra come i marcatori *visualtag* possono rendere più facile e intuitivo la gestione di aree di sosta.

Abstract

This thesis analyzes the study of applications based on *visualtag* markers (bidimensional symbols printable on surfaces) in which it is possible to encode data of varied nature. In a mobile context is possible to identify a real object by acquiring the *visualtag* image using the built-in device camera, without any dedicated reading devices. The information obtained from the decoded image allows you to take advantage of the functionalities associated to that object. It has been realized an application for mobile phones that demonstrates as the *visualtag* markers can render the management of parking areas easier and comfortable.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

La maggior parte degli oggetti di uso quotidiano sta evolvendo. In futuro questi saranno dotati di unità di calcolo e software per eseguire o controllare una moltitudine di attività, potranno comunicare tra loro e interagire con l'ambiente circostante. In questo scenario e con la massiccia diffusione delle reti di comunicazione (e di Internet), avremo una grande infrastruttura costituita da molti oggetti, diversi tra loro (per capacità di calcolo o interfaccia).

L'evoluzione del settore informatico deve sviluppare nei prossimi anni la possibilità di realizzare sistemi software che sfruttino le potenzialità offerte da questi nuovi dispositivi. La sfida è quella di voler definire e sfruttare sistemi, costituiti da un numero crescente di entità mobili, che si configurino dinamicamente e che interagiscano con il loro ambiente. Questo ambito applicativo prende il nome di *Ubiquitous Computing*.

Lo scopo centrale dell'*Ubiquitous Computing* è l'invisibilità, nel senso che non è necessario pensare continuamente che si sta interagendo con un sistema informativo. Dopo aver appreso il suo funzionamento, un oggetto cessa di apparire come strumento informatico, è *"letteralmente visibile, efficacemente invisibile"*; nello stesso modo in cui un carpentiere utilizza un martello senza pianificare ogni singolo movimento, oppure quando si dà un'occhiata a un cartello stradale, si acquisisce l'informazione senza pensare coscientemente

all'atto di "lettura".

> "*Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.*[14]
>
> **Mark Weiser**, 1996

Nel quotidiano abbiamo a che fare con dispositivi digitali, dotati di un'unità di calcolo e software *embedded*, eppure quando utilizziamo una lavatrice, un microonde o un distributore automatico di bibite, non pensiamo coscientemente che stiamo interagendo con un dispositivo informatico.

Durante lo svolgimento di questa tesi è stata realizzata un'applicazione basata sul sistema di interazione tra computer e oggetti attraverso il telefono cellulare. L'idea è quella di utilizzare questo dispositivo per interagire con ciò che si trova intorno a noi. La scelta è ricaduta su questo apparato perché ha il vantaggio di essere molto diffuso e più facile da trasportare rispetto a un palmare o un computer. Inoltre non va dimenticato che la maggior parte delle persone è ormai entrata in confidenza con questo strumento, e che, rispetto ad altri sistemi, è praticamente sempre in tasca al suo utilizzatore.

Il telefono cellulare ha tutti gli elementi per essere considerato un piccolo elaboratore: capacità di calcolo, memoria, display per la visualizzazione delle elaborazioni, tastiera e altri dispositivi di ingresso dati. È in grado di comunicare con altri dispositivi che si trovano nel raggio di pochi metri, grazie allo standard *bluetooth*, o effettuando connessioni dati via Internet, grazie alla rete del gestore di telefonia.

Un altro vantaggio è quello di poter installare facilmente nuove applicazioni software, in modo da offrire nuovi servizi evoluti e innovativi.

La fotocamera digitale è un dispositivo integrato ormai in tutti i nuovi modelli, attraverso questa si possono acquisire informazioni sugli oggetti circostanti in modo estremamente intuitivo: basta puntare il telefono e scattare una foto. L'iterazione con l'ambiente può avvenire attraverso la cattura di etichette opportunamente codificate denominate *visualtag*, che identificano in modo univoco un certo oggetto. L'utente potrà interagire con tale oggetto attraverso un'applicazione scaricata da Internet e installata sul proprio telefono.

Capitolo 1

Tecnologie pervasive

Nel corso degli ultimi anni è cresciuta la necessità di poter utilizzare servizi informativi in modo semplice, in qualsiasi momento e dovunque ci si trovi. Da qui nasce la necessità di adattare le reti informatiche alla fruizione di questi servizi in modo adeguato al loro contesto.

Uno *smartphone* è un dispositivo portatile che abbina funzionalità di gestione di dati personali e di telefono. Può derivare dall'evoluzione di un PDA a cui si aggiungono funzioni di telefono, o, viceversa, di un telefono mobile a cui si aggiungono funzioni di PDA. La caratteristica più interessante degli *smartphone* è la possibilità di installarvi altri programmi applicativi, che aggiungono nuove funzionalità. Questi programmi possono essere sviluppati dal produttore, dallo stesso utilizzatore, o da terze parti.

Gli smartphone possono migliorare l'uso dei cosiddetti *site-specific services* (servizi elettronici o applicazioni che risiedono in uno specifico luogo). Servizi dipendenti dal contesto esistono già sotto forma di distributori automatici di biglietti, chioschi elettronici, cataloghi interattivi, e così via. Tuttavia l'integrazione degli *smartphones* con questi servizi può aumentare le funzionalità del servizio stesso e ridurne i costi di gestione.

Questo tipo di approccio offre diversi benefici. Utilizzando le informazioni memorizzate nello *smartphone*, i servizi elettronici possono adeguare automaticamente le loro azioni alle necessità di quel particolare utilizzatore. L'utente può memorizzare sul proprio telefono le informazioni risultate dall'interazione con un servizio (ad esempio scaricando del testo o l'immagine di una mappa). Il servizio potrebbe inviare informazioni o aggiornamenti sul telefono dell'utente tramite brevi messaggi di testo (SMS) o messaggi multimediali contenenti testo, immagini o video (MMS), infine l'utente non dovrebbe più fare la fila per accedere a un singolo apparato (si pensi ad esempio ad un distributore di biglietti) in quanto più utilizzatori potrebbero collegarsi simultaneamente allo stesso servizio tramite i loro telefoni.

1.1 Identificazione degli oggetti in ambiente mobile

Per consentire l'identificazione degli oggetti ovunque essi si trovino si possono utilizzare delle etichette con marcatori *visualtag* che rappresentino il loro codice identificativo.

A differenza di altri sistemi di identificazione (come ad esempio i codici a barre), il *visualtag* si basa su forme geometriche ed ha il vantaggio di avere meno limitazioni in termini di angolo o distanza di visione. I simboli possono quindi essere facilmente letti sfruttando la fotocamera installata nel telefono cellulare.

Tenendo conto delle diverse condizioni di luminosità è possibile elaborare l'immagine ed estrarre le caratteristiche dell'etichetta. Non è necessario un apparato di lettura dedicato, perché il telefono è in grado di acquisire l'immagine e decodificarla. (Fig. 1.1).

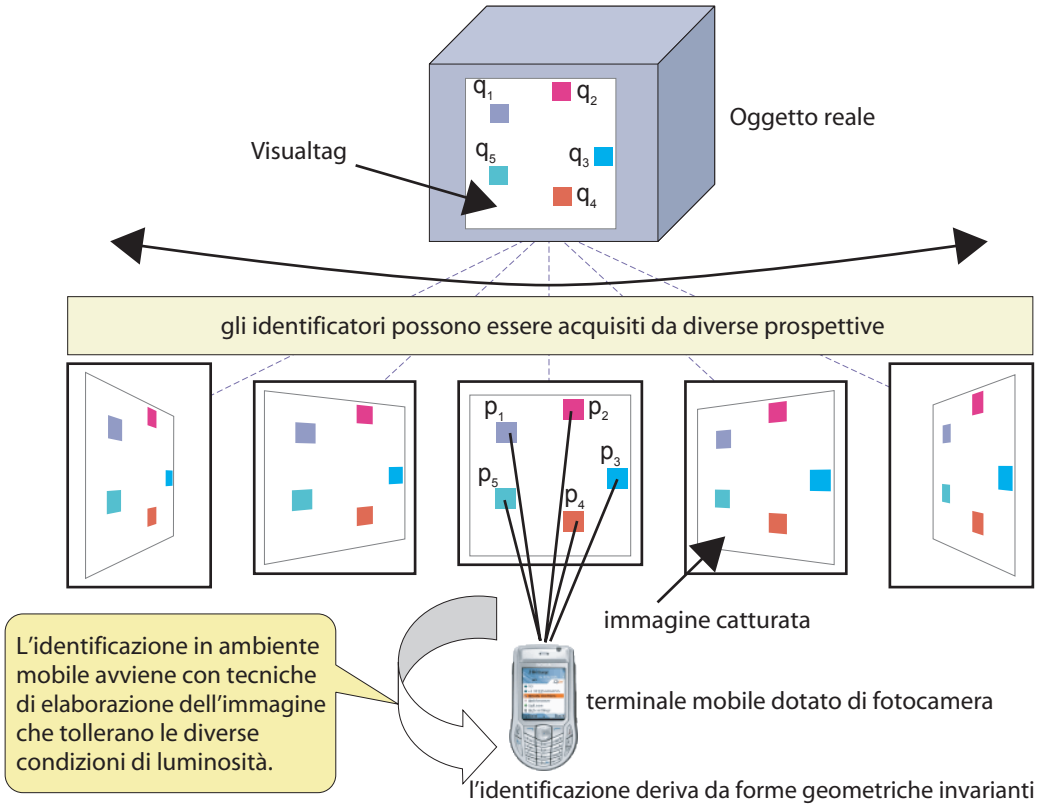


Figura 1.1: Riconoscimento dell'oggetto tramite Visualtag

1.2 Utilizzo di tag per l'accesso a funzioni relative a una locazione

In una ricerca apparsa sulla rivista IEEE Pervasive Computing [9] descrive un sistema basato su tag di forma circolare, questi permettono di accedere tramite telefono alle funzioni relative a un oggetto o una locazione.

La connessione, tra il cellulare e un oggetto che offre un servizio dipendente dal contesto, dovrà essere di tipo *wireless*. Si possono scegliere varie soluzioni, dalla connettività dati offerta dall'operatore telefonico (come GPRS o UMTS), alle soluzioni *punto-a-punto* di corto raggio, come *Bluetooth* o *infrarossi*. Tutte queste soluzioni permettono di scambiare dati tra telefono e oggetto e visto che ci troviamo in prossimità del servizio, è ragionevole utilizzare un collegamento a corto raggio come il Bluetooth, evitando di impegnare la rete del gestore di telefonia (che di solito fa pagare qualcosa per il traffico dati generato).

Il riconoscimento degli oggetti descritto sulla pubblicazione si basa su marcatori circolari in grado di codificare 63 bit di dati; questa quantità di informazione è sufficiente per contenere l'indirizzo MAC di un dispositivo *bluetooth* (48 bit) oppure l'indirizzo IP (32 bit) e porta TCP (16 bit) di un host su Internet (fig. 1.2). I bit rimanenti possono contenere informazioni utili all'applicazione specifica.

La ricerca di un dispositivo Bluetooth è un'operazione che può rivelarsi lenta, non è raro dover attendere anche più di 30 secondi. Ricavando dal *visualtag* l'*indirizzo fisico* (MAC) del dispositivo con cui vogliamo interagire, non sarà più necessario effettuare la ricerca prevista dallo standard riducendo di molto i tempi di connessione. L'indirizzo MAC permette anche che non si crei confusione nel caso siano presenti più dispositivi *bluetooth* o che un'ap-

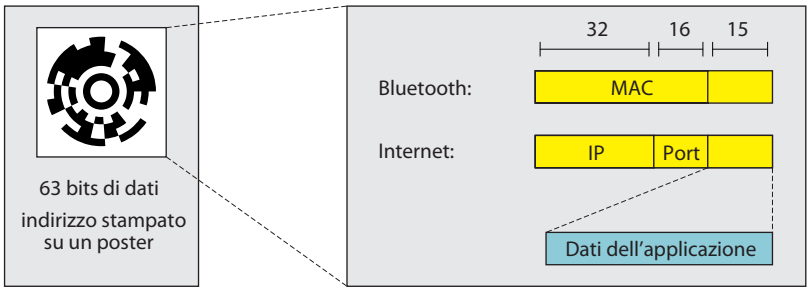


Figura 1.2: Visualtag contenente un indirizzo di rete

plicazione, programmata in modo opportuno, si possa sostituire al servizio che desideriamo veramente utilizzare.

La gestione di un indirizzo IP di Internet è invece troppo restrittiva poiché un server sulla rete potrebbe avere un indirizzo dinamico, per cui si utilizza normalmente un indirizzo simbolico. In questo caso il simbolo non sarebbe abbastanza capiente per codificare un'informazione di questo tipo.

Di seguito vediamo come sia possibile *"aumentare le funzionalità"* di un oggetto con dei casi d'uso dei *visualtag*.

1.2.1 Coda virtuale

Esistono molti scenari in cui è necessario gestire una coda. Consideriamo un utente che prende posto in una coda e riceve aggiornamenti sullo stato della sua posizione attraverso il proprio *smartphone*.

Nel nostro esempio ipotizziamo un ristorante pieno, dove i clienti devono attendere per un tavolo disponibile. Un poster affisso alla finestra descrive il sistema di coda virtuale con un marcatore *visualtag*. Dopo aver visto il poster, l'utente può catturare l'immagine dell'etichetta utilizzando il proprio telefono e stabilire così una connessione *bluetooth* con il servizio. Una volta stabilita la connessione viene visualizzato sul display un messaggio in cui

si indica il tempo di attesa e si chiederà all'utente se desidera entrare in coda. Successivamente il servizio chiederà, ad esempio, il numero di posti da prenotare e il numero di telefono dell'utente (questa informazione potrebbe essere recuperata automaticamente dalla rubrica del telefono).

Terminata la connessione il cliente è libero di allontanarsi dal ristorante visitando, ad esempio, gli altri negozi in zona. Un messaggio di testo (SMS) avviserà l'utente in anticipo, segnalando il progresso della coda e la disponibilità del tavolo prenotato, in tempo utile per tornare al ristorante e prendere posto.

Molti ristoranti utilizzano già un sistema in parte simile a quanto esposto: utilizzando un dispositivo hardware simile a un cercapersone che suona o si illumina quando il tavolo del cliente è pronto. In questi posti, generalmente, si assegna al cliente uno di questi dispositivi di notifica e lo si invita ad attendere al bar, poiché il funzionamento è relegato ad un'area circoscritta.

L'applicazione *visualtag* ha permesso di ottenere la medesima funzione e allo stesso tempo ha ridotto in modo significativo i costi per il proprietario del ristorante (che non deve più dotarsi di cercapersone), ha agevolato il cliente (non deve portarsi dietro alcun dispositivo se non il telefono che già possiede) e ha *aumentato le funzionalità* della locazione "ristorante" rispetto a quelle tradizionali.

1.2.2 Pubblicità interattiva

Vediamo come sia possibile integrare uno *smartphone* con un servizio specifico. Consideriamo un *touch screen* (chiosco elettronico) installato in un locale pubblico che permette ai clienti di sfogliare un catalogo di vacanze. Esistono già numerosi servizi che si basano su questo sistema, possiamo aumentare l'interattività e la facilità di utilizzo inserendo nel contesto anche il

telefono cellulare del cliente. Nel momento in cui l'utente vede una destinazione interessante può scattare una foto al *visualtag* che la contraddistingue, il servizio chiederà l'inserimento dei dati personali (indirizzo e-mail, indirizzo postale o numero di telefono) per consentire all'agente di viaggi di mettersi in contatto con lui inviando ad esempio una brochure per posta o contattandolo telefonicamente. I dati personali, normalmente, sono già presenti nella memoria del telefono (il cliente non deve perdere tempo digitando su piccole tastiere) pertanto queste informazioni vengono rapidamente trasferite al servizio elettronico.

1.2.3 Porta intelligente

Consideriamo un *visualtag* applicato sulla porta di un ufficio, la persona che stiamo cercando si è allontanata ed ha chiuso la sua stanza.

Un *visualtag* applicato sulla porta mi permetterà di contattare telefonicamente chi stavo cercando, oppure di inviargli un breve messaggio di testo (SMS), una e-mail con dei documenti allegati, ecc... La persona cercata potrebbe lasciare un avviso sulla porta, proprio come facciamo quando attacchiamo un *post-it* per avvertire qualcuno, con la peculiarità che il testo potrebbe cambiare dinamicamente a seconda di chi lo legge o essere modificato a distanza dall'autore. Anche in questo semplice esempio ci accorgiamo che l'oggetto "porta" vede aumentate le sue funzionalità: non è solo un serramento che chiude un ufficio ma fornisce anche un servizio informativo.

1.3 Ricerca dei servizi

In un articolo pubblicato sulle pagine di NTT Technical Review[6] si presenta una piattaforma che implementa un servizio di *discovery* e permette

di associare a oggetti nel mondo reale funzioni e servizi presenti in rete. Il sistema si basa sulle seguenti funzionalità:

- **funzione riconoscimento**: una funzione permette di distinguere gli oggetti reali con cui l'utente vuole interagire;

- **funzione scoperta**: una funzione scopre quali servizi di rete possono essere associati all'oggetto riconosciuto.

