

Università degli studi di Pisa
Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea Specialistica in Tecnologie Informatiche



Titolo della tesi di laurea:

Implementazione di un nuovo algoritmo di
fingerprint
matching e sua integrazione in ambiente smart card

Relatori:

Dott. Stefano Bistarelli

.....

Dott.ssa Anna Tonazzini

.....

Candidato:

Francesco Santini

.....

Anno accademico 2002/2003

Pisa 12/12/2003

Articolo 1

Tutti gli esseri umani nascono liberi ed eguali in dignità e diritti. Essi sono dotati di ragione di coscienza e devono agire gli uni verso gli altri in spirito di fratellanza.

Articolo 9

Nessun individuo potrà essere arbitrariamente arrestato, detenuto o esiliato.

Articolo 19

Ogni individuo ha il diritto alla libertà di opinione e di espressione, incluso il diritto di non essere molestato per la propria opinione e quello di cercare, ricevere e diffondere informazioni e idee attraverso ogni mezzo e senza riguardo a frontiere.

*Dichiarazione Universale dei Diritti dell'Uomo,
dall'Assemblea Generale delle Nazioni Unite il 10 Dicembre 1948*

Indice.

1	Introduzione	1
2	Le impronte digitali e il loro uso in applicazioni biometriche	9
2.1	Introduzione	9
2.2	Storia delle impronte digitali	10
2.3	Persistenza e individualità	13
2.4	Classificazione delle impronte	19
2.5	Acquisizione	25
2.5.1	Metodo di acquisizione <i>inked</i>	26
2.5.2	Metodo di acquisizione <i>live-scan</i>	28
2.6	Template e minuzie	31
3	Matching di impronte digitali	35
3.1	Sistemi automatizzati di identificazione di impronte digitali	35
3.2	Problemi relativi al fingerprint matching	40
3.3	Valutazione delle prestazioni dei sistemi biometrici	44
3.4	Template biometrici	49
3.5	Stato dell'arte	51

4	Smart card e Java Card™	59
4.1	Introduzione	59
4.2	Smart Card	60
4.2.1	Smart Card contactless	62
4.2.2	Super Smart Card	64
4.2.3	Smart Card contact	65
4.2.4	Architetture delle Smart Card	68
4.2.5	Standard ISO su Smart Card	71
4.3	Tecnologia Java Card™	73
4.4	Smart card e biometria	84
4.5	Introduzione a Java Card™ Forum	85
4.5.1	Caratteristiche delle API	88
5	Un nuovo algoritmo di fingerprint matching	93
5.1	Alcuni algoritmi esaminati	93
5.2	Descrizione dell'algoritmo	98
5.2.1	Teoria di base	98
6	Implementazione e prestazioni su PC	107
6.1	MINDTCT	107
6.2	Scelte implementative dell'algoritmo su PC	117
6.3	Caratteristiche di alcuni database di impronte	119
6.4	Test e prestazioni rilevate sul database utilizzato	127

7	Implementazione e prestazioni in ambiente Java Card™	139
7.1	Programmazione in Java Card™	139
7.1.1	Struttura della Application Protocol Data Unit (APDU)	140
7.1.2	Descrizione dell'Application Identifier (AID)	143
7.1.3	Toolkit Cyberflex di Schlumberger	144
7.1.4	Note di programmazione in ambiente Java Card™	149
7.2	Java Card™ Biometric API	152
7.3	Scelte implementative dell'algoritmo su Java Card™	166
8	Conclusioni	175
	Bibliografia	183
	Appendice A: biometria e applicazioni	195
A.1	Introduzione	195
A.2	Verifica basata sulla conoscenza	203
A.3	Verifica basata sul possesso	204
A.4	Verifica basata su dati biometrici	206
A.5	Pericoli per la sicurezza	209
A.6	Aree di applicazione	211
	Appendice B: modifiche al programma MINDTCT di NIST	215
	Appendice C: listato del codice sorgente dell'applicazione su PC	220

C.1	Package algMinInformation	227
C.2	Package algParsing	240
C.3	Package matching	254
C.3.1	Package graph	254
C.3.2	Package minCostAssigning	277
C.3.3	Package shortestPathTree	299
C.3.4	Classe CheckEdge	309
C.3.5	Classe Main Matcher	311
C.3.6	Classe MatcherException	322
C.4	Package minutiaeFileWriting	323
C.5	Package minutiaeInformation	328
C.6	Package minutiaeListConverting	346
C.7	Package minutiaeSuperClasses	367
C.8	Package parsing	386
C.9	Classe Main FinMatchTest	402
C.10	Classe Main FinalTest	410
C.11	Classe Main Farfr	416
Appendice D: listato del codice sorgente dell'applicazione Java Card™		421
D.1	Package converting	421
D.2	Package javaMatchOnCard	430
D.3	Classe Main EnrollMatch	475
D.4	Classe Main CardFinalTest	480
D.5	Classe Main FarfrCard	488

Elenco delle figure.

Figura 2.1	Esempio di impronta digitale [28]	13
Figura 2.2	Struttura della pelle. Mostra sia le impronte papillari, sia i pori del sudore [21]	15
Figura 2.3	Sezione trasversale del tessuto epidermico [21]	16
Figura 2.4	Esempio di punti di delta e core in un'impronta digitale [24]	21
Figura 2.5	Esempio di loop destro in a) e loop sinistro in b) [135]	21
Figura 2.6	Possibili formazioni di whorl in un'impronta digitale [43]	23
Figura 2.7	Distinzione tra central pocket loop in a) e plain loop in b) [43]	23
Figura 2.8	Esempi di plain arch in a) e tented arch in b) in un'impronta digitale [43]	24
Figura 2.9	Tecnologia a ultrasuoni per il rilevamento delle impronte digitali [61]	30
Figura 2.10	Esempi di ridge endings e ridge bifurcation [52]	32
Figura 2.11	Ridge bifurcation (cerchio) e ridge ending (quadrato) [51]	32
Figura 2.12	Questa figura mostra alcune tipologie di punti di singolarità. Partendo dalla prima colonna in alto a sinistra viene rappresentata una terminazione, una biforcazione, un punto, un'isola, un lago, un uncino, mentre sempre nello stesso	