Il riconoscimento degli oggetti, come abbiamo visto, avviene tramite un *visualtag*. Le informazioni contenute nel marcatore non vengono elaborate dal telefono mobile con cui è stata catturata l'immagine, bensì da un sistema di elaborazione centralizzato, sicuramente molto più potente rispetto alle capacità computazionali e alle limitazioni di uno *smartphone*. Il risultato dell'elaborazione viene poi inviato all'utente (fig. 1.3). Questa soluzione ha il vantaggio di essere indipendente dall'architettura o dal sistema operativo del telefono, ma si tratta di un approccio poco scalabile per ovvi motivi: tutto il carico è relegato a un server che si occupa della decodifica dell'immagine e della ricerca dei servizi (operazioni dispendiose). Aumentando il numero di utilizzatori è necessario aumentare anche il numero di macchine dedicate a questo specifico compito. Un *visualtag* può essere decodificato utilizzando la capacità di elaborazione del processore di uno *smartphone*, che seppur poco potente, è sufficiente a portare a termine l'operazione in tempi brevi. Terminata l'elaborazione del *visualtag* (e riconosciuto il tipo di oggetto) il telefono sarà immediatamente in grado di collegarsi al servizio di rete richiesto.

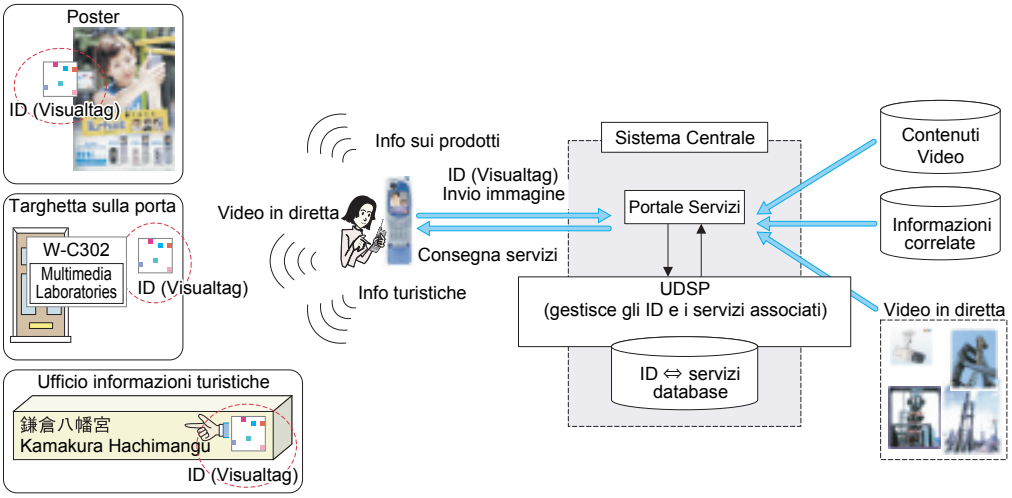


Figura 1.3: Ricerca dei servizi di rete associati a oggetti reali

1.4 Sistemi di riconoscimento

L'informazione può essere codificata attraverso simbologie stampate, questo avviene ad esempio con i codici a barre. Questa tecnica è stata studiata per venire incontro all'esigenza di leggere simboli con dispositivi elettronici in tempi rapidi. Esistono molte tipologie e la scelta è ampia: focalizzando l'attenzione sulle simbologie bidimensionali ci accorgiamo che queste sono in grado di ospitare una quantità di informazione superiore al normale codice a barre ed inoltre hanno il vantaggio di essere più robuste, permettendo di recuperare le informazioni anche nel caso una parte del simbolo risulti danneggiata o sbiadita.

Nella figura 1.4 vediamo alcuni esempi di simboli bidimensionali: MaxiCode (a), QR Code (b), PDF417 (c), Data Matrix (d).

MaxiCode MaxiCode è una simbologia bidimensionale di pubblico dominio creata nel 1992 dalla United Parcel Service adatta al tracciamento e gestione di spedizione pacchi. Utilizza punti disposti su una griglia esago-

(a) (b) (c) (d)

Figura 1.4: Simboli bidimensionali

nale. Il simbolo MaxiCode appare come un quadrato dei cerchi concentrici contornati da una serie di puntini esagonali. Può archiviare circa 93 caratteri e si possono concatenare insieme fino a 8 simboli per trasportare più informazioni. I cerchi concentrici servono al riconoscimento automatico del simbolo, senza tener conto del suo orientamento.

QR Code Anche QR Code è un codice a matrice bidimensionale, creato dalla società giapponese Denso Wave nel 1994. "QR" sta per "Quick Response" (replica veloce) poiché i creatori volevano che il codice permettesse una decodifica ad alta velocità. Questo tipo di codice è molto diffuso in Giappone e probabilmente è il sistema di simboli bidimensionali più popolare in quella nazione.

Inizialmente furono utilizzati per il tracciamento di parti meccaniche nella fabbricazione di veicoli. Adesso, i QR Code, sono utilizzati nella gestione di inventario di magazzino in molte industrie. Più recentemente, con l'introduzione di software atti alla decodifica dei QR Code tramite telefono cellulare, in Giappone si sono venute a creare molte nuove applicazioni orientate al consumatore, puntate a sollevare l'utente dalla tediosa azione di inserimento dati tramite le piccole tastiere dei telefonini. Un QR Code permette la memorizzazione di indirizzi, numeri di telefono o indirizzi Internet sempre piú

comuni nelle pubblicità su riviste giapponesi, in aggiunta i QR Code stampati su biglietti da visita sono diventati un sistema abbastanza comune per il trasferimento dei dati personali nella rubrica di un telefono cellulare.

Sfortunatamente le specifiche sono disponibili esclusivamente in Giapponese. Questa ‘e la ragione fondamentale per cui ‘e stata scartata questa simbologia.

PDF417 PDF417 è un codice bidimensionale utilizzato in molteplici applicazioni, compreso trasporti, gestione inventario e schede identificative. È pensato per tutti quei casi in cui le informazioni devono seguire l'articolo, infatti PDF sta per *Portable Data File*. Il formato PDF417 è stato sviluppato dalla Symbol Technologies ed è stato diffuso insieme a un codificatore e un decodificare entrambi *open source*.

Il codice PDF417 ha una grossa capacità di dati, fino a 2500 caratteri. Il codice PDF417 offre la possibilità di codificare anche i dati binari oltre ai caratteri alfanumerici o i caratteri della tabella ASCII, permettendo in tal modo di codificare foto o qualsiasi altro tipo di dato binario. Si tratta quindi di un sistema piuttosto potente. Tuttavia, per la scansione sono necessari dispositivi laser ad alta risoluzione, pertanto, le immagini catturate dalla fotocamera del telefono cellulare, non possono raggiungere la definizione necessaria richiesta per la decodifica.

Data Matrix Costituito da una serie di celle (quadrati) inserite in una griglia di righe e colonne, il Data Matrix è un codice a due dimensioni estremamente versatile che offre l'opportunità di inserire fino a 3116 caratteri in ogni simbolo.

I codici Data Matrix sono estremamente resistenti al danneggiamento grazie ad un algoritmo di correzione degli errori: anche se una porzione

dell'etichetta è danneggiata risulta comunque possibile recuperare i dati memorizzati. Sono quindi la soluzione ideale per il problema della tracciabilità della produzione nel mondo dell'industria.

Le specifiche sono disponibili presso AIM[1] e sono di pubblico dominio.

Si è scelto di utilizzare la simbologia Data Matrix per le sue caratteristiche di robustezza e semplicità di decodifica, pertanto verrà trattata meglio nella sezione 2.3.

[1] *Association for automatic Identification and Mobility*

Capitolo 2

Architettura

2.1 Analisi e specifica dei requisiti

Con *Ubiquitous Computing* intendiamo l'integrazione del processo informativo nell'ambiente che ci circonda, anziché avere computer come oggetti distinti. Termini come *pervasive computing*, *calm technology*, *things that think* e *everyware* esprimono questo concetto. I promotori di questa idea sperano che l'integrazione di processi informatici dentro gli oggetti di tutti i giorni possa portare le persone a interagire con questi in modo più naturale di come avvenga adesso, in qualsiasi luogo o circostanza ci si trovi.

Per ottenere un'ambiente di questo tipo servirebbe un'infrastruttura adatta a fornire un supporto a tutti i dispositivi: si tratta di una soluzione che non è presente ovunque e laddove non esista risulterebbe dispendioso crearne una.

Ci sono situazioni dove non è possibile dotare tutti gli oggetti in un ambiente di capacità informatiche, sia per limiti tecnici che per caratteristiche di contesto. Ad esempio in una zona a bassa densità di frequentazione dove un investimento di questo tipo risulterebbe antieconomico, o in aree rurali o

comunque lontane, dove potrebbe essere assente un'adeguata infrastruttura di telecomunicazione o di energia per alimentare gli apparati.

Per risolvere questo problema, possiamo supporre che ogni oggetto abbia la capacità di creare un'interfaccia adatta alla fruizione delle sue funzioni. Possiamo facilmente immaginare di affidare questa funzionalità ad un dispositivo comune per tutti gli oggetti, che l'utente ha sempre a portata di mano. In questo caso si avrebbero possibilità progettuali più ampie.

Sulla base di queste considerazioni possiamo pensare di sfruttare le sempre più numerose funzionalità dei dispositivi mobili, soffermandoci sui telefoni cellulari più evoluti (detti anche *smartphone*). Questi dispositivi sono di gran lunga i più diffusi, gran parte della popolazione possiede un telefono cellulare quindi è una buona approssimazione associare a ciascun individuo le funzionalità di questi apparati. L'infrastruttura di rete dei gestori telefonici è ben sviluppata e raggiunge anche le zone meno frequentate, inoltre i telefoni cellulari hanno una fonte di alimentazione propria (la batteria) in grado di garantire buona autonomia.

Vediamo come concretizzare le osservazioni appena fatte con una situazione reale di tutti i giorni, prendiamo in considerazione un'*area di sosta*: si tratta di un'entità che normalmente non è integrata con servizi informatici, sebbene altamente frequentata. L'interazione con un dispositivo mobile come uno smartphone potrebbe "aumentare le funzionalità" dell'area e rendere la sua fruizione più semplice e vantaggiosa per l'utente.

2.1.1 Aree di sosta regolamentate

Le aree di sosta di tutte le città sono ormai regolamentate da tariffe a tempo o periodi massimi di permanenza. Le amministrazioni contano in questo modo di incentivare gli utenti della strada a utilizzare mezzi pubblici

e ridurre così traffico e inquinamento ed allo stesso tempo si garantiscono una buona fonte economica.

Le aree di sosta hanno tariffe differenti a seconda della loro ubicazione e nella maggioranza dei casi il pagamento avviene tramite macchinette automatiche. Queste accettano soltanto monete e stampano un biglietto da posizionare all'interno del veicolo, in modo da essere letto da un addetto al controllo. L'automobilista che desidera parcheggiare il proprio veicolo, normalmente, si deve trovare nella condizione di avere monete sufficienti per pagare la sosta. Deve anche determinare in anticipo per quanto tempo vuole lasciare il mezzo in permanenza. Nell'esperienza di tutti i giorni ci accorgiamo che nell'incertezza e non volendo rischiare di andare incontro a una sanzione pecuniaria salata, paghiamo molti più minuti di quanti ne usufruiamo.

In collaborazione con alcune compagnie di telefonia mobile, sono stati sviluppati sistemi di pagamento tramite telefono cellulare. Vodafone ha realizzato un sistema denominato *M-Pay Park* basato su brevi messaggi di testo (SMS), adottato in alcune città europee e italiane (recentemente anche a Pisa).

L'utente che vuole usufruire di questo servizio deve prima crearsi un *"portafoglio elettronico"* da abbinare al proprio telefono cellulare. Da questo verrano scalati tutti i costi relativi alle soste, pertanto è necessario prima "ricaricarlo", utilizzando le tessere del gestore di telefonia.

Successivamente è necessario richiedere al gestore delle aree di sosta un contrassegno. Questo riporta un codice a barre che viene verificato dai controllori del pedaggio, pertanto deve essere posizionato in modo ben visibile, tale da permetterne la scansione. Di volta in volta che viene lasciato il veicolo in sosta, l'automobilista deve segnalare il suo arrivo inviando un SMS, compilato con la giusta sintassi. Il messaggio deve contenere un comando

seguito dal codice identificativo dell'area di sosta e dall'importo previsto per la permanenza.

Questo sistema ha il vantaggio di eliminare il problema di dover utilizzare i parchimetri a monete, infatti il parchimetro diventa di fatto "virtuale", ma si può notare che ci sono ancora diversi svantaggi, che rendono il servizio poco agevole:

- la creazione di un *portafoglio elettronico* ha dei costi aggiuntivi, infatti il gestore applica un balzello sulle ricariche, dato che sono le stesse tessere utilizzate per acquistare credito telefonico;
- è necessario recarsi presso un rivenditore o presso un ufficio per farsi rilasciare il contrassegno da posizionare dentro la propria auto;
- è necessario scrivere comandi testuali con la giusta sintassi (da inviare via SMS), l'utente deve ricordarseli per fruire del servizio e deve stare attento a non commettere errori durante la digitazione;
- il proprio operatore di telefonia prevede sempre un costo per l'invio di SMS, da aggiungere alla tariffa per la sosta: senza considerare i messaggi inviati a vuoto per eventuali errori, questo sistema fa lievitare le spese anche del 30%;
- il tempo di pagamento è troppo alto rispetto al metodo classico (fino a 2,5 volte superiore).

Un'indagine condotta da Hiser Group nella città di Melbourne, dove i guidatori possono pagare il parcheggio tramite messaggio oppure tramite moneta, ha rivelato che per il 20% degli intervistati, il pagamento via cellulare non è andato a buon fine, creado insoddisfazione per il servizio. Per ovviare

a questo problema, in Italia è disponibile anche un risponditore automatico, ma rende i tempi di pagamento ancora più lunghi.

2.1.2 Requisiti funzionali del sistema

Consideriamo l'interazione fra l'utente e l'area di sosta. Ogni volta che si parcheggia il proprio veicolo, l'azione si suddivide in due fasi: l'identificazione visiva dall'area e l'interazione vera a propria. Nel contesto in cui l'azione è mediata da un dispositivo mobile si può ricalcare la stessa sequenza di azioni. Raffinando l'analisi possiamo dividere la sequenza in più passi:

- l'utente riconosce l'area di sosta;
- l'utente agisce sul dispositivo mobile;
- il dispositivo mobile riconosce l'area di sosta;
- il dispositivo mobile media l'interazione tra utente e parcheggio.

In questo modo l'area di sosta *acquista funzionalità* offerte all'utente tramite il proprio telefono cellulare. Il sistema deve quindi permetterne il riconoscimento preciso e offrire un metodo di interazione con l'utente stesso.

2.1.3 Requisiti di sistema

Requisiti software

Il modello proposto utilizza come piattaforma di interazione il telefono cellulare, pertanto il riconoscimento delle aree di sosta, l'interfaccia utente e la gestione delle funzionalità dovranno funzionare su questo tipo di dispositivo, tenendo conto delle differenze dei vari modelli in commercio.

Requisiti dei dispositivi mobili

Il dispositivo mobile dovrà essere in grado di riconoscere visivamente l'area di sosta, pertanto dovrà essere in grado di catturare immagini e avere sufficiente capacità di calcolo, per portare a termine le elaborazioni in tempo adeguato. Per lo scambio dei dati è necessaria la capacità di connettersi a Internet in modalità wireless.

Requisiti infrastruttura di comunicazione

Le interfacce di comunicazione devono essere realizzate in modo da essere accessibili almeno nel raggio visivo dell'automobilista, dovranno anche essere associate alla specifica area di sosta. Dal momento che l'identificazione avviene in maniera visuale, è necessario che questa sia possibile nei pressi dell'area di sosta.

2.2 Descrizione concettuale del sistema

L'applicazione permette, da lato utente, di semplificare la procedura di accesso e pagamento ad aree di sosta. Per il gestore, il sistema garantir'a una semplificazione delle operazioni di verifica dei veicoli autorizzati.

L'automobilista potrà utilizzare il proprio telefono cellulare per segnalare il suo arrivo, verranno così gestiti anche eventuali addebiti dove è prevista una tariffa a tempo. In questo modo si evita la necessità di avere a disposizione monete o di dover decidere in anticipo la durata della sosta (se questa non è regolata da un limite massimo). Il riconoscimento del parcheggio avviene in modo intuitivo, scattando una foto a un *visualtag*, stampato su cartelli disposti all'interno dell'area di sosta. Non vi è nessuna necessità di esporre biglietti o contrassegni all'interno del veicolo, in quanto la verifica dei mezzi

autorizzati avviene tramite la ricerca del numero di targa. L'utente dovrà quindi compiere solo due azioni:

- all'arrivo, segnalare l'area di sosta in cui è posteggiato il veicolo, scattando una foto al *visualtag* che la identifica;
- all'uscita, segnalare l'abbandono del parcheggio tramite l'applicazione sul telefono cellulare.

A seconda delle politiche del gestore è possibile un meccanismo di pagamento prepagato o postpagato.

La verifica delle vetture cui è concessa la sosta avviene utilizzando un'altra applicazione, in esecuzione sui dispositivi mobili degli addetti al controllo: sarà sufficiente verificare il numero di targa del veicolo per sapere se un mezzo è ammesso oppure no.