	ordine, nella seconda colonna ci sono un ponte, una doppia biforcazione, una triforcazione, due biforcazioni opposte, un incrocio di creste ed una biforcazione contrapposta ad una terminazione [43]	33
Figura 3.1	Binarizzazione di un'impronta digitale [28]	37
Figura 3.2	Schema del funzionamento di un Automated Fingerprint Identification System (AFIS)	39
Figura 3.3	Esempi di piccole distorsioni in un quadrato [69]	42
Figura 3.4	Esempio di come una piccola distorsione della figura 3.3 possa portare a vistosi effetti se ripetuta [69]	42
Figura 3.5	Fattori che influiscono la qualità dell'immagine [51]	44
Figura 3.6	FFR, FAR e applicazioni biometriche nella curva di ROC	47
Figura 4.1	Alcuni esempi riguardanti l'uso di smart card: a) una carta di credito bancario, b) carta telefonica prepagata, c) un validatore di carte <i>contactless</i> per il trasporto urbano, d) kit per "costruirsi" in casa un accesso alla TV satellitare	60
Figura 4.2	Contatti esterni di una smart card	61
Figura 4.3	Trasmissione nella tecnologia <i>contactless</i> . L/S rappresenta il lettore/scrittore di carte	63
Figura 4.4	Super Smart Card	64
Figura 4.5	Contact card	66
Figura 4.6	Sezione di una smart card [37]	67
Figura 4.7	Struttura hardware di una smart card intelligente (con processore)	68
Figura 4.8	Bit trasmessi per byte di informazione nel protocollo	

	T=0 [37]	70
Figura 4.9	Passi per lo sviluppo di un applet Java Card™ [122]	77
Figura 4.10	Architettura di un'applicazione Java Card™ [124]	81
Figura 4.11	Interazione tra applet e JCRE nel ciclo di vita dell'applet [124]	83
Figura 4.12	Interazione tra diversi standard biometrici esistenti [95]	91
Figura 4.13	Architettura del Biometric Service Provider [120]	92
Figura 5.1	Esempio di riconoscimento dei caratteri [71]	94
Figura 5.2	Rappresentazione grafica delle quattro feature	100
Figura 5.3	Rappresentazione di una minuzia e di una suo vicino	100
Figura 5.4	Possibile ulteriore feature: area del poligono individuato dai vicini. Quella dentro il poligono è la minuzia centrale dei cinque vicini	105
Figura 6.1	Angoli trovati dal programma MINDTCT e che rappresentano la direzione di due minuzie di tipo “bifurcation” e “ridge ending” [46]	113
Figura 6.2	Tutte le minuzie trovate da MINDTCT su una impronta digitale con la direzione segnata da un segmento [45]	114
Figura 6.3	Due impronte diverse dello stesso dito con alcune minuzie rilevate dal programma MINDTCT: si possono notare come vengano rilevate anche delle “false” minuzie sul bordo dell'immagine tagliata; la loro affidabilità sarà comunque più bassa rispetto alle altre. Le immagini sono due acquisizioni diverse dello stesso dito: si può notare come le minuzie “false” possono non ripetersi	115
Figura 6.4	Diagramma del processo di rilevamento delle minuzie in	

	MINDTCT [45]	116
Figura 6.5	Due immagini dello stesso dito create con SFINGE [134]	126
Figura 6.6	Esempi di immagini di impronte digitali per ciascuno dei quattro database FVC2002 [67]	128
Figura 6.7	Grafo FAR-FRR ottenuto settando a 1000 il peso di <i>ed</i> e a 0 tutti gli altri. EER a 34,5% al valore soglia di 50.5	131
Figura 6.8	Grafo FAR-FRR ottenuto settando a 1000 il peso di <i>dra</i> e a 0 tutti gli altri. EER a 27,8% al valore soglia di 136.5	132
Figura 6.9	Grafo FAR-FRR ottenuto settando a 1000 il peso di <i>ra</i> e a 0 tutti gli altri. EER a 33,7% al valore soglia di 121.5	133
Figura 6.10	Grafo FAR-FRR ottenuto settando a 1000 il peso di <i>rc</i> e a 0 tutti gli altri EER a 39% al valore soglia di 55.5	134
Figura 6.11	Grafo FAR-FRR ottenuto sul DB1 con l'aiuto di tutte e quattro le feature	136
Figura 6.12	Receiving Operating Curve delle prestazioni dell'algoritmo	137
Figura 7.1	Formato delle Application Protocol Data Unit di comando e risposta [124]	140
Figura 7.2	Varie possibilità di costruzione della APDU di comando in base alla comunicazione del lettore/scrittore con la carta [124].	143
Figura 7.3	Struttura di un Application Identifier (AID) [122]	144
Figura 7.4	Passi necessari all'installazione di un programma su smart card con la Piattaforma Cyberflex [89]	146
Figura 7.5	Interfacce nelle Java Card™ Biometric API con relazione di ereditarietà [95]	153
Figura 7.6	Possibili schemi di implementazione di applet biometrici [95]	156

Figura A.1	Come aumentare la sicurezza di un sistema di identificazione usando tutte e tre le tipologie di autenticazione	198
Figura A.2	Alcuni strumenti inventati da Galton per misurare la vista	200

Elenco delle tabelle.