2.2.1 Scelte tecnologiche

Piattaforma software

L'applicazione deve essere eseguita su un telefono cellulare. Osservando il panorama dei dispositivi in circolazione ci rendiamo conto dell'esistenza di sistemi operativi e tecnologie diverse. Dato che si vuole rendere l'applicazione disponibile alla maggioranza degli utenti di telefoni cellulari, sarebbe un errore scegliere un determinato sistema operativo o architettura usata da uno specifico produttore. Volendosi orientare verso standard ampiamente diffusi ci accorgiamo che *Symbian OS* e *Java 2 Micro Edition* (J2ME) sono le piattaforme più comuni.

Symbian OS è l'erede del sistema operativo EPOC, creato dalla Psion e gira esclusivamente su processori ARM, attualmente è detenuto da alcune

grandi aziende che operano anche nel capo della telefonia mobile. Si tratta di un sistema operativo progettato per dispositivi mobili, quindi in grado di sfruttare al meglio le ridotte capacità di elaborazione di questa tipologia di sistemi. Non è *open source* poiché il codice sorgente non è disponibile pubblicamente. Tuttavia, quasi tutto il codice è fornito ai produttori di dispositivi mobili basati su Symbian e ad altri operatori. Inoltre la documentazione relativa alle API è disponibile pubblicamente, dunque chiunque ha la possibilità di sviluppare software per Symbian. Questo è in contrasto con altri sistemi operativi per terminali mobili che generalmente accettano soltanto applicazioni Java.

Lo svantaggio di Symbian st'a nel fatto che funziona solo su processori ARM della serie V5t e 9. Pur essendo dei chip molto comuni, non sono adottati da tutti i produttori, quindi attualmente Symbian non 'e la scelta che permette di raggiungere il maggior numero possibile di utenti.

Java Micro Edition (noto anche come Java ME o J2ME) è l'ambiente Java adattato ai dispositivi a risorse limitate come PDA, telefoni cellulari e simili. È la tecnologia più diffusa per lo sviluppo di giochi e utilities per i cellulari. Come le altre edizioni di Java, J2ME è una piattaforma portabile, il suo funzionamento può essere emulato con un personal computer, cosa che semplifica l'attività di sviluppo e di testing.

J2ME può essere utilizzato per sviluppare applicazioni per una ampia gamma di apparati. Diverse tipologie di apparati sono identificate da diversi profili a loro volta riferiti a diverse configurazioni. La configurazione *Connected Limited Device Configuration* (CLDC), per esempio, include un sottoinsieme minimo di classi Java, ed è utilizzata su dispositivi con scarsissime capacità di calcolo. Fra i profili che operano in configurazione CLDC

compare il *Mobile Information Device Profile* (MIDP), pensato per i cellulari. Il MIDP prevede un sistema di GUI orientato a display a cristalli liquidi e una API di base per giochi in 2D. Molti cellulari moderni vengono forniti con un'implementazione residente dell'MIDP. Un altro profilo che utilizza la configurazione CLDC è l'*Information Module Profile* (IMP), usato per esempio in distributori automatici e altri apparati dotati di funzioni minime di display e di connettività di rete.

Java fa ricorso all'emulazione di una macchina virtuale: la *Java Virtual Machine* (JVM): in questo modo si ha il vantaggio di poter eseguire l'applicazione su qualsiasi architettura sia disponibile un interprete JVM, di contro questa soluzione spreca risorse e di conseguenza rallenta l'esecuzione. Il vantaggio cardine, però, sta nel fatto che J2ME è diffuso nella quasi totalità dei dispositivi mobili ad oggi in circolazione (anche Symbian OS supporta Java). Un altro vantaggio viene dal fatto che è disponibile un framework per l'utilizzo delle capacità multimediali del dispositivo, le Mobile Media API (MMAPI), questo sarà utile per sfruttare la funzionalità di cattura delle immagini attraverso la fotocamera incorporata nel telefono.

Dispositivi mobili

Uno degli obbiettivi è quello di allargare al massimo la compatibilità del sistema a tutti i dispositivi mobili in circolazione. Tuttavia *le MMAPI (JSR-135) sono un pacchetto opzionale*, quindi è necessario che il produttore del dispositivo mobile abbia implementato questa funzionalità. Questo requisito è facilmente verificabile dalle informazioni che si trovano sui siti delle aziende produttrici, per ciascun modello. I problemi non sono comunque del tutto risolti: anche se MMAPI è implementata, la specifica non pone alcun requisito su certe capacità. Ad esempio nel caso della cattura di immagini

non viene detto a quale risoluzione debbano essere catturate; questo fa si che molto spesso i cellulari vengano forniti con questa funzionalità limitata (basse risoluzioni, poco utili per elaborazioni), o addirittura assente.

Durante l'elaborazione dei simboli si fa ricorso all'*aritmetica in virgola mobile*, questa funzionalità è supportata a partire dalla versione CLDC 1.1[8] ed è pertanto necessario che il dispositivo la supporti.

Sistema di memorizzazione

Il sistema deve permettere la memorizzazione e gestione dei dati dei clienti. Per rendere possibile l'accesso e l'elaborazione di queste informazioni anche da parte di altri sistemi è conveniente utilizzare un *Database Management System*.

MySQL è un Database Management System (DBMS) relazionale, composto da un client con interfaccia a caratteri e un server, entrambi disponibili sia per sistemi Unix che per Windows, anche se prevale un suo utilizzo in ambito Unix. Il codice di MySQL è di proprietà della omonima società, viene però distribuito con la licenza GNU GPL oltre che con una licenza commerciale. Una buona parte del codice del client è licenziato con la GNU LGPL e può dunque essere utilizzato per applicazioni commerciali. Nel tempo è diventato il più famoso database open source grazie anche alle sue performance velocistiche e di affidabilità.

Si è scelto quindi MySQL come componente per la memorizzazione e gestione dei dati: la sue caratteristiche lo rendono più che adeguato ai nostri scopi e i suoi termini di licenza permettono di utilizzarlo senza problemi.

2.3 Standard di codifica Data Matrix

Data Matrix è uno standard di codifica bidimensionale, basato su matrici costituite da piccoli quadrati bianchi o neri disposti su una struttura quadrata o rettangolare (fig. 2.1 a). Le informazioni codificate possono essere testo o dati grezzi. Normalmente la dimensione varia da pochi byte fino 2 kilobytes e la lunghezza dei dati codificati dipende dalla dimensione dei simboli utilizzati.

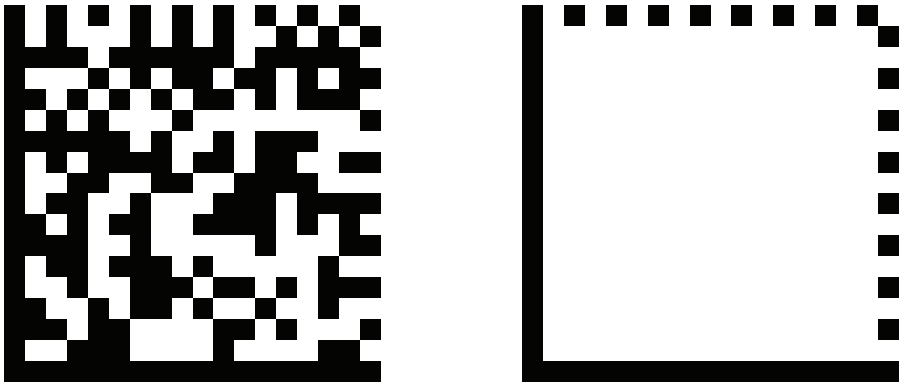
I simboli Data Matrix sono costituiti da piccoli *moduli* (quadretti che rappresentano i bit dell'informazione codificata). A seconda della situazione un modulo bianco è uno `0` e un modulo nero un `1`, o viceversa. Ogni Data Matrix è costituito anche da strutture utilizzate dal meccanismo di elaborazione: *handles* e *syncs*. Gli *handles* sono due linee perpendicolari dello stesso colore dei moduli rappresentanti `1` e vengono utilizzati per ottenere l'allineamento del simbolo, mentre i *syncs* sono simili agli *handles* con la differenza che sono fatti da moduli bianchi e neri alternati e servono a individuare la griglia di posizionamento (fig. 2.1 b). Se le dimensioni della matrice aumentano (superando 26×26 moduli, *handles* e *syncs* compresi) vengono aggiunti fino ad altri 8 *handles* e *syncs* sia orizzontalmente che verticalmente (fig. 2.2). La dimensione dei simboli varia da 8×8 fino a 144×144.

La codifica avviene in due fasi:

1. *high level encoding*: i dati vengono convertiti in *codeword* da 8 bit;
2. *low level encoding*: i *codeword* (di seguito CW) vengono convertiti in quadretti bianchi e neri (*moduli*).

I simboli Data Matrix supportano ECC200 secondo le specifiche ANSI/AIM BC11 e ISO/IEC 16022. ECC200 è la specifica per la correzione degli errori basata sull'algoritmo Reed-Solomon[1] (si pensi ad un'etichetta

[1]Il Codice Reed-Solomon è impiegato per correggere errori di flusso in svariate applica-



(a) (b)

Figura 2.1: Simbolo Data Matrix ed evidenza *handles* e *syncs*

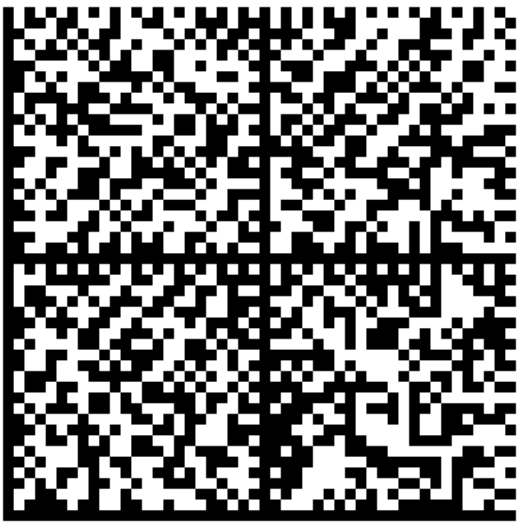


Figura 2.2: Simbolo Data Matrix composto da 4 sottosimboli

stropicciata o sbiadita), questo permette il riconoscimento di simboli danneggiati anche del 60%. In ciascun simbolo è possibile codificare informazioni di varia natura:

- fino a 2335 caratteri alfanumerici;
- fino a 1556 byte;
- fino a 3116 caratteri numerici.

Lo standard è molto flessibile e permette di usare la codifica più opportuna a seconda dell'applicazione. Le codifiche permesse da Data Matrix ECC200 prevedono 6 modalità di "compattamento" (tab. 2.1):

- **ASCII**: questo schema codifica caratteri ASCII in tre modi
  - per i caratteri ASCII da 0 a 127, CW vale <codice ASCII> + 1;
  - per i caratteri ASCII esteso da 128 a 255, un primo CW con valore 235 e un secondo CW con valore <codice ASCII> - 127
  - per coppie di cifre 00, 01,... 99, CW assume il valore della coppia di cifre + 130
- **C40**: caratteri alfanumerici maiuscoli;
- **TEXT**: caratteri alfanumerici minuscoli;
- **X12**: set ANSI X12;
- **EDIFACT**: modalità che permette di compattare 4 caratteri in 3 CW, ogni carattere è codificato su 6 bit e appartiene ai valori ASCII da 32 a 94;

zioni di comunicazione digitale e memorizzazione di dati. Attualmente viene utilizzato in diversi sistemi quali: supporti di memorizzazione (CD, DVD), dispositivi mobili e wireless, comunicazioni satellitari, connessioni a banda larga xDSL.

Modalità	Dati da codificare	Tasso di compattazione
ASCII	Caratteri ASCII da 0 a 127	1 byte per CW
ASCII esteso	Caratteri ASCII da 128 a 255	0.5 byte per CW
ASCII numerico	Cifre ASCII	2 byte per CW
C40	Caratteri alfanumerici maiuscoli	1.5 byte per CW
TEXT	Caratteri alfanumerici minuscoli	1.5 byte per CW
X12	ANSI X12	1.5 byte per CW
EDIFACT	Caratteri ASCII da 32 a 94	1.33 byte per CW
BASE256	Caratteri ASCII da 0 a 256	1 byte per CW

Tabella 2.1: Modalità di codifica Data Matrix

- **BASE256**: permette di codificare qualsiasi byte. Si ottiene tramite il CW 231 seguito da un campo che contiene il numero di byte che seguono, e dallo *stream* binario. Quest'ultimo deve passare attraverso un processo di "randomizing" descritto nelle specifiche[10].

Ai dati codificati viene applicato un codice a correzione di errore Reed-Solomon. Il numero dei CW di correzione dipende dalla grandezza della matrice, più esattamente dalla grandezza del blocco (vedi appendice D). Il codice Reed-Solomon è basato su un'equazione polinomiale dove la potenza di x è il numero di CW utilizzati. Ad esempio per una matrice 8×8, che prevede 5 CW di correzione, si ottiene un'equazione del tipo $x^5+ax^4+bx^3+cx^2+dx+e$ dove a, b, c, d ed e sono i fattori del polinomio.

Ottenuta la stringa di byte, si dispongono i moduli corrispondenti secondo un algoritmo di piazzamento che segue un'andatura a zig-zag[11], il simbolo così creato può essere stampato su una superficie (anche in negativo) o inciso. Si possono trovare Data Matrix incisi su parti meccaniche o schede elettroniche.

2.4 Architettura software

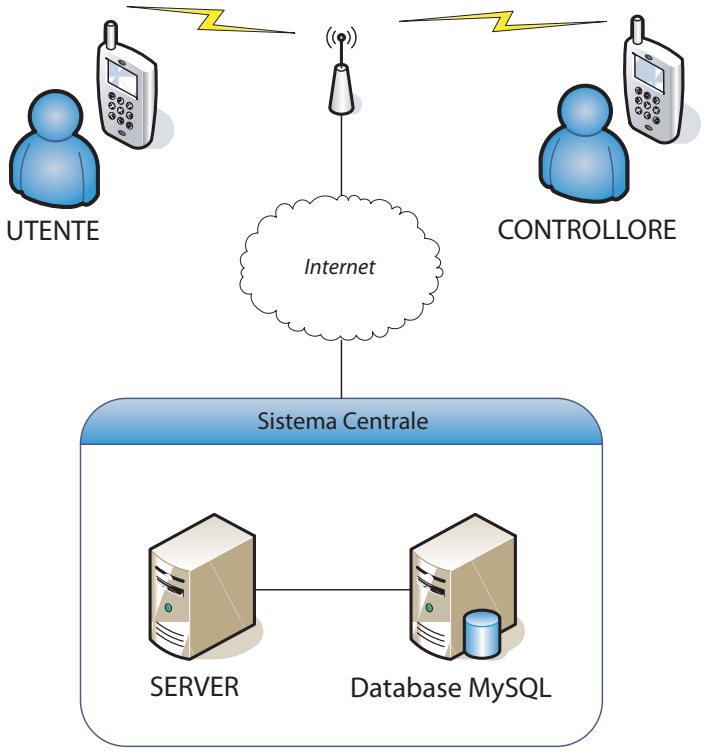


Figura 2.3: Blocchi funzionali del sistema

In figura 2.3 sono rappresentati i blocchi funzionali del sistema. Il funzionamento del sistema è gestito dal blocco server con cui colloquiano i terminali in ambiente mobile, tramite connessione Internet wireless. Il collegamento potrà avvenire in modalità GPRS o UMTS, inoltre stanno comparendo sul mercato telefoni con integrate funzionalità Wi-Fi. I dati utili al funzionamento del sistema, relativi a clienti ed aree di sosta, vengono archiviate in un database MySQL.

I blocchi controllore e utente rappresentano, rispettivamente, le applicazioni in esecuzione sui telefoni mobili degli addetti al controllo dei pedaggi e degli automobilisti.

2.4.1 Applicazione utente

L'applicazione, caricata sul telefono cellulare dell'utente, permette di segnalare l'arrivo del suo veicolo in un area di sosta, e il successivo abbandono.

Al primo utilizzo è permessa solo la registrazione di un nuovo utente o in alternativa il ripristino di una precedente registrazione. Durante la fase di registrazione l'utente viene invitato a inserire i propri dati seguendo una procedura guidata. In questa fase verrà richiesta anche la targa del veicolo utilizzato, il metodo di pagamento preferito e una password personale. Nel caso i dati utente siano già memorizzati nel sistema (a seguito di una precedente registrazione) sarà possibile il recupero delle informazioni identificandosi con la targa del proprio veicolo e la password personale.

Prima di autorizzare una sosta è necessario inserire la password personale, questa misura di sicurezza è stata considerata per impedirne l'uso da estranei. Si presume che il telefono sia uno strumento personale, pertanto l'utente potrà anche disattivare questa funzione.

Il sistema gestisce un meccanismo di pagamento prepagato. L'utente potrà verificare il proprio credito e effettuare "ricariche", in modo analogo ad altri sistemi.

L'identificazione delle aree di sosta avviene attraverso *visualtag*, l'applicazione si occupa anche di elaborare l'immagine acquisita dalla fotocamera e alla decodifica del marcatore. In modo analogo, anche per il meccanismo di credito prepagato si è scelto di utilizzare un *visualtag* applicato sulle ricariche in modo da semplificare l'azione di inserimento.

2.4.2 Applicazione controllore

Gli addetti al controllo dei parcheggi avranno a disposizione un dispositivo che permetterà di verificare a quali auto è concessa la sosta. L'applicazione di controllo, essendo realizzata in J2ME, potrà funzionare su telefoni cellulari o palmari.