Tabella 2.1	Probabilità secondo modelli probabilistici differenti che due impronte di due individui diversi corrispondano per $N=12$ e $N=30$ minuzie [54]	17
Tabella 3.1	Alcune delle prestazioni di algoritmi rilevate dai siti web delle compagnie. Nessuna di esse specifica il database sul quale sono state effettuate le prove e cos le prestazioni non possono essere comparate in maniera equilibrata [135]	52
Tabella 3.2	Nella tabella sono riportati i risultati dei test di alcuni algoritmi di fingerprint matching effettuati su un database comune; maggiori informazioni sono disponibili su [65]. Nella prima colonna viene riportata la sigla dell'algoritmo in competizione corrispondente in generale alla sigla delle compagnia che lo presenta [135]	53
Tabella 6.1	Prestazioni dei primi 10 algoritmi rilevate durante la competizione FVC2002: FAR100 : la più bassa FRR per $FAR \leq 1\%$, FAR1000: la più bassa FRR per $FAR \leq 0.1\%$, ZeroFAR: la più bassa FRR per $FAR=0\%$ [67]	123

Tabella 6.2 Tabella riassuntiva delle caratteristiche dei quattro database di
test [70] 128

Tabella A.1 Comparazione tra le varie tecnologie biometriche 208

Capitolo 1

Introduzione.

Aeroporto di Tokyo, anno 2010. In mezzo alla folla, un bambino indica incuriosito la pistola di un agente. L'uomo in divisa la sgancia dalla cintura e gliela porge. Un incosciente? No. L'arma nelle mani del ragazzino non spara, non scatta, semplicemente non funziona se non viene impugnata dal poliziotto a cui è assegnata. Il dispositivo interno non riconosce le impronte digitali del piccolo.

Articolo di Francesca Tarissi su "L'Espresso" on line del 16/06/2003

A metà strada fra la scienza pura e la tecnologia applicata, la biometria si occupa di capire come alcune caratteristiche del corpo umano uniche per ciascun individuo possano essere utilizzate come strumento di riconoscimento personale.

La biometria, dunque, in primo luogo, esamina tutti questi dati fisionomici (legati alle caratteristiche anatomiche e di comportamento di un individuo) andando a evidenziare quelle particolarità, spesso minuscole, che ci rendono unici al mondo; questo

stimola la tecnologia a creare strumenti appropriati per far diventare tali peculiarità la "chiave" necessaria e indispensabile per aprire una porta, avviare la propria automobile, accedere a un computer, utilizzare i servizi del bancomat o di altre reti telematiche.

Storicamente la biometria trova spunto dallo studio delle impronte digitali ai fini del riconoscimento delle persone, ma è con l'avvento dell'elettronica e, soprattutto, con l'esigenza di nuovi strumenti a servizio della sicurezza "esplosa" dopo l'attentato alle Torri Gemelle, che queste tecnologie sono venute alla ribalta.

Tra l'altro il riconoscimento biometrico non è certo una novità: in fondo una normale carta d'identità è un efficiente strumento di riconoscimento biometrico: grazie alla fotografia ed alle indicazioni di età, altezza, colore degli occhi e dei capelli possiamo identificare con una ragionevole sicurezza il suo proprietario.

Uno dei principali fattori trainanti della biometria è la capacità di offrire un'alternativa praticabile alla password universale, forma di riconoscimento che ha ampiamente dimostrato la propria debolezza. Sempre più persone si rendono conto che la biometria presenta vantaggi non solo per le applicazioni ad alto livello di sicurezza. La semplicità di utilizzo ("la chiave è sempre con te") rende questi sistemi molto attraenti anche per altre applicazioni. Sappiamo di varie industrie e organizzazioni di servizi che hanno introdotto la biometria per controllare l'accesso non solo dei propri dipendenti ma anche di clienti e visitatori. L'autenticazione tramite dispositivi biometrici è per sua molto sicura: dettagli fisiologici, quali l'impronta digitale o la retina, variano in modo quasi¹ univoco da individuo a individuo a meno di percentuali piccolissime. Al risparmio in termini di rischio, è da aggiungere a favore della tecnologia biometrica la naturale fragilità che un sistema basato su password racchiude in sé: gli utenti di un sistema

¹ Tuttavia molti sono gli esempi che hanno dimostrato che a tutt'oggi i dispositivi di acquisizione e riconoscimento non sono ancora sufficientemente affidabili. Non sarà parte di questa tesi la valutazione sull'affidabilità dei dispositivi utilizzati per l'acquisizione delle informazioni biometriche.

informativo tendono ad utilizzare con poca attenzione la password, a divulgarla accidentalmente o molto spesso a dimenticarla. Ma la vera freccia all'arco della autenticazione biometrica, come già accennato precedentemente, è la semplicità d'uso: non c'è nulla di più naturale che premere il dito indice su una superficie al fine permettere al dispositivo di leggere l'impronta digitale, o addirittura restare fermi quell'istante necessario ad un dispositivo ottico per ricavare le informazioni relative alla retina.

Come tutte le innovazioni, la biometria ha bisogno di tempo per essere assimilata e compresa da un pubblico che attualmente è ancora molto scettico nei suoi riguardi. È in forte espansione il mercato delle tecnologie biometriche per il riconoscimento individuale: il trend di crescita è esponenziale. Secondo l'*International Biometric Group* [97], società che offre consulenza e fornisce servizi in questo settore, il giro d'affari passerà dagli attuali 600 milioni di dollari a oltre 4 miliardi entro il 2007. Un sondaggio sponsorizzato dal Servizio di Statistica Giudiziaria degli Stati Uniti afferma che, pur con le dovute precauzioni prese a tutela della privacy, i cittadini sono favorevoli all'introduzione di queste tecnologie di controllo.

Naturalmente esistono ancora molti problemi relativi a queste tecniche: come sottolineato da Simson Garfinkel in [141], "purtroppo c'è un difetto fondamentale: queste metodiche non identificano persone, bensì corpi". Falsificare le informazioni biometriche è molto difficile, ma in linea di principio è possibile. Del resto è già avvenuto che, attraverso impronte digitali modellate in diversi materiali, si sia tentato di ingannare un sistema di riconoscimento. Due hacker tedeschi infatti sono riusciti a dimostrare che si può creare un falso dito in grado di ingannare qualsiasi tipo di scanner: creando cioè una sottile pellicola in lattice che può essere apposta su un vero dito. Questa tecnica, essendo minimo lo spessore del lattice, consente di ingannare anche gli scanner

capacitivi, dotati di elettrodi che dovrebbero rilevare se l'impronta digitale appartiene a un dito umano o artificiale.

Nel lavoro effettuato per questa tesi di laurea ci siamo dedicati alla ricerca di un nuovo algoritmo di verifica di impronte digitali (fingerprint matching) abbastanza semplice e veloce da poter essere utilizzato anche su una smart card, che rappresenta un supporto in forte espansione, ma ancora caratterizzato da limitate risorse di calcolo. Più precisamente, dopo un primo periodo di documentazione sulla tecnologia biometrica scelta e sui metodi proposti dalla letteratura in questo settore, l'algoritmo sviluppato è stato implementato in linguaggio Java ed in seguito ne sono state migliorate le prestazioni con l'aiuto di numerosi test ed di uno studio sui relativi grafi. È seguito poi il porting del codice su piattaforma Java Card™ [107, 110] (Java su smart card) per realizzare il *match on card*: in pratica sia l'algoritmo di verifica, sia i dati relativi al possessore della smart card (una rappresentazione dell'impronta digitale di un proprio dito) si trovano all'interno della carta e quindi non possono essere manomessi per violarne la sicurezza; soprattutto, i dati personali non lasciano mai il supporto e quindi non attraversano mai nessun canale di comunicazione, impedendo così l'intercettazione da parte di un possibile "man on the middle" e preservando la sicurezza del metodo di verifica. In un semplice e generale scenario di utilizzo, il portatore della smart card inserisce la sua carta nel lettore, preme il dito su uno scanner che elabora l'immagine, ne estrae le informazioni e infine le invia alla carta dove avviene il match con le informazioni dello stesso dito memorizzate al momento del rilascio della smart card; se le due impronte matchano, allora la carta avverte il lettore che il match è avvenuto positivamente e quindi può lasciare che il portatore acceda ad un determinato servizio (per esempio autorizzano l'accesso ad una chiave privata per una firma digitale).

Nel seguito descriviamo brevemente il contenuto dei capitoli della tesi. Nel capitolo 2 ci occupiamo di descrivere la tecnologia biometrica da noi scelta e cioè quella basata sulle impronte digitali: riportiamo quindi la storia del loro uso, i possibili metodi di acquisizioni presenti al giorno d'oggi, la classificazione in alcune grandi famiglie di impronte ed infine la descrizione di alcuni punti di singolarità (le minuzie) che le rendono uniche e che costituiscono l'informazione utilizzata dalla maggior parte degli algoritmi di verifica, compreso il nostro.

Nel capitolo 3 passiamo a trattare i sistemi automatizzati di identificazione di impronte digitali, elenchiamo e approfondiamo i parametri necessari per valutare le prestazioni (per esempio in termini di affidabilità) di tali sistemi, presentiamo i principali problemi legati alla verifica e ne descriviamo le cause. Infine esaminiamo lo stato dell'arte in questo settore portando come esempio alcuni algoritmi di riferimento.

Il capitolo 4 rappresenta l'ultimo dei capitoli di background e presenta quindi la tecnologia smart card ed in particolare Java Card™: nella smart card è presente una Java Virtual Machine molto ridotta, ma in grado di interpretare il bytecode ottenuto da un sottoinsieme del linguaggio Java opportunatamente trasformato per tale piattaforma. Sono descritte quindi le principali caratteristiche fisiche di tale supporto, le principali tipologie e le loro architetture, passando poi ad una relazione sulle componenti e sul funzionamento di Java Card™ (Virtual Machine, API, ecc...). Infine è presente un paragrafo che delinea l'applicazione della biometria al mondo delle smart card con una descrizione delle possibili funzionalità implementate sulla carta, partendo dalla più semplice *template on card* (TOC), per arrivare poi crescendo nel numero di funzionalità a *match on card* (MOC) e *system on card* (SOC). L'ultimo paragrafo del capitolo contiene un'introduzione sull'organizzazione Java Card™ Forum [88] e sulle API

biometriche da essa rilasciate per fornire un'interfaccia standard tra applet (programmi in Java Card™) e algoritmi di riconoscimento biometrico.

Nel capitolo 5 viene presentato il nuovo algoritmo da noi progettato, con una trattazione completa che parte dagli algoritmi già esistenti che ci hanno fornito alcuni suggerimenti basilari, e continua con la descrizione delle sue procedure, tratte essenzialmente dalla ricerca biometrica, e della sua complessità.

Nel capitolo 6 ci siamo occupati dell'implementazione su PC dell'algoritmo descritto nel capitolo 5: nel primo paragrafo parliamo del programma MINDTCT [45] del *National Institute of Standards and Technology* (NIST) e facente parte del *Nist Fingerprint Image Software* (NFIS), un insieme di strumenti open source per il trattamento delle impronte digitali e che abbiamo utilizzato per estrarre la lista delle minuzie dall'immagine dell'impronta; in seguito, oltre alle caratteristiche del database di impronte utilizzato per eseguire i nostri test, presentiamo alcune scelte effettuate per l'implementazione dell'algoritmo ed infine concludiamo con una descrizione delle performance ottenute e corredate da alcuni grafici.