Il riconoscimento dell'area da verificare avviene sempre tramite i *visualtag* disposti nel parcheggio. Successivamente viene interrogato il *sistema centrale* per ottenere un elenco di targhe, relativo ai veicoli autorizzati. L'addetto al controllo potrà consultare questo elenco filtrando i dati con un semplice strumento di ricerca: l'individuazione di un veicolo avviene digitando la sua targa o parte di essa.

È previsto un meccanismo di aggiornamento, che permette di mantenere sincronizzati gli elenchi in memoria sul dispositivo mobile con quelli nel *sistema centrale*, senza la necessità di ricaricare tutti i dati ogni volta.

2.4.3 Sistema centrale

Il sistema centrale è costituito da un'applicazione server e un database. Le applicazioni sui telefoni cellulari colloquiano con il server, attraverso una connessione Internet, e il suo compito è quello di gestire le autorizzazioni dei vari utenti (segnalano arrivo/abbandono dalle aree di sosta) e gli elenchi dei veicolo ammessi (consultati dai controllori).

Vengono verificati in modo periodico quali veicoli sono ammessi nelle aree di sosta regolate da tempo massimo di permanenza, e vengono "espulsi" quelli che superano tale limite.

I costi delle soste vengono calcolati sulla base dell'effettivo tempo di permanenza e secondo le regole definite per ogni area di sosta. Vengono conside-

rati gli orari di arrivo e abbandono dei veicoli, registrati dall'orologio interno del server.

Viene gestito anche il credito prepagato di ogni utente, da cui vengono scalati gli importi delle soste. La memorizzazione dei dati e la gestione delle transazioni è affidata a un database server MySQL.

Capitolo 3

Realizzazione

L'applicazione realizzata mostra come sia possibile concepire un sistema di pagamento delle aree di sosta tramite telefono cellulare. Questo è molto diverso dai sistemi già presenti, che si affidano normalmente all'invio di SMS. L'utente non dovrà compilare alcun messaggio con comandi precisi, perché sarà guidato da un'interfaccia grafica semplice da usare.

Sempre tramite telefono, sarà possibile la verifica delle aree di sosta da parte del personale di controllo.

L'applicazione si divide in tre entità:

- un **sistema centrale** comprendente un *server* (a cui fanno capo le applicazioni installate sui telefoni degli utenti e dei controllori) e un *database*;
- **applicazione utente**, utilizzata dagli automobilisti;
- **applicazione controllore**, utilizzata dagli addetti al controllo delle aree di sosta.

Questo sistema sfrutterà i *visualtag* per il riconoscimento delle aree di

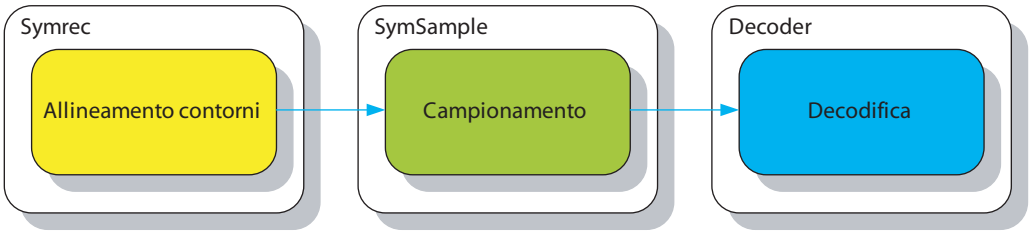


Figura 3.1: Algoritmo di decodifica in blocchi di alto livello e classi

sosta, riducendo a poche operazioni l'intervento da parte dell'utente ed evitando così che possa commettere errori.

Un decodificatore di simboli Data Matrix era già stato sviluppato presso il dipartimento di Ingegneria dell'Informazione nel corso di precedenti studi[4], questo viene utilizzato anche nel package tagReader.

3.1 Package tagReader

Il package tagReader è responsabile dell'estrazione e della decodifica dei simboli Data Matrix. Le classi contenute nel package sono numerose: classi per la manipolazione geometrica (*line*, *segment*, etc...), classi per il trattamento di bitmap (*byteImageBW*, *shortImageBW*), classi specifiche per l'image processing (*symRec*, *symSample*, *decoder*) ed altre. Di queste l'unica pubblica e visibile all'esterno è symRec. L'uso è molto semplice: si inizializza il costruttore con la bitmap che contiene la foto, più alcuni parametri e si chiama il metodo go(), che fa partire l'elaborazione. Il risultato ritornato da go() è una stringa che contiene l'informazione decodificata. Se la decodifica non è andata a buon fine verranno sollevate alcune eccezioni del tipo tagDecoderException contenenti un messaggio che indica il problema incontrato.

In figura 3.1 vediamo le tre fasi principali dell'algoritmo di decodifica.

(a) (b) (c)

Figura 3.2: Immagine true color, in scala di grigi e con soglia adattiva

Nella fase di *allineamento ai contorni* del simbolo, in cui si crea un poligono che coincida il più possibile con i bordi del simbolo nell'immagine. Segue una fase di *campionamento* in cui vengono campionati i moduli che costituiscono il simbolo, da cui si ottiene una sequenza di byte che verrà poi decodificata nella fase di *decodifica*.

Allineamento ai contorni

Il primo passo per poter decodificare il simbolo è "agganciarlo" con precisione all'interno dell'immagine che si è ottenuta dalla fotocamera. Partiamo dal presupposto di aver a disposizione un'immagine *true color*, vale a dire un'immagine in cui ogni pixel è descritto da tre byte: uno per ogni componente di colore primario secondo il modello RGB (rosso, verde, blu). Questa configurazione permette di rappresentare circa 16 milioni di colori (figura 3.2 a). A partire da questa immagine sarà necessario eseguire un certo numero di operazioni.

Conversione in scala di grigi

L'informazione di colore non è di nessuna utilità. Quindi si passa a una rappresentazione in scala di grigi (1 byte per pixel, con 256 livelli di grigio). Oltre che un passo necessario a proseguire le elaborazioni, questo permette di ridurre l'occupazione di memoria di 2/3 (figura 3.2 b).

Soglia adattiva

Prima di poter procedere all'etichettatura delle componenti connesse dell'immagine è necessario applicare l'operatore 'soglia'. Questo consiste nello stabilire un valore (compreso tra 0 e 255) e impostare come *bianchi* tutti i punti che stanno sopra questa soglia e *neri* quelli che stanno sotto. Durante questo processo si potrebbero perdere troppi dettagli, un'immagine in ingresso molto scura potrebbe dare come risultato un'immagine in uscita completamente nera, o viceversa per un'immagine piuttosto chiara.

Si rende opportuno utilizzare una *soglia adattiva* che operi nel seguente modo: viene scandita l'intera immagine alla ricerca dei valori di luminosità selezionando il massimo e il minimo, successivamente viene calcolato il valor medio tra i due e si utilizza questo risultato come valore di soglia. Questa soluzione permette di operare correttamente sia su immagini in ingresso molto scure o molto chiare (figura 3.2 c).

Etichettamento componenti connesse

Dopo aver applicato la soglia otteniamo un'immagine composta da aree nere o bianche. Essendo il simbolo nero e stampato su uno sfondo bianco, le aree nere faranno parte del simbolo. Tramite l'algoritmo *Connected Components Labeling*, nella variante *riga-per-riga* e usando la struttura *Union Find* per risolvere le dipendenze[7], si identificano tutti quei gruppi di punti dello

stesso colore che sono direttamente a contatto. Ciascuno di questi insiemi si chiama *componente connessa* ed è identificato con un numero. Il risultato di questo algoritmo è un buffer di memoria che rappresenta l'immagine, dove per ogni pixel, invece del colore, è indicata l'area di appartenenza. Questo buffer si chiama "mappa delle aree" (figura 3.3), da qui è necessario scegliere l'area che conterrà le cosidette *handles*, ovvero due righe nere, dello spessore di un modulo, perpendicolari fra loro. Sicuramente questi si troveranno nella stessa area, in quanto connesse, ma non è detto che questa area connessa rappresenti l'intero simbolo, anzi nella maggior parte dei casi non è così poiché risulta suddiviso in varie sezioni. È comunque possibile identificare la regione che contiene gli *handles* grazie ad alcune informazioni statistiche[12]:

- non tocca i bordi dell'immagine (si suppone che la foto sia stata scattata bene);
- è contenuta nell'area bianca di maggiore estensione;
- è nera;
- è l'area con un perimetro più lungo delle altre.

Usando questi criteri nell'ordine indicato, si riesce sempre ad identificare la porzione di simbolo che contiene gli *handles*. Queste operazioni sono suddivise nelle seguenti fasi:

Estrazione caratteristiche: vengono estratte varie caratteristiche necessarie alla scelta in base ai criteri statistici. Si crea una tabella che indichi la relazione di inclusione fra le varie aree e si calcola il perimetro di ciascuna area: per eseguire questa operazione è necessario seguire il contorno fra le aree di colore diverso, usando una versione riadatta-

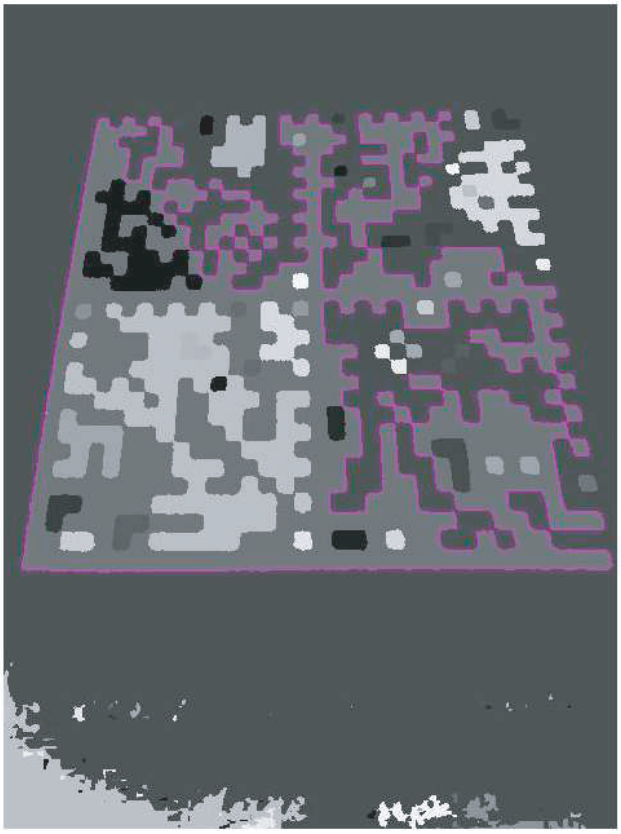


Figura 3.3: Mappa delle componenti connesse, in viola il contorno di un'area. Ogni gradazione di grigio rappresenta un'etichetta diversa

ta dell'algoritmo *Neighbor tracing* di Moore[5] (figura 3.3). Infine si calcola l'area di ciascuna componente connessa.

Scelta del miglior candidato: si applicano nell'ordine i criteri sopra elencati. Dapprima si seleziona l'area bianca di maggiore estensione, che rappresenta il piano su cui è stampato il simbolo; successivamente fra tutte le aree nere in essa contenute si sceglie quella con il perimetro maggiore.

La parte del simbolo che contiene gli *handles* è stata riconosciuta. Quello che ora interessa è ricavare un poligono che incornici il simbolo in modo preciso. Si procede secondo i seguenti passi:

Segmentazione dei contorni: è necessario identificare con precisione gli *handles* (i segmenti più lunghi del perimetro dell'area). Il contorno ricavato tramite l'algoritmo di Moore è solo un insieme di punti consecutivi e questo rende necessario applicare un'approssimazione al set di punti in modo da suddividerlo in segmenti rettilinei. Per ottenere questo risultato si sfrutta un algoritmo di segmentazione[4].

Scelta dei segmenti più lunghi: uno dei risultati del passo precedente è anche la lunghezza dei segmenti. Basta scegliere i due segmenti più lunghi e si ottengono i due *handles*.

Allineamento di precisione: quello che si vuole sono due segmenti che seguano il bordo degli *handles* nel modo più fedele possibile. Per arrivare a questo risultato si prendono come riferimento gli estremi di ciascun segmento e da questi si considerano i punti fra i due estremi ottenuti con l'algoritmo di Moore. Si calcola la retta di interpolazione fra tutti i punti, si ottiene così l'approssimazione migliore del bordo.

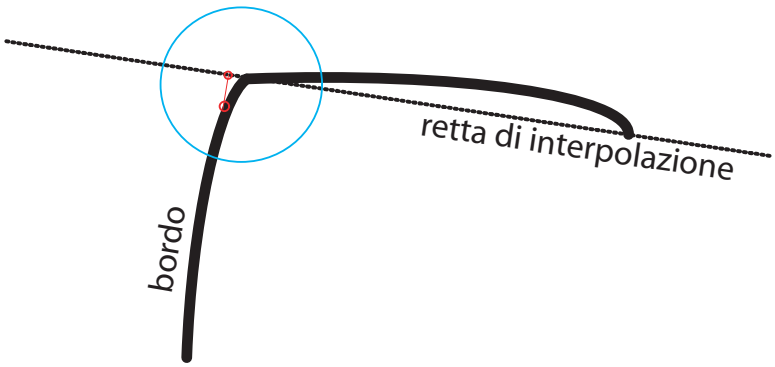


Figura 3.4: Approssimazione degli estremi degli handles

Può capitare che sia presente un effetto "occhio di pesce": una deformazione dell'immagine che la fa apparire come se fosse adagiata su una grossa sfera. Questa aberrazione ottica è dovuta alle piccole dimensioni degli obiettivi delle fotocamere dei cellulari ed è presente sopratutto nel caso di foto molto ravvicinate. Nonostante il fenomeno sia in genere di lieve entità, può capitare che sia sufficiente a frammentare gli *handles* e a non coprirli con un segmento solo. Visto che nella maggior parte dei casi si riesce comunque ad identificare i segmenti che si trovano su ciascuno *handle*, è necessario un ulteriore passo per determinarne con precisione gli estremi. Si calcola una retta di interpolazione, se scorrendo i bordi questi risultano entro una certa distanza, le due proiezioni più distanti rappresenteranno gli estremi dell'*handle* (figura 3.4). Questo procedimento ha senso in quanto è noto che il simbolo è quadrato.

Allineamento al pattern alternato: è necessario posizionare due segmenti che costeggino perfettamente il pattern alternato. In questo caso non si può ricorrere alle tecniche usate per gli *handles*, visto che abbiamo a che fare con delle "linee immaginarie" che non ci sono nella realtà.

Si usa un approccio totalmente diverso: di questi segmenti sappiamo il punto di partenza, individuato con il passo precedente, basta calcolarne la direzione, i punti finali saranno dati dall'intersezione dei loro prolungamenti. Si considera il punto finale di uno degli handles e si traccia una retta con ancolo α: tracciando rette con angoli diversi si arriva a determinare la retta coincidente al pattern alternato (fig. 3.5).

Con questo abbiamo ottenuto un poligono che incornicia molto fedelmente il simbolo.

Campionamento

Quando la posizione del simbolo nell'immagine è nota, è necessario determinare il numero e la dimensione dei moduli, i loro centri e successivamente campionare nei punti centrali di ciascun modulo. La classe che si occupa di queste operazioni è SymSample. Vediamo le fasi principali di questa operazione:

Allineamento al pattern: per poter creare la griglia di campionamento si devono determinare i centri dei moduli alternati che rappresentano il pattern alternato del simbolo. Per trovarli verrà scandito un segmento di immagine che passa sul pattern alternato e se ne troveranno i centri.

A questo punto abbiamo a disposizione un poligono che incornicia perfettamente il simbolo. Inoltre sappiamo a priori (dal processo di allineamento ai contorni) quali sono i lati degli *handles* e quali quelli dei pattern alternati (*Syncs*). Se il simbolo fosse su una superficie perfettamente parallela al piano visivo, l'unica deformazione prospettica sarebbe un fattore di scala. Purtroppo non è quasi mai così. In genere

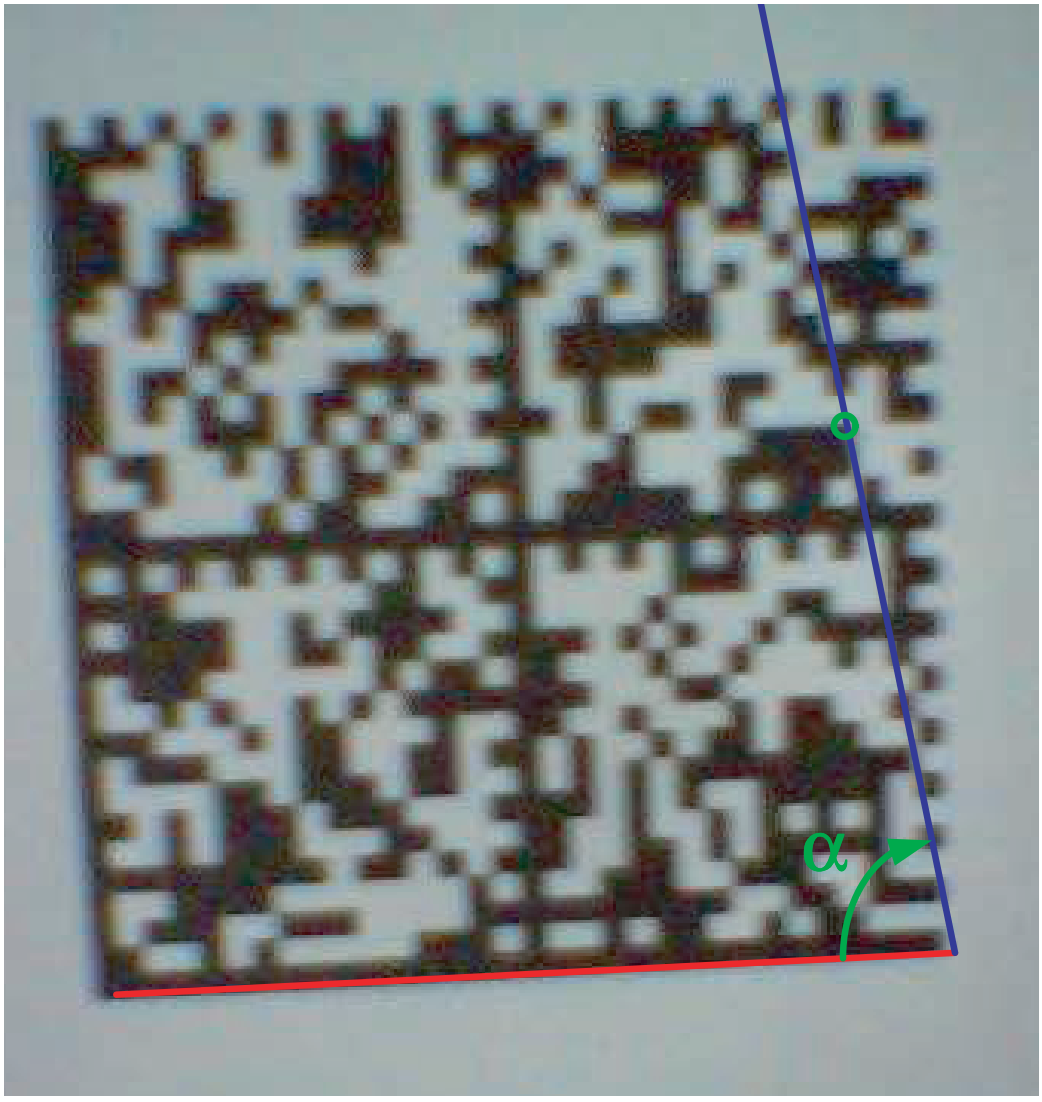


Figura 3.5: Algoritmo per incorniciare il pattern alternato

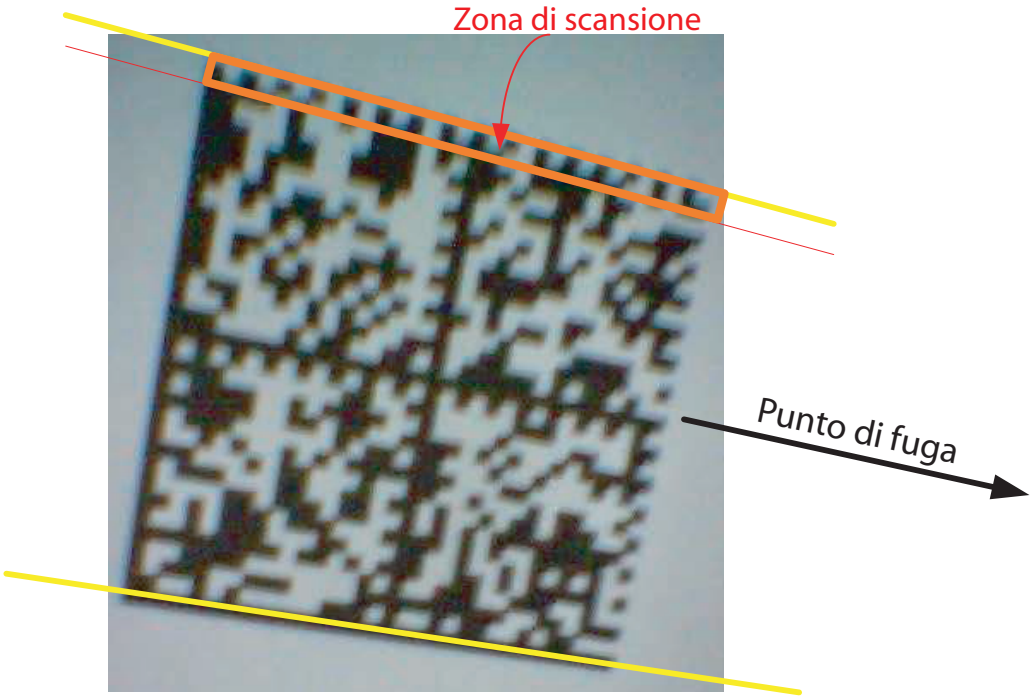


Figura 3.6: Prospettiva del simbolo

il piano è inclinato e il simbolo, pur essendo quadrato, presenta dei punti di fuga (figura 3.6).

Estrazione timing: a questo punto non resta che estrarre il timing. Si scandiscono i segmenti trovati con il passo precedente, quello che otteniamo è vettore di valori al quale si applicheranno i seguenti passi (per ciascuna *sync*):

1. viene determinata la lunghezza del segmento con la *box-distance*[4].
2. gli errori derivati dal filtraggio a soglia (un punto bianco in un segmento nero o viceversa), vengono eliminati con un fitro a media su finestra di 3 punti:

$$result(i) = \frac{\sum_{j=-1}^{+1} original(i+j)}{3}$$

che pur togliendo un po' di precisione elimina eventuali disturbi;

3. viene segmentato il vettore, per questa operazione si sfruttano i passaggi attraverso il valore 128 (medio fra minimo, 0 e massimo, 255);
4. vengono calcolati i centri di ciascun segmento;
5. si riscalano i centri in base alla lunghezza del segmento e se ne calcola la posizione.

Creazione della griglia: adesso si possono usare i centri di campionamento sulle *syncs* per ricavare tutta la griglia di campionamento. Per ogni centro si traccia una retta che passa attraverso il punto di fuga dell'altra *sync* e il centro stesso. Si traccia un'altra retta attraverso il punto di fuga della *sync* e un centro dell'altra *sync*. L'intersezione fra queste due rette da un centro della griglia di campionamento. Ripetendo questa operazione per ogni coppia di punti sulle due *sync* si crea la griglia (figura 3.7).

Campionamento: infine si possono campionare tutti i punti della griglia. Il risultato andrà in una matrice che poi sarà passata al *decoder*[1].

Usando la tecnica dei punti di fuga si ottiene il campionamento dei centri indipendentemente dalla deformazione prospettica del simbolo e dalla sua rotazione.

[1]La matrice non includerà i punti dei *syncs* e *handles* esterni, mentre includerà quelli degli *handles* interni che si trovano nei simboli con sottosimboli.

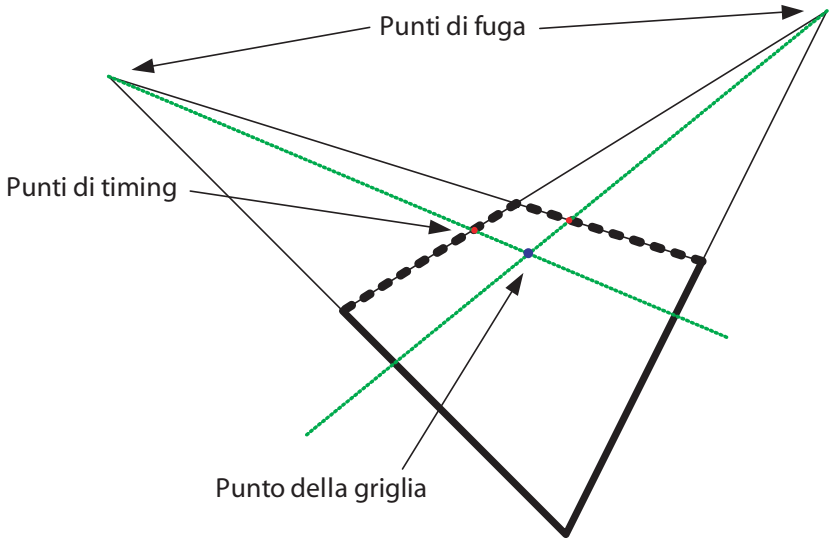


Figura 3.7: Processo geometrico per ottenere i punti della griglia

Decodifica

Il processo di decodifica viene effettuato dalla classe *Decoder*. Sono supportati solo i simboli che seguono la più recente specifica ECC200, pertanto le tecniche descritte di seguito si riferiscono a tale specifica.

Dal precedente processo di campionamento sono state ricavate una matrice che rappresenta i valori dei moduli nel simbolo e le dimensioni del simbolo. Sono supportati solo i simboli quadrati, sebbene le specifiche prevedano anche simboli rettangolari.

Creazione della mapping matrix: Il simbolo Data Matrix, se ha una dimensione superiore a 24×24 moduli (esclusi *handles* e *syncs*), sarà composto da dei sottosimboli[12]. Questo significa che all'interno del simbolo ci saranno delle ulteriori suddivisioni tramite delle *syncs* e *handles*. Le *syncs* e *handles* aggiuntive vanno rimosse, prima di passare alla fase di decodifica, in modo da creare la *mapping matrix*, cioè una matrice risultante dall'eliminazione delle strutture dei sottosimboli. Ovviamente

se il simbolo ha una dimensione inferiore a 24×24 questa operazione non è necessaria.

Per eliminare *syncs* e *handles* dai sottosimboli sfruttiamo il fatto che le dimensioni dei simboli sono fissate dallo standard e lo sono anche il numero di sottosimboli per una certa dimensione: a titolo esemplificativo, se ho un simbolo 26×26 avrò 4 sottosimboli e così via (vedi appendice D). Quindi, nota la dimensione del simbolo, si sa in che posizione sono i sottosimboli che si possono facilmente estrarre e mettere nella *mapping matrix*.

Creazione del buffer di byte: Ciascun *modulo* che compone la mapping matrix corrisponde ad un bit di una stringa di byte. In fase di codifica, questi bit sono posizionati sul simbolo secondo uno schema a "zig zag", che parte dall'angolo in alto a sinistra fino all'angolo in basso a destra. Questo algoritmo è anch'esso nelle specifiche sotto forma di codice C[11]. Applicando, al contrario, questo algoritmo alla mapping matrix si ottiene un buffer di *codeword*. Le codeword sono delle parole di 8 bit in cui è suddiviso lo *stream*.

Correzione errori: Lo standard Data Matrix ECC 200 prevede di inserire dei codici a correzione di errore che usino l'algoritmo di Reed-Solomon. Tramite questa tecnica si possono aggiungere bit di ridondanza ad un messaggio esistente, in modo tale che se si presentano degli errori nella decodifica verranno rilevati e, entro un certo limite, corretti. Per specificare la codifica usata con Reed-Solomon (d'ora in poi RS), si usa la dicitura $RS(n, k)$ con cui si intende che dato uno stream di k simboli, tutti composti da s bit, verranno aggiunti simboli di "parità" in modo da formare uno stream finale di n simboli. In questo modo sarà pos-

sibile correggere al più t errori, con t tale che $2t = n - k$. In parole povere, se si vogliono correggere t errori, sarà necessario aggiungere $2t$ simboli a quelli di base. Quindi una volta specificati il numero di bit s che compongono ciascun simbolo, n e k, si stabilisce univocamente come è codificato uno *stream*.

Nel caso in esame s vale 8 bit (numero di bit per codeword). Per quanto riguarda n e k, questi dipenderanno dalla dimensione del simbolo e sono indicati nelle specifiche[13] (si veda l'appendice D). A titolo di esempio, un simbolo 10×10, con *mapping matrix* 8×8, che quindi può contenere 64 bit (8 codeword) avrà 5 codeword di correzione. Si osserva che nei simboli più piccoli lo standard specifica una percentuale maggiore di simboli di correzione rispetto a quelli più grandi.

Le funzionalità di codifica e decodifica sono ottenute usando il package RSClasses, che è un *wrapper* alle librerie scritte da Benjamin Barras[3]. La funzionalità del package è gestita dalla classe RS, la quale presenta il costruttore:

```
public RS(int n, int t)
```

che permette di specificare il numero finale di codeword n e il numero massimo di errori t. Si osserva che il simbolo 10×10 dell'esempio sopracitato, non è supportato. Infatti t sarebbe dovuto essere 2.5 (non rappresentabile con una variabile di tipo int). La limitazione di questa libreria sta proprio nel fatto di dover specificare t invece di k. Tuttavia le dimensioni non usabili sono solo due: 10×10 e 12×12, che sono le più piccole e verrebbero usate raramente. Il vantaggio di questa libreria è dato dal fatto che in grado di girare su J2ME quasi senza modifiche.

RS presenta altri due metodi:

```
public byte[] decode(byte[] encoded)
```

e

```
public byte[] encode(byte[] buffer)
```

Il primo serve per *decodificare* un buffer di byte secondo i parametri inseriti nel costruttore. Il secondo per *codificare*. Questa classe wrapper è volutamente poco flessibile in modo da essere più semplice da usare nel contesto di Data Matrix. Con le librerie di Barras è anche possibile specificare il numero di bit per simbolo.

Decodifica: Lo standard Data Matrix permette numerose codifiche (vedi sezione 2.3). Viene usato ASCII per una maggiore praticità d'uso. Una volta ottenuta una stringa di byte codificata in ASCII, viene convertita in una String Java. Questo è il risultato del processo di decodifica ed è ciò che viene restituito dal metodo symRec.go().

Resto del package

Uniche funzionalità pubbliche del package sono quelle descritte, ovvero il costruttore

```
public symRec(byte []rawImmData, int width, int height)
                                        throws tagDecoderException;
```

e il metodo

```
public String go();
```

Se per qualche motivo la decodifica fallisce viene lanciata un'eccezione del tipo tagDecoderException, che può contenere i seguenti messaggi:

Error detecting syncs non si è riusciti a trovare le *syncs* con l'algoritmo descritto nel paragrafo "Allineamento al pattern alternato". Molto probabilmente il simbolo non è inquadrato bene.

No timing detected non si riesce ad estrarre un timing adeguato dalle *syncs*. Possibili cause possono essere scarsa risoluzione della foto o eccessivo disturbo.

Can't create sampling grid non si riesce a creare la griglia di campionamento. Questo accade se a monte si sono verificati degli errori e non sono stati rilevati.

Unsupported symbol size dimensione del simbolo non supportata. È il caso dei simboli non supportati, vedi paragrafo "decodifica".

Can't correct, too many errors si sono trovati degli errori, ma sono troppi ed è impossibile correggerli.

Unsupported Encodation si è usato una codifica diversa da quella ASCII.

Wrong symbol size la dimensione del simbolo non è fra quelle standard. Questo accade perché c'è un errore nel riconoscimento del timing o perché il simbolo è ECC 000-140.

Unknown encoding error errore sconosciuto generico.

3.2 Server

L'applicazione ServerMultiThread è stata sviluppata in linguaggio *Java 2 Standard Edition* (J2SE), questa ha il compito di svolgere servizi per le applicazioni installate sui dispositivi degli utenti (EasyPark) e degli addetti

al controllo delle aree di sosta (EasyCheck), vengono svolte diverse funzioni tra cui:

- gestione account utenti;
- gestione elenchi veicoli ammessi alla sosta;
- calcolo dei costi di parcheggio;
- gestione delle aree regolate da tempo massimo di permanenza;
- gestione credito prepagato (addebiti e accrediti).

L'applicazione opera come server multi thread e si appoggia a un database server MySQL per la memorizzazione dei dati. Per stabilire una connessione sicura, tra client e server, è necessario un certificato digitale.

3.2.1 Certificato digitale

Un certificato digitale è un documento elettronico che associa l'identità di un'entità (persona fisica, società, ecc...) ad una chiave pubblica, viene emesso da una *Autorità di Certificazione* riconosciuta secondo standard internazionali (X.509) e viene firmato con la chiave privata dell'Autorità. Gli enti che fanno da Autorità devono sottostare a regole rigidissime per quanto riguarda la gestione dei dati personali, pertanto si possono considerare sicuri.

I certificati per le applicazioni Java sono memorizzati in un file detto *keystore* che può essere manipolato con il programma keytool fornito con la distribuzione Java.

I certificati vengono rilasciati dietro pagamento (circa 15000 € quelli di VeriSign). Senza ricorrere ad un'Autorità di Certificazione è stato creato un

certificato con il programma keytool, tale certificato va bene per lo svolgimento delle nostre prove ed è stato utilizzato per firmare le applicazioni Java installate sui dispositivi mobili. Per creare un nuovo *keystore* è sufficiente utilizzare il comando

```
keytool -genkey -keystore testStore.ks -keyalg RSA
```

dove testStore.ks è il nome del file *keystore* che viene creato al termine della procedura guidata. Vengono richiesti i dati da inserire nel certificato:

```
jdk1.5.0_07\bin>keytool -genkey -keystore testStore.ks -keyalg RSA
Immettere la password del keystore:  segreto
Specificare nome e cognome
 [Unknown]:  Alessandro Mancini
Specificare il nome dell'unita' aziendale
 [Unknown]:
Specificare il nome dell'azienda
 [Unknown]:
Specificare la localita'
 [Unknown]:  Pisa
Specificare la provincia
 [Unknown]:  PI
Specificare il codice a due lettere del paese in cui si trova
l'unita'
 [Unknown]:  IT
Il dato CN=Alessandro Mancini, OU=Unknown, O=Unknown, L=Pisa, ST=PI,
C=IT e' corretto?
 [no]:  si

Immettere la password della chiave per <mykey>
        (INVIO se corrisponde alla password del keystore):
```

In questo modo è stato creato un certificato da utilizzare con il *server* e i *client*.