Nel capitolo 7 abbiamo descritto la seconda implementazione effettuata e rappresentata dal porting del codice dall'ambiente Java a quello Java Card™. Nel primo paragrafo sono presentate alcune problematiche legate allo sviluppo sotto questo ambiente, partendo dal formato dei messaggi (APDU) scambiati tra lettore/scrittore e smart card, passando dall'identificatore unico dell'applicazione (AID), dal toolkit Cyberflex [89] di Schlumberger utilizzato per lo sviluppo e terminando con alcune note che devono essere necessariamente conosciute per la programmazione sotto questo ambiente. Nel secondo paragrafo dello stesso capitolo vengono approfondite le API biometriche già introdotte nel capitolo 3, compreso alcuni dettagli su come sono state

implementate nella nostra applicazione. Infine il terzo paragrafo è costituito da alcune linee seguite per adattare il nostro algoritmo all'ambiente Java Card™.

Nel capitolo 8 sono riportate infine alcune importanti conclusioni sul lavoro svolto, mettendo anche in evidenza alcuni possibili sviluppi futuri volti ad affinare le prestazioni ottenute.

L'ordine dei capitoli segue a grandi linee anche la cronologia delle tappe del lavoro svolto.

Sono presenti anche quattro appendici al testo principale della tesi: nell'appendice A viene descritta cos'è la biometria nei suoi aspetti generali e la sua collocazione rispetto agli altri sistemi di verifica come quelli basati sulla conoscenza (PIN) o sul possesso (una comune carta a banda magnetica); vengono descritti anche i pericoli per la sicurezza dei sistemi di verifica e alcune delle più importanti aree di applicazione di questi sistemi con particolare riferimento a quelli biometrici. In appendice B sono invece riportate le modifiche da noi apportate al codice del programma MINDTCT presentato nel corso del capitolo 6: abbiamo infatti dovuto adattarlo ad alcune nostre particolari esigenze per ottenere l'informazione necessaria come input dell'algoritmo di verifica. In appendice C è riportato il codice Java dell'implementazione del nostro algoritmo di fingerprint matching, mentre in appendice D viene mostrato il codice del porting dello stesso algoritmo in ambiente Java Card™.

Capitolo 2

Le impronte digitali e il loro uso in applicazioni biometriche.

2.1 Introduzione.

Le impronte digitali sono comunemente definite come l'insieme di creste e valli rilevate sui polpastrelli [34], solitamente sull'ultima falange e sono utilizzate da molto tempo ed estensivamente per l'identificazione delle persone [35]. La figura 2.1 ne mostra un esempio.

Le proprietà biologiche della formazione delle impronte sono state ben comprese sotto la spinta del loro uso sin dall'inizio del ventesimo secolo da parte delle polizie scientifiche di diversi stati [36].

Proprio a causa di questo uso originale legato alla criminologia comunque, molte persone si mostrano restie a concederle per l'utilizzo in applicazioni civili, anche se i sistemi biometrici basati su di esse offrono un'identificazione positiva con un alto grado di confidenza ed allo stato attuale, sono basati su una tecnologia a basso costo e, appunto, ben studiata. La disponibilità di scanner a basso prezzo [63] e di algoritmi di

matching molto robusti, ha difatti influito positivamente sulla loro popolarità e utilizzo negli ultimi anni. L'uso di tecniche biometriche basate su queste impronte presenta però anche alcuni problemi di rilevante importanza: per esempio il 4% della popolazione non può fornire impronte di buona qualità, che quindi pregiudicano le prestazioni del riconoscimento.

Alcuni esempi di questi problemi sono forniti soprattutto da lavoratori che utilizzano intensivamente le mani e che quindi possono presentare temporanei, ma frequenti ferite sui polpastrelli, o anche pieghe cutanee dovute all'immersione in acqua per un periodo prolungato, dovute al naturale invecchiamento, spellature dovute a malattie, ma anche dita sporche che possono non essere correttamente analizzate dai sensori.

Inoltre, al contrario del riconoscimento dei tratti del volto, le impronte non possono essere catturate senza la consapevolezza dell'utente e quindi non si adattano ad applicazioni di sorveglianza.

2.2 Storia delle impronte digitali.

Lo studio delle impronte digitali, che va sotto il nome di *dattiloscopia*, affonda le sue radici in un passato abbastanza remoto: le moderne tecniche si sono evolute da studi compiuti per la prima volta alla fine del 17esimo secolo dopo Cristo: il documento redatto dal botanico e fisico inglese Nehemiah Grew [48] e riguardante uno studio sulla struttura delle creste e dei pori è ritenuto infatti il primo scritto scientifico sulle impronte digitali. In precedenza, solo Marcello Malpighi, professore di anatomia all'Università di

Barcellona, aveva esaminato le impronte al microscopio. Successivamente, nel 1788, J.C.A. Mayer individuò e descrisse alcune caratteristiche ricorrenti delle impronte papillari [53]: *"Although the arrangement of skin ridges is never duplicated in two persons, nevertheless, the similarities are closer among some individuals. In others, the differences are marked, yet in spite of their peculiarities of arrangement, all have a certain likeness"*.

Il primo uso dell'impronta come firma personale si deve all'intagliatore Thomas Bewick, che dal 1809 cominciò appunto a firmare le proprie opere con la propria impronta [34]: in realtà sono state trovate tavolette Babilonesi risalenti al 500 A.C. (e contemporaneamente anche in Cina) riguardanti transazioni commerciali e recanti impronte impresse sopra e questo viene in realtà considerato come una delle più importanti tappe della moderna dattiloscopia.

La prima classificazione delle impronte, in nove categorie basate sulla struttura generale delle creste, arriva nel 1823 con John Evangelist Purkinji, professore di anatomia all'Università di Breslau in Germania [34]; allo stesso tempo però questo studio non aveva evidenziato le potenzialità delle impronte nel riconoscimento delle persone.

Nel 1880 l'inglese Henry Fauld, dottore ed evangelista in Giappone, suggerì in un articolo sulla rivista scientifica *Nature*, l'individualità delle impronte digitali ed un loro possibile utilizzo nell'identificazione dei criminali. Quasi contemporaneamente William Herschel annunciò di averle utilizzate per più di 20 anni in India per fini investigativi [34].