3.2.2 Accesso al Database

Per potersi collegare correttamente al database server MySQL è necessario definire i parametri di accesso in un file di testo di nome dbconfig.txt, questo deve contenere cinque linee con le seguenti informazioni:

1. indirizzo del server;
2. porta TCP;
3. nome del database;
4. username per l'accesso;
5. password per l'accesso.

Nell'esempio che segue vediamo i parametri di accesso per un database server residente sulla macchina locale, con credenziali di root. Il nome del database è park:

```
localhost
3306
park
root
mypass
```

3.2.3 Avvio server

Per lanciare l'esecuzione del *server* è necessario specificare obbligatoriamente il numero di porta "ascoltata" per le connessioni in entrata. È possibile

anche aggiungere un parametro opzionale v (o V), quest'ultimo permette di visualizzare le operazioni eseguite in corso di esecuzione da parte del server. Questa funzionalità era stata inserita in fase di sviluppo per individuare eventuali anomalie di esecuzione. Va specificato anche il file *keystore* dove sono contenuti i certificati digitali.

Vediamo come avviare il *server* in modo che utilizzi il certificato testStore.ks e sia in ascolto sulla porta 50000:

```
>java -Djavax.net.ssl.keyStore=testStore.ks
    -Djavax.net.ssl.keyStorePassword=segreto
    -jar ServerMultiThread.jar 50000 v
```

3.2.4 Protocollo client-server

La comunicazione tra applicazione mobile e *server* avviene tramite scambio di messaggi: la connessione viene stabilita utilizzando l'oggetto SSLServerSocket in modalità sicura grazie alle JSSE[2]. Una volta stabilita la connessione il *server* risponde con la stringa RDY ed è pronto ad accettare comandi. Si utilizza il comando XIT per chiudere la connessione[3].

Vediamo il set di comandi utilizzati dall'applicazione in dedicata agli automobilisti.

LIV utilizzato per segnalare l'abbandono di un'area di sosta. Il server risponde con una stringa alfanumerica di 8 byte generati a caso (detta *magic*) e attende in risposta dal *client* una stringa del tipo <targa>* <num>*, dove <targa> è la targa del veicolo segnalato e <num> è un numero di controllo.

[2] *Java Secure Socket Extention*

[3] La connessione può venir chiusa anche per *time-out*.

Il numero di controllo è stato introdotto per evitare che un malintenzionato possa collegarsi al *server* e segnalare l'arrivo o l'abbandono di un certo veicolo, attraverso il numero di targa (informazione pubblica poiché tutti i veicoli la espongono) al fine di creare disagio al altri utenti. Una spiegazione più dettagliata e funzionamento è descritto in appendice B.

Se l'operazione va a buon fine, il server risponde con OK seguito dal credito residuo del conto prepagato dell'utente (solo nel caso si utilizzi il meccanismo prepagato), altrimenti risponde con ER.

PRK utilizzato per segnalare l'arrivo di un certo veicolo in un'area di sosta. Il server risponde con la stringa *magic* allo stesso modo descritto per il comando LIV e attende dal client una stringa del tipo <codice_area>*<targa>*<num>*, dove <codice_area> è un codice identificativo dell'area di sosta, <targa> la targa del veicolo che è stato posteggiato e <num> il numero di controllo.

Se l'operazione va a buon fine il server risponde con OK, altrimenti con ER.

QRY permette di conoscere le regole dell'area di sosta. Questo comando viene utilizzato dal client per ottenere informazioni come tariffa oraria, limiti massimi di tempo, orari in cui si applica l'eventuale tariffa, ecc...

Il server attende il codice identificativo dell'area di sosta e risponde con OK seguito da una stringa del tipo <n>*<n>*<n>*<n>*<n>*<n>* <n>* dove <n> è un numero che codifica, nell'ordine, la tariffa oraria, i giorni in cui si applica tale tariffa, il numero minimo di minuti da pagare, il tempo massimo (in minuti) di permanenza, il numero di

minuti gratuiti, l'orario di inizio in cui si applica la tariffa e l'orario di fine applicazione.

Queste informazioni vengono mostrate all'utente prima di effettuare la sosta, in questo modo potrà decidere se lasciare il veicolo oppure andarsene.

RCC registrazione carta di credito. Si tratta di un comando per simulare la registrazione di una carta di credito, come metodo di pagamento post-pagato in alternativa al metodo prepagato.

Il server attende 4 stringhe che sono nell'ordine: codice identificativo dell'utente, nome del titolare della carta di credito, numero carta, scadenza. Il server risponde con OK se la registrazione è andata a buon fine o con ER in caso di problemi.

REG serve per annunciare la registrazione di un nuovo cliente. Il server attende 5 stringhe che contengono i dati dell'utente e devono rispettare nell'ordine: cognome, nome, indirizzo e-mail, numero telefonico, targa veicolo. Queste informazioni vengono archiviate sul database per gli usi necessari. Il server risponde con OK seguito dal codice identificativo dell'utente (4 byte) e da una stringa alfanumerica che verrà utilizzata per la generazione dei *numeri di controllo*. In caso di fallimento risponde con ER.

REV permette di recuperare i dati di un cliente già registrato, si evita così di dover effettuare una nuova registrazione si cambi telefono o vengano cancellati inavvertitamente i dati in uso dall'applicazione.

Il *server* attende due stringhe: il numero di targa dell'utente e la password personale, se queste informazioni sono corrette il server risponde

con OK seguito da una stringa del tipo <id>*<pag>*<credito>* <magic>* dove <id> è un codice identificativo assegnato all'utente, <pag> indica il metodo di pagamento utilizzato, <credito> il credito prepagato disponibile (vale 0 se il metodo di pagamento è diverso), <magic> è una stringa di caratteri, utilizzata per generare il numero di controllo, quando viene segnalato l'arrivo o l'abbandono di un'area di sosta.

In caso di fallimento risponde ER seguito da -NODATA per indicare che i dati inseriti non sono corretti o -WRONGPASS per indicare che è stata inserita una password errata.

SPW imposta una nuova password. Il server risponde con la stringa *magic* allo stesso modo descritto per il comando LIV e attende dal client una stringa del tipo <id>*<password>*<num>*, dove <id> è il codice idendificativo dell'utente, <password> la nuova password e <num> il numero di controllo. Se l'operazione va a buon fine, il server risponde con OK, altrimenti con ER.

SYB permette di interrogare il server e conoscere il credito prepagato di un cliente. Il server attende il codice identificativo del cliente e risponde con OK seguito dall'importo del credito. In caso di errori risponde con ER.

TUP utilizzato per effettuare una "ricarica", relativa al conto prepagato. Il server attende due stringhe: nell'ordine, il codice identificativo del cliente e il codice identificativo di una ricarica.

Se la procedura va a buon fine il server risponde con OK seguito dal credito aggiornato disponibile per il cliente, viene così allineata l'infor-

mazione nella memoria del telefono del cliente, mentre in caso di errori risponde con ER.

Vediamo adesso il set di comandi utilizzati dall'applicazione utilizzata dai controllori delle aree di sosta.

CAP restituisce il numero di posti auto disponibili in una certa area. Il server attende una stringa con il codice identificativo dell'area di sosta, risponde con OK seguito da un numero, che indica la capacità dell'area, in caso di errori risponde con ER.

CHK ottiene una lista delle targhe cui è concessa la sosta per una certa area. Il server attende una stringa con il codice identificativo dell'area di sosta. Risponde con OK, seguito dal numero di elementi dell'elenco e successivamente viene trasmesso l'intero elenco. In caso di errori risponde con ER.

CHU ottiene un aggiornamento dell'elenco delle targhe. Vengono trasmesse due liste: prima quella delle targhe da eliminare dall'elenco in memoria, successivamente la lista delle targhe da inserire. Questa funzione permette di trasmette una minore quantità di dati rispetto a dover ritrasmettere l'intero elenco ogni volta. Il server attende una stringa con il codice identificativo dell'area di sosta, risponde con OK e seguono nell'ordine: il numero di elementi da eliminare, il numero di elementi da inserire, l'elenco delle targhe da cancellare e l'elenco delle targhe da inserire. In caso di errori risponde con ER.

Field	Type	Null	Key	Default	Extra
code	char(8)	NO	PRI		
plate	char(8)	NO	PRI		

Tabella 3.1: Descrizione di allowed_plates, deletedallowedentries, newallowedentries

Field	Type	Null	Key	Default	Extra
code	char(8)	NO	PRI		
capacity	tinyint(3) unsigned	YES		NULL	
description	char(30)	YES		NULL	

Tabella 3.2: Descrizione di areadescription

3.3 Database

Di seguito analizziamo le tabelle memorizzate nel database denominato park e utilizzate dall'applicazione *server*.

allowed_plates Qui vengono memorizzate tutte le targhe dei veicoli autorizzati alla sosta (plate) e la relativa area (code) (Tab. 3.1).

areadescription Contiene una descrizione delle aree di sosta: codice identificativo (code), posti disponibili (capacity) e un campo description in cui è possibile inserire una stringa di testo che può essere utile per funzioni di ricerca (Tab. 3.2)

areatimetable Contiene le *regole* per ogni area di sosta (identificata dall'attributo code). La tariffa oraria (attributo rate) è espressa in centesimi. I tempi di addebito minimo (tmin), durata massima della sosta (tmax) e durata sosta gratuita (tfree) sono espressi in minuti. Gli orari di inizio e di fine

Field	Type	Null	Key	Default	Extra
code	char(8)	NO	PRI		
rate	smallint(5) unsigned	YES		NULL	
days	tinyint(3) unsigned	YES		NULL	
tmin	smallint(5) unsigned	YES		NULL	
tmax	smallint(5) unsigned	YES		NULL	
tfree	smallint(5) unsigned	YES		NULL	
begin	time	YES		NULL	
end	time	YES		NULL	

Tabella 3.3: Descrizione di areatimetable

applicazione della tariffa sono definiti dagli attributi begin e end. Si considera che al di fuori di tali orari la sosta sia libera. (Tab. 3.3).

L'attributo days definisce per quali giorni vale la regola. Si esprime con un valore numerico compreso tra 0 e 127, che rappresenta l'equivalente decimale di un numero binario a 7 cifre (per i 7 giorni della settimana). La cifra più significativa rappresenta il Lunedì mentre la Domenica si riferisce alla cifra meno significativa. Ad esempio il numero 1111110 = 126 rappresenta tutti i giorni della settimana escluso la Domenica (dal Lunedì al Sabato) oppure 1010100 = 84 rappresenta solo i giorni Lunedì, Mercoledì, Venerdì.

creditcard Questa tabella è stata realizzata a scopo dimostrativo e permette la memorizzazione degli estremi di una carta di credito: titolare (cardholder), numero di carta (number), scadenza (expirey). La carta è associata a un utente tramite il suo identificativo (id) (Tab. 3.4).

customer Contiene i dati dei clienti: cognome (surname), nome (name), indirizzo e-mail (email) e numero di telefono (phone). Si veda la Tabella 3.5.

Field	Type	Null	Key	Default	Extra
code	char(8)	NO	PRI		
cardholder	char(40)	YES		NULL	
number	char(16)	YES		NULL	
expirey	char(4)	YES		NULL	

Tabella 3.4: Descrizione di creditcard

Field	Type	Null	Key	Default	Extra
id	char(4)	NO	PRI		
magickey	char(8)	YES		NULL	
surname	char(30)	YES		NULL	
name	char(30)	YES		NULL	
email	char(30)	YES		NULL	
phone	char(16)	YES		NULL	
payment	char(1)	YES		NULL	

Tabella 3.5: Descrizione di customer

Il codice identificativo del cliente (id) viene generato dal server ed è costituito da una stringa alfanumerica di 4 caratteri casuali. Ovviamente l'identificativo deve essere univoco e nel caso venga creato un codice già esistente, questo viene scartato e generato uno nuovo. Si utilizzano solo caratteri ASCII dell'alfabeto (a . . . z, A . . . Z) e cifre (0 . . . 9), che sono sufficienti a identificare oltre 14 milioni di utenti.

L'attributo magickey contiene una stringa di 8 caratteri, anche questa viene generata in modo casuale dal server ed è utilizzata per la funzione di verifica con *numero di controllo*, offerta dal package magic (Appendice B).

Il metodo di pagamento viene descritto con un byte dall'attributo payment.

Field	Type	Null	Key	Default	Extra
plate	char(8)	YES		NULL	
expiry	datetime	YES		NULL	

Tabella 3.6: Descrizione di parking_expiry

deletedallowedentries Contiene le targhe (plate) dei veicoli che hanno abbandonato un'area di sosta (code). Questa tabella viene utilizzata come *meccanismo di consistenza della memoria* (vedi sezione 3.5.2) dell'applicazione utilizzata dagli addetti al controllo delle aree, durante la fase di aggiornamento degli elenchi dei veicoli autorizzati (Tab. 3.1).

newallowedentries Contiene l'elenco delle targhe (plate) che sono entrate in sosta autorizzata. Questo è un elenco parziale rispetto a quello della tabella allowed_plates, perché contiene solo i nuovi elementi a partire dall'ultima interrogazione, effettuata dall'applicazione di controllo. Si utilizza questa tabella come *meccanismo di consistenza della memoria*, permettendo l'aggiornamento della lista dei veicoli autorizzati, senza dover trasferire l'intero elenco ad ogni volta (Tab. 3.1).

parking_expiry Si utilizza per memorizzare le targhe (plate) ammesse in aree regolate da tempo massimo di permanenza. Il campo expiry contiene data e ora di scadenza del parcheggio (Tab. 3.6).

parking_times Qui vengono memorizzate tutte le soste autorizzate. La tabella (3.7) contiene le targhe (plate), codice area (areacode) e relativi orari di inizio/fine permanenza (begin, end).

password Contiene le password (passwd) dei clienti (id) (Tab. 3.8).

Field	Type	Null	Key	Default	Extra
plate	char(8)	YES		NULL	
areacode	char(8)	YES		NULL	
begin	datetime	YES		NULL	
end	datetime	YES		NULL	

Tabella 3.7: Descrizione di parking_times

Field	Type	Null	Key	Default	Extra
id	char(4)	NO	PRI		
passwd	char(8)	NO			

Tabella 3.8: Descrizione di password

prepaid_account Contiene l'importo del credito prepagato (credit) per i clienti (id) che hanno scelto questa forma di pagamento. L'importo è espresso in centesimi (Tab. 3.9).

prepaid_topup Questa tabella contiene i codici di ricarica, utilizzati dal sistema di credito prepagato. La ricarica è identificata da un codice (code) di 6 caratteri. L'importo (value) si intende in unità (Tab. 3.10). I codici vengono eliminati dopo il loro utilizzo.

Field	Type	Null	Key	Default	Extra
id	char(4)	NO	PRI		
credit	smallint(6)	YES		NULL	

Tabella 3.9: Descrizione di prepaid_account

Field	Type	Null	Key	Default	Extra
code	char(6)	NO	PRI		
value	tinyint(3) unsigned	YES		NULL	

Tabella 3.10: Descrizione di prepaid_topup

3.4 Applicazione utente

L'*applicazione utente*, denominata EasyPark, viene utilizzata dagli automobilisti. Questa permette di segnalare l'arrivo e il successivo abbandono da un'area di sosta ogni volta che si vuole parcheggiare il proprio mezzo, trasmettendo la targa dell'auto al *sistema centrale* che gestisce i parcheggi.

L'applicazione è stata sviluppata per la piattaforma J2ME, profilo MIDP 2.0, configurazione CLDC 1.1. È richiesto il supporto MMAPI (JSR 135) per poter catturare i *visualtag* con la fotocamera del telefono.

3.4.1 Primo utilizzo

L'applicazione deve essere configurata con i dati del cliente, questi vengono memorizzati nella memoria del telefono insieme ad altre informazioni inviare dal server (come il codice identificativo dell'utente e la stringa per generare i numeri di controllo).

Queste informazioni vengono memorizzate in *Record Store* gestito dall'ambiente Java. Nel caso l'applicazione venga eseguita la prima volta, questo archivio sarà assente, pertanto verrà avviata una procedura che guida l'utente nell'inserimento dei propri dati personali. Analogamente, se il Record Store viene cancellato o non contiene dati, verrà richiesta la registrazione di un nuovo utente.

Figura 3.8: Registrazione nuovo utente.

In alternativa, se il cliente è già registrato, è possibile recuperare i propri dati (fig. 3.8).

Se si sceglie il metodo di pagamento con *carta di credito* viene proposta un'altra procedura guidata, in cui sarà necessario inserire i dati di una carta. Questa funzione è solo dimostrativa, l'unico controllo effettuato è la consistenza del numero inserito attraverso la *formula di Luhn* (appendice C). Le informazioni vengono poi memorizzate (insieme ai dati utente) nel *sistema centrale* e tutte le funzioni relative alla gestione del credito *prepagato* vengono disabilitate.

(a) (b)

Figura 3.9: Procedura di "ricarica" del credito prepagato.

3.4.2 Gestione credito prepagato

L'importo del credito prepagato è mantenuto sia sul *sistema centrale* che nella memoria del telefono cellulare. Se il credito risulta inferiore a una certa soglia, l'utente non potrà parcheggiare e la sola funzione abilitata è quella che serve ad effettuare una ricarica. In alternativa può cambiare metodo di pagamento (passando alla modalità con *carta di credito*). Il valore di soglia è definito nella classe controller (nell'esempio il valore 100 definisce €1.00):

```
private static int creditBias = 100;
```

Le ricariche sono identificate da un *visualtag* che l'utente dovrà fotografare, questo sistema è intuitivo ed evita di dover inserire manualmente sequenze numeriche. Una volta acquisito e decodificato il *visualtag* l'applicazione interrogherà il *sistema centrale* che provvederà ad aggiornare l'importo di credito disponibile (fig. 3.9).

L'utente potrà verificare in qualsiasi momento il credito disponibile, selezionando l'apposita voce nel menù principale. L'informazione in memoria è

allineata con quella mantenuta sul *sistema centrale* e viene aggiornata durante le operazioni in cui è previsto un addebito (come ad esempio la segnalazione di *fine permanenza*).

Nel *visualtag* di una ricarica è codificata una stringa di caratteri del tipo:

`TUP;<codice_ricarica>*`

dove la sequenza `TUP` identifica che si tratta di una ricarica, mentre il codice univoco che la identifica è contenuto tra i caratteri `;` (punto e virgola) e `*` (asterisco).

3.4.3 Effettuare la sosta

Quando l'applicazione è configurata correttamente per il suo utilizzo, viene mostrato un menu come quello in figura 3.11 (a). Per parcheggiare il proprio veicolo è sufficiente identificare l'area di sosta catturando l'immagine del *visualtag* che la contraddistingue. Il diagramma di stato in figura 3.10 ne mostra il funzionamento.

Nel *visualtag* di un'area di sosta è codificata una stringa di caratteri del tipo:

`PRK;<codice_area>*`

dove la sequenza `PRK` identifica che si tratta di un parcheggio, mentre il codice che identifica l'area è contenuto tra i caratteri `;` (punto e virgola) e `*` (asterisco).

Una volta catturata e decodificata l'immagine, l'applicazione interroga il sistema centrale per recuperare le informazioni che regolano l'area di sosta (come ad esempio tariffe e orari), queste vengono mostrate all'utente che potrà così decidere se parcheggiare il mezzo o andarsene (fig. 3.11 d).

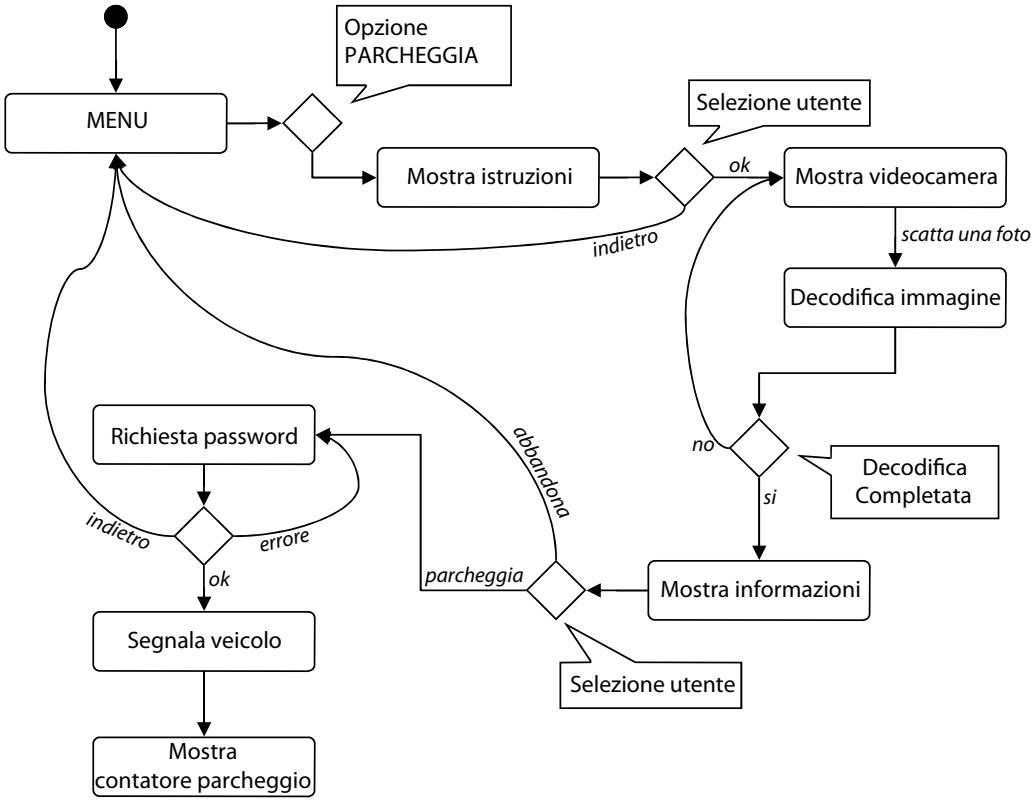


Figura 3.10: State diagram funzione 'parcheggia'

Selezionando *Parcheggia* l'utente accetta le condizioni e potrà lasciare il veicolo nell'area di sosta. Viene trasmessa la targa dell'autovettura al *sistema centrale*. A conferma viene mostrata una schermata con un timer *conta tempo* e l'indicazione dell'eventuale importo addebitato. Questa indicazione è puramente indicativa, il costo visualizzato viene calcolato sulla base del tempo trascorso moltiplicato la tariffa oraria. L'importo esatto del parcheggio viene calcolato dal *sistema centrale* secondo i criteri che regolano l'area di sosta e l'esatto orario di arrivo/abbandono. (fig. 3.11 f).

Se l'area è regolata da un tempo massimo di permanenza (come in esempio), viene visualizzato anche un *indicatore di progresso*, sotto il riquadro di tempo e costo, tale indicatore avanza fino a riempire la piccola *barra* bianca e assume diversi colori (verde, giallo o rosso) all'avvicinarsi dello scadere del tempo massimo (come in figura). Con l'avvicinarsi al tempo massimo di permanenza, l'applicazione avviserà l'utente anche con dei messaggi audio. Il primo avviso viene riprodotto 15 minuti prima dello scadere della sosta, successivamente è previsto un segnale ogni 5 minuti. I messaggi audio sono codificati sotto forma di file .wav, pertanto possono essere personalizzati facilmente.

Per maggiore sicurezza è possibile proteggere l'operazione con la richiesta della *password* definita al primo utilizzo (o successivamente modificata), questo controllo può anche essere successivamente disattivato agendo sulla *check-box* Ricorda o modificando le opzioni del programma(fig. 3.11 e).

Dopo aver segnalato l'arrivo è possibile chiudere l'applicazione: selezionando il pulsante STOP (fig. 3.11 f) viene mostrato un menù che permette all'utente di segnalare l'uscita dall'area, effettuare una ricarica (solo nel caso si utilizzi il metodo prepagato) o terminare l'applicazione (per chiuderla). In quest'ultimo caso, al successivo avvio, viene riproposta la schermata in figura

Figura 3.11: Procedura di segnalazione sosta.

3.11 (f) fino a quando l'utente non segnala l'abbandono dell'area di sosta, così si da la possibilità di verificare il tempo trascorso e l'eventuale costo. È utile anche per verificare in modo immediato l'avvicinarsi dello scadere del tempo massimo di sosta, per le aree regolate da un limite massimo di permanenza.

3.4.4 Indirizzo server di rete

L'applicazione scambia messaggi con il *server* del *sistema centrale*, l'indirizzo è definito dalle variabili SERVER_ADDRESS (contenente IP o indirizzo

simbolico) e SERVER_PORT per la porta TCP, nel package controller:

```
private static final String SERVER_ADDRESS = "example.com";
private static final int SERVER_PORT = 50000;
```

3.5 Applicazione controllore

L'*applicazione controllore*, denominata EasyCheck, viene utilizzata dagli addetti al controllo dei parcheggi, questa permette essenzialmente di conoscere a quali veicoli è concessa la sosta, per una certa area, scaricando l'elenco delle targhe dal *sistema centrale*.

L'applicazione è stata sviluppata per la piattaforma J2ME, profilo MIDP 2.0, configurazione CLDC 1.1. È richiesto il supporto MMAPI (JSR 135) per poter catturare i *visualtag* con la fotocamera del telefono.

3.5.1 Utilizzo

L'applicazione si occupa di identificare l'area di sosta soggetta a controllo tramite i *visualtag* disposti nel parcheggio, quindi viene richiesto al *sistema centrale* l'invio dell'elenco delle targhe dei veicoli ammessi (figura 3.12).

Con un semplice strumento di ricerca è possibile conoscere l'elenco delle targhe in memoria, raffinando la ricerca inserendo una parte della stringa da cercare (figure 3.13 e 3.14).

Nel caso non venga trovato alcun risultato viene visualizzato il messaggio NESSUNA CORRISPONDENZA, questa condizione fa scattare anche l'aggiornamento dell'elenco, come descritto di seguito.

In qualsiasi momento è possibile ricaricare l'intera lista selezionando Ricarica lista dal menu principale (fig. 3.13 a), in questo modo, a differenza

(a) (b) (c)

Figura 3.12: Identificazione area e caricamento dati

(a) (b) (c)

Figura 3.13: Ricerca targhe (tutte quelle che iniziano con CJ)

(a) (b)

Figura 3.14: Ricerca targhe (tutte quelle che iniziano con C)

della procedura di aggiornamento, viene trasmesso l'elenco di tutte le targhe presenti nell'area, e reimpostato l'elenco.

3.5.2 Algoritmo di consistenza

Il dispositivo mobile utilizzato dagli addetti al controllo non resta costantemente connesso con il *sistema centrale*: tenere la trasmissione sempre attiva implica un consumo maggiore di energia, diminuendone l'autonomia inoltre, in movimento, il dispositivo potrebbe capitare in una posizione non raggiunta dal segnale radio della rete mobile e perdere così il collegamento.

È stato pensato un semplice algoritmo di *consistenza della memoria* che permette di aggiornare l'elenco memorizzato sul dispositivo mobile con quello presente nel *sistema centrale*. In questo modo l'applicazione effettua brevi trasmissioni solo quando necessario, inoltre l'operazione impiega poco tempo permettendo così di ridurre eventuali problemi dovuti a interruzione della connessione (che possono essere piuttosto probabili data la natura del collegamento).

L'elenco di targhe in memoria viene aggiornato periodicamente, la fre-

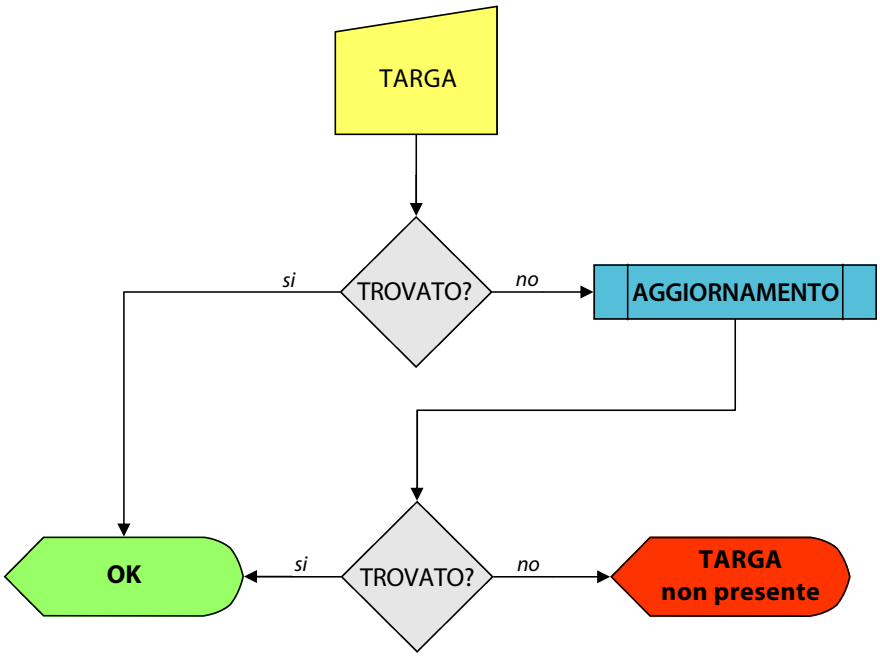


Figura 3.15: Algoritmo di consistenza della memoria.

quenza degli aggiornamenti definita dalla variabile autorefreshMinutes nel package controller, nell'esempio vediamo che impostato su 15 minuti:

```
private static final int autorefreshMinutes = 15;
```

L'elenco delle targhe viene memorizzato in un array di stringhe e gestito con il package list (appendice A). Questo mette a disposizione alcune funzioni pensate per l'inserimento, eliminazione e ricerca degli elementi memorizzati. L'array è dimensionato in base alla capienza dell'area di sosta.

La fase di aggiornamento prevede la ricezione di un elenco di targhe, di questo, la prima parte contiene tutti gli elementi da eliminare (si libera memoria) e la seconda, contiene gli elementi da inserire. La trasmissione di questa lista viene preceduta da due numeri, che indicano, rispettivamente, quanti elementi vanno eliminati e quanti inseriti.

L'aggiornamento, come schematizzato in figura 3.15, viene attivato ogni

volta che la ricerca di una targa non produce alcun risultato. Questo serve a proteggere l'utente, nel caso la sua targa sia stata segnalata dopo l'ultimo aggiornamento effettuato dall'applicazione controllore.

3.5.3 Indirizzo server di rete

L'applicazione scambia messaggi con il server del sistema centrale, l'indirizzo è definito dalle variabili SERVER_ADDRESS (contenente IP o indirizzo simbolico) e SERVER_PORT per la porta TCP del ackage controller:

```
private static final String SERVER_ADDRESS = "example.com";
private static final int SERVER_PORT = 50000;
```

Capitolo 4

Conclusioni e sviluppi futuri

Osservando le figure relative all'applicazione dimostrativa, si nota come sia stato realizzato un sistema per la gestione di aree di sosta più semplice da usare. L'utilizzo di *visualtag* ha permesso di nascondere tutta la complessità dei vari componenti del sistema dietro un'etichetta facilmente riconoscibile e un'interfaccia utente molto semplificata. Era possibile ottenere la stessa cosa utilizzando le tecnologie esistenti, ad esempio inviando SMS compilati con i comandi giusti ad un *sistema centrale*, ma questa non sarebbe un'operazione di immediata comprensione. Inoltre, la totale mancanza di un'interfaccia che guida l'utente nelle sue azioni, porterebbe a una certa percentuale di operazioni non andate a buon fine (basti pensare ai comuni errori di inserimento del testo in un messaggio).

I *visualtag* permettono di acquisire *stringhe numeriche o di testo* di varia lunghezza, sollevando l'utente dal doverle digitare sulla piccola tastiera del proprio telefono, secondo un modello *"inquadra-e-scatta"* che risulta più veloce, intuitivo ed elimina il fattore di errore umano nell'immissione dati.

L'ambiente Java è molto adatto allo sviluppo di *applicazioni immersive*, poiché fornisce un framework ampio e semplice da usare, con il quale si

possono realizzare innumerevoli soluzioni che possono operare su piattaforme diverse. Ciò permette di sfruttare i nuovi telefoni cellulari senza ricorrere ad hardware dedicato con beneficio sui costi e maggiore diffusione sui potenziali utenti.

Le applicazioni funzionanti sui telefoni cellulari comunicano con il *sistema centrale* via Internet, attraverso una connessione dati, questa modalità è supportata da tutte le reti di telefonia mobile GSM o UMTS. I costi di comunicazione vengono normalmente contabilizzati sulla base del *volume di dati scambiati* (o a *forfait* per certe tipologie di abbonamento). La quantità di dati scambiati è molto ridotta e dalle prove effettuate gli addebiti applicati dall'operatore mobile sono risultati nell'ordine di $1 \sim 2$ €cent, si tratta di una riduzione notevole dei costi rispetto all'invio di un singolo SMS (che può costare fino a 15 volte in più). In ogni caso, l'evoluzione delle reti 3G/UMTS porterà un ulteriore abbattimento dei costi di comunicazione.

Una limitazione dell'ambiente Java è la ridotta capacità di acquisizione delle immagini attraverso la fotocamera del telefono, infatti le MMAPI J2ME permettono di ottenere in ingresso piccole foto alla risoluzione di 120×160 pixel, sebbene i dispositivi attuali siano in grado di acquisire immagini a risoluzioni che sono nell'ordine di $1 \sim 2$ Mpixel. Questo costringe a utilizzare simboli "piccoli" e poco capienti.

Nonostante queste condizioni è stato comunque possibile realizzare un'applicazione sufficientemente stabile e veloce nello svolgere le varie operazioni. Con questo modello di interazione nuovo e intuitivo è possibile creare un *ambiente immersivo*, in cui all'utente, attraverso i dispositivi mobili è data la possibilità di accedere a servizi di varia natura.

La rapida evoluzione dei telefoni cellulari, e delle reti di comunicazione, renderà ancora più semplice e veloce l'accesso a questi nuovi servizi.

Appendice A

Package list

Il package list è stato realizzato per gestire in modo semplice *elenchi di stringhe*, in particolare viene utilizzato nella memorizzazione di elenchi di targhe e nella successiva ricerca degli elementi memorizzati.