Le ultime due scoperte nel ramo furono apportate da Sir Francis Galton, che introdusse la struttura di minuzia nel 1888 e Sir Edward Henry, che nel 1899 realizzò un

sistema di classificazione che semplificò molto l'identificazione che al tempo era chiaramente ancora realizzata manualmente da esperti [34, 36].

Si può quindi ritenere che già nei primi anni del 20esimo secolo, la formazione e i principi generali alla base delle impronte digitali e della loro verifica fossero ben compresi a tal punto da un loro primo utilizzo, come difatti avvenne, nei tribunali di giustizia sparsi in diversi paesi.

Dopo decine di anni di uso in ambito legale senza dubbi rilevanti, le testimonianze riguardanti impronte digitali da parte di esperti all'interno dei processi, sono state messe in discussione nel 1993, col caso *Daubert contro Merrell Dow Pharmaceuticals, Inc.* [58] (conosciuta nell'ambiente come *Daubert Hearing*): la Suprema Corte degli Stati Uniti decretò infatti che deve venir valutata l'affidabilità di una testimonianza scientifica di questo tipo. Più precisamente la corte asserì che nello stabilire tale affidabilità, dovevano essere considerati i seguenti punti.

- ◇ Se la particolare tecnica o metodologia in questione può essere valutata dal punto di vista statistico.
- ◇ Se è stata valutata la probabilità di errore di tale metodologia.
- ◇ Se esistono degli standard che “controllano” le operazioni effettuate e, qualora esistano, se questi standard sono stati mantenuti nel caso specifico.
- ◇ Se la metodologia sia stata precedentemente pubblicata e esaminata da un secondo esperto.
- ◇ Se la tecnica sia stata ampiamente accettata.

Negli anni successivi, vista questa restrizione applicata all'uso delle impronte digitali, in molti casi giudiziari venne richiamato il caso *Daubert* anche per la valutazione

nel riconoscimento calligrafico, ma molte corti sono ormai arrivate alla conclusione che i criteri di Daubert non sono applicabili in questo campo. Per quanto riguarda l'identificazione attraverso le impronte papillari invece, i criteri di Daubert sono stati richiamati 40 volte (al 26 Agosto 2002), la prima nel 1999 nel caso degli Stati Uniti contro Byron Mitchell [59], ma la testimonianza di tre esperti nominati dalla difesa che asseriva la non scientificità della prova dell'impronta digitale, fu negata dal giudice Curtis Joyner con la motivazione che, citando liberamente, le creste papillari umane sono uniche e permanenti, comprese quelli visibili in una porzione ristretta dell'impronta [60].



Figura 2.1
Esempio di impronta digitale [28].

2.3 Persistenza e individualità.

L'identificazione attraverso l'utilizzo delle impronte digitali è basata su due basilari premesse:

- ◇ Persistenza: le caratteristiche delle impronte non cambiano attraverso il tempo.
- ◇ Individualità: l'impronta è unica da individuo a individuo.

La prima delle due premesse è stata verificata dall'anatomia e dalla morfogenesi nella frizione delle creste dell'impronta. La pelle è infatti riconosciuta come un organo a tutti gli effetti, dato che è costituita da diversi tipi di tessuto differenti e fa parte del sistema tegumentario, insieme ai vari derivati come le unghie, i capelli e le ghiandole sebacee e sudoripare. La funzione principale è quella di assicurare protezione all'organismo e permettere i rapporti con il mondo esterno. Mentre nella maggior parte del corpo la cute si presenta sostanzialmente "liscia", sulle dita, sui palmi delle mani e sulla pianta del piede si presentano le cosiddette impronte papillari o, meglio, creste papillari.

Per trattare brevemente l'aspetto dal punto di vista biologico, la pelle è costruita da tre tipi di tessuto, epidermide e derma, che insieme formano la cute e l'ipoderma che si trova più in profondità.

Mentre l'epidermide costituisce lo strato più superficiale (vedi figura 2.1 e 2.2), il derma si trova immediatamente sotto a questa e quindi ha con lei intimi rapporti, perché la sostiene, la nutre ed offre sede alle appendici epidermiche, cioè le ghiandole e i peli. Il derma è spesso circa 2-3 mm. ed è costituito da 2 parti: papillare e reticolare; mentre la prima, che è costituita dalle papille e dallo strato subpapillare ha una vita metabolica molto attiva per la sua vicinanza all'epidermide, la seconda può essere considerata come uno stroma di sostegno. Nel derma sono presenti i vasi sanguigni, le innervazioni e gli annessi cutanei, cioè le ghiandole sudoripare, i follicoli piliferi, i peli, le ghiandole sebacee e il muscolo del pelo; in esso è presente anche un fitto intreccio di vasi linfatici

che si dirigono verso il sottocutaneo o ipoderma. L'epidermide e il derma sono uniti tramite le papille dermiche, cioè dei prolungamenti conici di tessuto connettivo che dal derma si estendono a compenetrare l'epidermide. I capillari sanguigni si portano fino all'apice delle papille e costituiscono la fonte di nutrimento per l'epidermide che non è vascularizzata. Il disegno superficiale della cute è in rapporto al variare della disposizione e dello spessore delle fibre connettive del derma e questo dà origine ad una precisa disposizione papillare. Questo disegno è appunto così tipico che viene usato per l'identificazione di un individuo con l'impronta digitale [21].

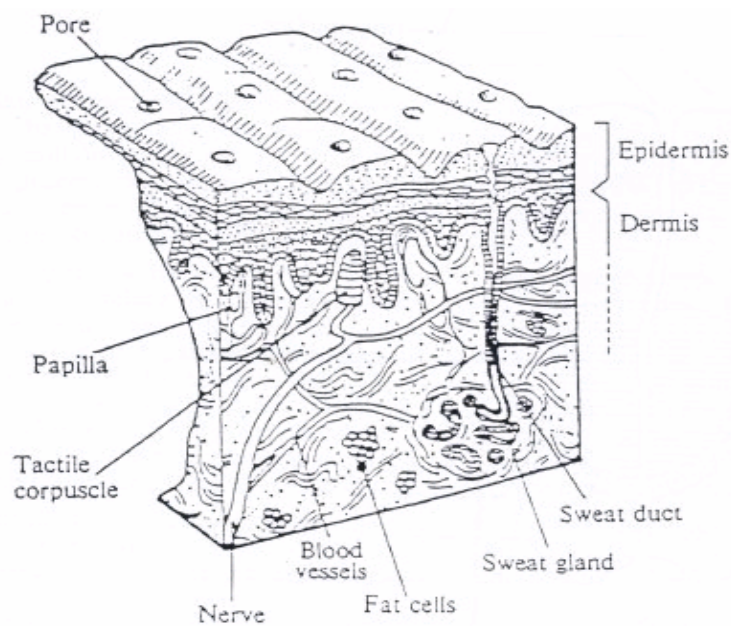


Figura 2.2
Struttura della pelle.
Mostra sia le impronte papillari, sia i pori del sudore [21].

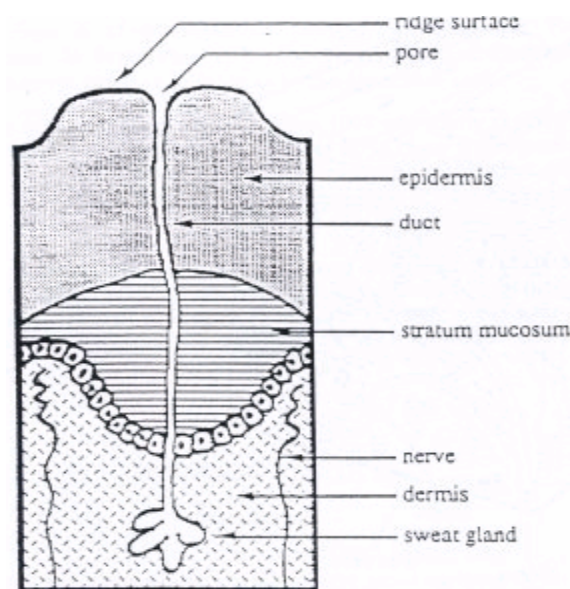


Figura 2.3
Sezione trasversale del tessuto epidermico [21].

A meno di traumi o di interventi, la prima caratteristica di persistenza è assicurata quindi dalla ricrescita dello strato di pelle morta con le stesse esatte caratteristiche.

Le impronte digitali si formano definitivamente nel feto di 7 mesi e sono in generale, insieme all'aspetto fisico, facenti parte del fenotipo di un individuo, che si ritiene sia univocamente determinato dalla combinazione di uno specifico genotipo con uno specifico ambiente [55]. Ecco così che la loro formazione sia simile a quella dei vasi sanguigni nell'angiogenesi; le caratteristiche generali cominciano ad emergere con la definizione della pelle sui polpastrelli, ma allo stesso tempo la posizione del feto nell'utero e i flussi del liquido amniotico cambiano durante questo processo di formazione rendendolo unico.

Questo micro ambiente varia da mano a mano e da dito a dito: esistono quindi molti fattori che variano durante lo sviluppo, anche se esiste un patrimonio genetico che

lo ha fortemente influenzato. Si può definire allora la formazione di impronte papillari come un processo caotico e non un processo casuale [55].

Per quanto riguarda quindi la seconda premessa, viene ritenuta essere vera sulla base di risultati empirici, ma non ne è stata scientificamente dimostrata la validità, che non solo porterebbe alla definitiva accettabilità dell'impronta digitale all'interno dei tribunali di giustizia [56, 57], ma stabilirebbe anche un limite superiore per le performance dei sistemi di riconoscimento automatico: il riconoscimento terminerebbe al primo risultato positivo.

Molti modelli probabilistici sono stati studiati per definire la probabilità che due dita di due individui diversi corrispondano [54], come riportato nella tabella 2.1 (sempre tratta da [54]).

Author	N=12	N=36
Galton (1892)	9.54×10^{-7}	1.45×10^{-11}
Pearson (1930)	8.65×10^{-17}	1.09×10^{-41}
Henry (1900)	3.72×10^{-9}	1.32×10^{-23}
Balthazard (1911)	5.96×10^{-8}	2.12×10^{-22}
Bose (1917)	5.96×10^{-8}	2.12×10^{-22}
Wentworth & Wilder (1918)	4.10×10^{-21}	6.87×10^{-62}
Cummins & Midlo (1943)	1.32×10^{-22}	2.22×10^{-63}
Gupta (1968)	1.00×10^{-14}	1.00×10^{-38}
Roxburgh (1933)	3.35×10^{-18}	3.75×10^{-47}
Trauring (1963)	2.91×10^{-9}	2.47×10^{-26}
Osterburg et al. (1980)	3.05×10^{-15}	1.33×10^{-27}
Stoney (1985)	3.5×10^{-26}	1.2×10^{-89}

Tabella 2.1
Probabilità secondo modelli probabilistici differenti che due impronte di due individui diversi corrispondano per N = 12 e N= 30 minuzie [54].