A.1 Utilizzo

Il package permette di realizzare l'oggetto Series, che altro non è che un *array* di oggetti String, si crea l'oggetto con `new Series()` che deve poi essere inizializzato definendo il numero di elementi in grado di memorizzare[1], in aggiunta sono disponibili metodi per le operazioni più comuni come inserimento, ricerca o eliminazione, vediamo di seguito i metodi pubblici e la loro funzione.

void deleteAll() cancella tutti gli elementi memorizzati nell'array.

int freeElements() restituisce il numero degli elementi disponibili alla memorizzazione.

[1]Si può utilizzare il metodo int setSequence(String seq, int separator) anche senza inizializzare l'oggetto Series.

String get(int i) restituisce il contenuto dell'elemento i, ritorna null se viene indicato un elemento vuoto.

String[] getList() restituisce un array di oggetti String contenente i soli elementi memorizzati, ritorna null se non è disponibile alcun elemento.

String getSequence(int separator) restituisce una stringa di caratteri contenenti tutti gli elementi memorizzati divisi da un carattere *separatore*. Il separator è il codice ASCII del carattere separatore. Se non vi sono elementi memorizzati viene restituita una stringa vuota.

void initialize(int n) inizializza l'oggetto con n elementi vuoti.

int insert(String item) memorizza un elemento (item) e ritorna l'indice in cui è stato posizionato.

boolean insert(String[] list) memorizza tutti gli elementi del vettore list, ritorna false se il numero di elementi disponibili è insufficiente, true se l'operazione è andata a buon fine.

int insert(int number) memorizza un numero convertendolo in stringa, ritorna l'indice in cui è stato posizionato.

String[] matchWith(String string) restituisce un array di stringhe con tutti gli elementi memorizzati la cui stringa combacia in tutto o in parte con string.

int numOfElements() restituisce il numero di elementi del vettore.

int remove(String item) cancella il primo elemento corrispondente alla stringa item, ritorna -1 se la stringa non esiste.

int remove(int i) cancella l'elemento i, ritorna -1 se è stato indicato un elemento che non esiste.

int setSequence(String seq, int separator) inizializza un array con gli elementi in sequenza separati da un carattere separatore. La variabile separator rappresenta il valore ASCII del separatore.

int storedElements() restituisce il numero di elementi memorizzati.

Appendice B

Package magic

Il package magic è stato realizzato per generare un *numero di controllo* sulle stringhe scambiate tra *client* e *server*, questo serve per verificare che il messaggio provenga effettivamente dall'utente che ha generato la richiesta e non da un malintenzionato che vuol provocare disturbo. Non dovendo proteggere informazioni sensibili, questo sistema, permette di raggiungere un buon livello di sicurezza senza dover mettere in campo algoritmi eccessivamente complessi.

B.1 Funzionamento

Il suo funzionamento si basa su due stringhe (chiamate *chiavi*) di lunghezza identica tra loro e una stringa di lunghezza qualsiasi (che rappresenta il *messaggio* da trasmettere). Il *messaggio* viene diviso in parti coincidenti con la lunghezza delle *chiavi*, pertanto deve essere di lunghezza multipla, se non lo è viene eseguito un *riempimento* con il carattere *'punto'*.

Il *numero di controllo* scaturisce da una serie di operazioni di *moltiplica-*

zione e *modulo* applicate iterativamente sui valori ASCII dei caratteri delle chiavi e del messaggio.

Siano le due *chiavi* K_A e K_B entrambe stringhe di lunghezza n:

$$K_A = (K_{A_1}, K_{A_2}, \ldots, K_{A_n})$$

$$K_B = (K_{B_1}, K_{B_2}, \ldots, K_{B_n})$$

e il *messaggio* M una stringa di caratteri di lunghezza m (con m multiplo intero di n[1]):

$$M = (M_1, M_2, \ldots, M_m)$$

Si applica il seguente algoritmo ricorsivo:

$$\sum_{j=1}^{t} \left[\sum_{i=1}^{n} (K_{A_i} \cdot K_{B_i} \cdot M_{(i \cdot j)}) \right]_{mod} \qquad \text{dove} \qquad t = \frac{m}{n}$$

mod è il modulo definito inizialmente.

B.2 Utilizzo

Il package permette di realizzare l'oggetto MagicNumber che si inizializza con `MagicNumber(int n, int m)`, dove `n` è la lunghezza delle due *chiavi* e `m` definisce il *modulo*, i metodi pubblici sono spiegati di seguito.

boolean checkMagicNumber(String message) verifica il messaggio e il numero di controllo (incluso nel messaggio), la stringa in ingresso deve essere del tipo `xxxx*NNN*` dove `NNN` è il *numero di controllo* che deve

[1] La lunghezza del messaggio viene portata alla lunghezza giusta con un'operazione di *riempimento*.

essere separato da asterischi dal resto del messaggio. Ritorna true se il controllo ha dato esito positivo, altrimenti false.

boolean checkMagicNumber(String message, int num) verifica il messaggio con il numero di controllo della variabile num. Ritorna true se il controllo ha dato esito positivo, altrimenti false.

int getMagicNumber(String message) ritorna un numero di controllo sulla base della stringa fornita.

boolean setFirstKey(String key) imposta la prima *chiave* utilizzata dall'algoritmo che genera il *numero di controllo*. Ritorna true se la chiave è stata memorizzata, altrimenti (ad esempio la dimensione non è corretta) false.

boolean setSecondKey(String key) imposta la seconda *chiave* utilizzata dall'algoritmo che genera il *numero di controllo*. Ritorna true se la chiave è stata memorizzata, altrimenti (ad esempio la dimensione non è corretta) false.

Appendice C

Formula di Luhn

La *formula di Luhn*, anche conosciuta come *Modulo 10*, è un semplice algoritmo di controllo utilizzato per la verifica di una varietà di numeri di identificazione, tra cui anche le carte di credito. È stato creato da uno scienziato della IBM di nome Hans Peter Luhn nel 1954.

L'algoritmo è di pubblico dominio e largamente utilizzato, non è inteso come funzione *hash* crittografica, è stato progettato per proteggere da errori accidentali ma non da attacchi di malintenzionati.

C.1 Spiegazione informale

La formula verifica un numero sulla base di una *cifra di controllo* inclusa nel numero stesso. Il numero deve passare il seguente test:

1. partendo dalla cifra più a destra (la *cifra di controllo*) e muovendosi verso sinistra, si raddoppia il valore delle cifre ogni due posizioni. Per ogni cifra che, raddoppiata, assume il valore di 10 o più, si sommano le sue cifre insieme. Ad esempio: 1111 diventa 2121, mentre 8763 diventa 7733 ($2 \times 6 = 12 \rightarrow 1 + 2 = 3$ e $2 \times 8 = 16 \rightarrow 1 + 6 = 7$);

2. si sommano tutte le cifre insieme. Ad esempio: se 1111 diventa 2121, allora 2+1+2+1 fa 6, mentre 8763 diventa 7733, quindi 7+7+3+3 fa 20;

3. se la somma finisce con 0 (il totale modulo 10 è congruente con 0) allora il numero è *valido* secondo la *formula di Luhn*, altrimenti *non è valido*. Quindi, 1111 *non è valido* (la somma fa 6) mentre 8763 è *valido* (la somma fa 20).

C.2 Punti di forza e di debolezza

L'*algoritmo di Luhn* permette di individuare gli errori di una cifra, più comunemente dovuti a trasposizione di cifre adiacenti, mentre non è in grado di individuare la trasposizione di due cifre in sequenza da 09 a 90[1] (o viceversa). Altri algoritmi più complessi (come quello di *Verhoeff*) possono individuare più errori. L'*algoritmo di Luhn modulo N* è una variante che permette di essere applicata a stringhe non numeriche.

C.3 Package luhn

Il package luhn mette a disposizione l'oggetto Mod10 che permette di verificare un numero secondo la *formula di Luhn* o di generare un numero completo di *cifra di controllo*.

[1] ad esempio 1909 e 1099 rispettano la *formula di Luhn* e hanno la stessa *cifra di controllo*, così come 3905 e 3095.

C.3.1 Utilizzo

Si inizializza l'oggetto Mod10 con `new Mod10()` quindi si hanno a disposizione i seguenti due metodi a cui va dato in ingresso una stringa di caratteri contenente solo cifre:

boolean check(String number) ritorna true se il numero rispetta la *formula di Luhn*, altrimenti false. Nei casi in cui la stringa number valga null, contenga caratteri che non sono cifre o abbia un numero di cifre dispari, il metodo ritorna false.

String gen(String number) ritorna una stringa di testo contenente il numero completato della sua *cifra di controllo*. La stringa deve contenere un numero dispari di cifre. Se viene data in ingresso una stringa non valida il metodo ritorna un messaggio:

- `NULL`: è stato dato in ingresso il valore null;
- `GIVE-ONLY-DIGITS`: la stringa in ingresso non contiene solo cifre;
- `GIVE-MORE-DIGITS`: la stringa non contiene alcuna cifra o il numero di cifre è pari.

Appendice D

Tabelle riassuntive simboli Data Matrix

Lo standard di codifica Data Matrix prevede 6 simboli *rettangolari* e 24 simboli *quadrati*. Le tabelle seguenti forniscono i valori di base dei simboli per dimensione.

Dim. simbolo	Regioni	CW Reed-Solomon	Blocchi
8×18	1	7	1
8×32	2	11	1
12×26	1	14	1
12×36	1×2	18	1
16×36	1×2	24	1
16×48	1×2	28	1

Tabella D.1: Valori simboli rettangolari

Dim. simbolo	Regioni	CW Reed-Solomon	Blocchi
10×10	1	5	1
12×12	1	7	1
14×14	1	10	1
16×16	1	12	1
18×18	1	14	1
20×20	1	18	1
22×22	1	20	1
24×24	1	24	1
26×26	1	28	1
32×32	2×2	36	1
36×36	2×2	42	1
40×40	2×2	48	1
44×44	2×2	56	1
48×48	2×2	68	1
52×52	2×2	2×42	2
64×64	4×4	2×56	2
72×72	4×4	4×36	4
80×80	4×4	4×48	4
88×88	4×4	4×56	4
96×96	4×4	4×68	4
104×104	4×4	6×56	6
120×120	6×6	6×68	6
132×132	6×6	8×62	8
144×144	6×6	10×62	8

Tabella D.2: Valori simboli quadrati

Appendice E

Glossario

3G Nell'ambito della telefonia mobile con il termine *3G* (acronimo di 3^{rd} Generation) si indicano, relativamente a tale campo, le tecnologie e gli standard di *terza generazione*.

API *Application Program(ming) Interface*, indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per un determinato compito.

Bluetooth Specifica industriale per *Personal Area Networks* (PAN) senza fili, conosciuto anche come IEEE 802.15.1, permette di scambiare informazioni tra telefoni cellulari, palmari, computer e altri dispositivi utilizzando onde radio. Questo standard offre bassi consumi, un corto raggio di azione (da 10 a 100 metri) e un basso costo di produzione per dispositivi compatibili.

EDGE *Enhanced Data rates for GSM Evolution*, è una delle tecnologie di telefonia mobile che permette di realizzare trasferimenti dati a media velocità e maggior stabilità. Viene convenzionalmente definita generazione 2.75, vale a dire una via di mezzo della seconda generazione ma molto orientata al 3G.

GNU GPL *GNU General Public License* è una licenza per software libero, permette all'utente libertà di utilizzo, copia, modifica e distribuzione. Il testo ufficiale della licenza è disponibile all'indirizzo http://www.gnu.org/licenses/gpl.html.

GPRS *General Packet Radio Service* è una delle tecnologie di telefonia mobile. Viene convenzionalmente definita di generazione 2.5, vale a dire una via di mezzo fra la seconda e la terza generazione. È stato progettato per realizzare il trasferimento di dati a media velocità.

GSM *Global System for Mobile Communications* è attualmente lo standard di telefonia mobile pi diffuso del mondo. È stato lanciato sul mercato come sistema di seconda generazione (2G). Offre la possibilità di scambiare dati, oltre che conversazioni.

GUI interfaccia grafica (in inglese *graphical user interface*, abbreviato GUI).

IP IP è un protocollo di *interconnessione di reti* (*Inter-Networking Protocol*), nato per interconnettere reti eterogenee per tecnologia, prestazioni, gestione.

LAN *Local Area Network*, in italiano *rete locale*. Identifica una rete costituita da computer collegati tra loro all'interno di un ambito fisico delimitato.

MAC L'indirizzo MAC viene detto anche *indirizzo fisico* o *indirizzo ethernet* o *indirizzo LAN*, ed è un codice di 48 bit (6 byte) assegnato in modo univoco ad ogni scheda di rete ethernet prodotta al mondo.

MMS *Multimedia Messaging Service*, è la logica evoluzione di SMS. I telefoni cellulari abilitati a MMS permettono di inviare messaggi composti da una o più parti multimediali (foto digitali, audio, video).

Open Source Indica un software rilasciato con un tipo di licenza per la quale il codice sorgente è lasciato alla disponibilità di eventuali sviluppatori, in modo che con la collaborazione il prodotto finale possa raggiungere una complessità maggiore di quanto potrebbe ottenere un singolo gruppo di programmazione.

PDA *Personal Digital Assistant*, è un computer di ridotte dimensioni, tale da essere portato sul palmo di una mano (detto anche *palmare*), dotato di uno schermo sensibile al tocco (o Touch Screen).

Reti di Sensori Reti wireless costituite da unità autonome sparpagliate ed equipaggiate di sensori. Queste cooperano comunicando tra loro nel monitoraggio di condizioni fisiche o ambientali (come temperatura, pressione, vibrazioni, inquinamento) di un certo ambiente.

Smart phone Dispositivo portatile che abbina funzionalità di gestione di dati personali e di telefono. Può derivare dall'evoluzione di un PDA a cui si aggiungono funzioni di telefono, o, viceversa, di un telefono mobile a cui si aggiungono funzioni di PDA. La caratteristica più interessante degli smart phone è la possibilità di installarvi altri programmi applicativi, che aggiungono nuove funzionalità. Questi programmi possono essere sviluppati dal produttore, dallo stesso utilizzatore, o da Terze parti.

SMS *Short Message Service*, comunemente usato per indicare un breve messaggio di testo inviato da un telefono cellulare ad un altro, con un costo

esiguo. Il termine corretto sarebbe SM (*Short Message*), ma ormai è invalso l'uso di indicare il singolo messaggio col nome del servizio.

TCP *Transmission Control Protocol* (TCP) è uno dei principali protocolli della Suite di protocolli Internet. TCP è il protocollo di trasporto, definito nel RFC 793, su cui si appoggiano gran parte delle applicazioni Internet.

UMTS *Universal Mobile Telecommunications System* (UMTS) è una delle tecnologie di telefonia mobile di terza generazione. Il sistema UMTS supporta velocità di trasmissione che possono arrivare a 1920 kbit/s. In un prossimo futuro le attuali reti UMTS potranno essere potenziate mediante il sistema di accesso denominato HSDPA (*High Speed Downlink Packet Access*), con una velocità massima teorica di trasferimento dati di 10 Mbit/s.

Visualtag Simbolo bidimensionale che può essere riconosciuto da un dispositivo mobile, tramite un sistema lettura o l'acquisizione di un'immagine digitale.

Wi-Fi *Wireless Fidelity*, è il nome commerciale delle reti locali senza fili (WLAN) basate sulle specifiche IEEE 802.11. Le reti Wi-Fi sono infrastrutture relativamente economiche e di veloce attivazione e permettono di realizzare sistemi flessibili per la trasmissione di dati usando frequenze radio, estendendo o collegando reti esistenti ovvero creandone di nuove.

Bibliografia

[1] Qr code standardidation. http://www.denso-wave.com/qrcode/qrstandard-e.html.

[2] Vicom - virtual immersive communications. http://www.vicom-project.it/.

[3] Benjamin Barras. *Simulation de codes de Reed-Solomon*, 2001. http://lgl.epfl.ch/~barras/projects/index.html.

[4] Rossi Dario. *Applicazioni immersive per dispositivi mobili basate su Visualtag*. Tesi di Dottorato di Ricerca, Università di Pisa, 2005.

[5] Abeer George Ghuneim. Contour tracing. http://www.imageprocessingplace.com/.

[6] Toshiki Iso, Yoshinori Isoda, Kiyotaka Otsuji, Hiroki Suzuki, Shoji Kurakake, e Toshiaki Sugimura. Platform technology for ubiquitous services. *NTT Technical Review*, 1(8):82–88, novembre 2003.

[7] Linda Shapiro e George Stockman. *Computer Vision*, p. 69 (Binary Image Processing). Prentice Hall, 2000.

[8] Antero Taivalsaari. *JSR 139: Connected Limited Device Configuration 1.1*. http://jcp.org/en/jsr/detail?id=139.

[9] Eleanor Toye, Richard Sharp, Anil Madhavapeddy, e David Scott. Using smart phones to access site-specific services. *IEEE Pervasive Computing*, 4(2):60–66, aprile-giugno 2005.

[10] aa. vv. *Internation Symbology Specification - Annexe H, 255-State Algorithm*, p. 59. AIM Inc., 1996.

[11] aa. vv. *Internation Symbology Specification - Annexe M, ECC 200 Symbol Character Placement*, pp. 68–70. AIM Inc., 1996.

[12] aa. vv. *Internation Symbology Specification - Datamatrix*. AIM Inc., 1996.

[13] aa. vv. *Internation Symbology Specification - Datamatrix, ECC 200 Symbol Attributes*, p. 16. AIM Inc., 1996.

[14] Mark D. Weiser. http://en.wikipedia.org/wiki/Mark_Weiser, 1996.