L'unicità di un'impronta digitale è un'ipotesi di lavoro che in senso matematico è difficile (se non impossibile) da provare; la dimostrazione opposta è sicuramente più facile da ottenere in teoria, trovando nella pratica due impronte identiche di due dita diverse, anche se ad oggi questo evento non è ancora verificato. Ecco quindi che in ambito scientifico il termine di univocità è stato sostituito dalla probabilità di trovare due impronte identiche provenienti da individui differenti ed in questo campo possono essere utili test su database forense di grandi dimensioni comparando ciascuna impronta con tutte le altre: se tale "collezione" raccogli un insieme di 100 milioni di dita diverse, possono essere controllate le probabilità fino a 10^{-14} . Per quanto riguarda quest'ultimo argomento, un'analisi scientifica sull'individualità delle impronte è stata condotta in [54].

Quindi riassumendo, i principi biologici possono essere velocemente riassunti in tre punti fondamentali:

- ◇ Creste e valli hanno caratteristiche epidermiche diverse per dita differenti.
- ◇ I vari tipi di configurazioni delle strutture papillari sono variabili solo all'interno di limiti ben individuati, che consentono lo sviluppo di una classificazione sistematica.
- ◇ Le impronte papillari sono invariabili per un dato polpastrello.

2.4 Classificazione delle impronte.

Il primo sistema di classificazione delle impronte fu ideato da Sir Edward Henry in India nei primi anni del ventesimo secolo espandendo un lavoro precedente di Sir Francis Galton. Già agli albori della loro utilizzazione infatti, appariva necessario un sistema di suddivisione delle impronte in classi per rendere più veloce il processo di comparazione: per dare un'idea, nel 1924 la *Identification Division* del Federal Bureau of Investigation (FBI) contava nei suoi archivi più di 800 mila schede con 10 impronte ciascuna [38]; oggi il totale è salito a più di 630 milioni, ma risulta difficile mantenere il dato aggiornato, visto che la polizia federale ne riceve mediamente 5 mila nuove al giorno per criminali al primo arresto [39].

Per quanto riguarda la classificazione possono risultare utili le definizioni di due punti “speciali” all'interno dell'impronta, le cui definizioni sono qui tratte dal vocabolario redatto da *Scientific Working Group on Friction Ridge Analysis, Study and Technology*, o più brevemente SWGFAST [40]; molte delle definizioni ivi contenute sono tratte dalla classificazione utilizzata dall'FBI, a sua volta basata su quella di Henry:

- ◇ Punto di *core*: approssimativamente il centro di uno schema di creste. Di punti di core ce ne può essere al massimo uno. Nella pratica consiste nel punto più estremo della curvatura della cresta più interna che forma la spirale. Le creste di un loop sono tutte a forma di una “u” (vedi figura 2.4).
- ◇ Punto di *delta*: definito come il punto su una cresta e coincidente o il più vicino possibile ad un punto di divergenza: quest'ultimo individua un punto in cui due creste che correvano pressoché parallele, si dividono. Il delta viene definito anche

come il punto al centro di una regione triangolare, che si trova normalmente in basso a destra o sinistra, nella quale confluiscono creste da direzioni diverse [41] (vedi figura 2.4).

Va detto però che questi punti singolari non si manifestano in tutti i casi e anche quando sono presenti possono venire eliminati o corrotti dal processo di acquisizione dell'immagine.

Le principali tre categorie, le quali si suddividono a loro volta in altre, sono tre: *Loop*, *Whorl* e *Arch*.

Loop:

I loop costituiscono la configurazione più comunemente osservabile tra gli schemi delle impronte digitali, con una leggera predominanza statistica da attribuire ai loop destri: 36,46 per cento contro 17,03 per cento [42]. Sono caratterizzati da più creste che attraversano la linea immaginaria tra punto di core e punto di delta e formando una “u”, ritornano approssimativamente nella direzione da cui sono venute; i loop sono infatti caratterizzati dall'avere esattamente un punto di loop e uno di core, come mostrato in figura 2.4 (tratta da[28]).

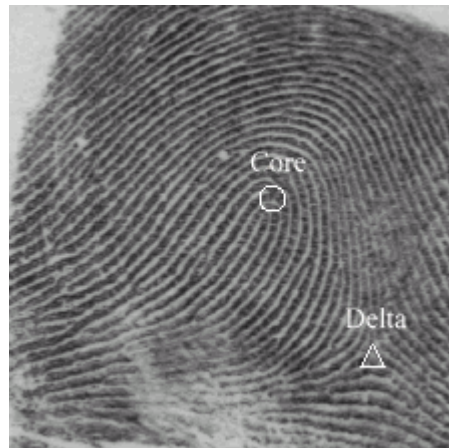


Figura 2.4
Esempio di punti di delta e core in un'impronta digitale [28].

I *pattern* di loop si possono a loro volta suddividere, come accennato precedentemente, in destri e sinistri, o più correttamente in *ulnar loop* e *radial loop* se si conosce anche la mano di cui fanno parte. Nel loop sinistro, le creste che lo formano provengono e ritornano



a) Right loop.

b) Left loop.

Figura 2.5
Esempio di loop destro in a) e loop sinistro in b) [135].

verso la direzione sinistra e quindi, se il dito a cui appartengono si trova nella mano destra, tendono verso la direzione del pollice e quindi verso l'osso del radio, mentre se appartenessero alla mano sinistra, si affaccerebbero verso il mignolo e quindi verso l'ulna. Se non si conosce la mano è comunque sufficiente osservare l'impronta per stabilire semplicemente se il loop è destro o sinistro (vedi esempio di figura 2.5 tratta da [135]).

Per quanto riguarda i radial loop, rappresentano quelli meno comuni e si trovano solitamente sul dito indice.

Whorl:

Questa classe è la seconda per frequenza di apparizione, costituendo più del 30 per cento del totale e per definizione contengono due o più punti di delta. Il pattern presenta una parte di creste che tendono a formare un cerchio. Come i loop, anche i whorl si possono suddividere in ulteriori sottoclassi [43] osservabili in figura 2.6 (sempre tratta da [43]):

- ◇ *Plain loop.*
- ◇ *Central pocket.*
- ◇ *Double loop.*
- ◇ *Accidental.*

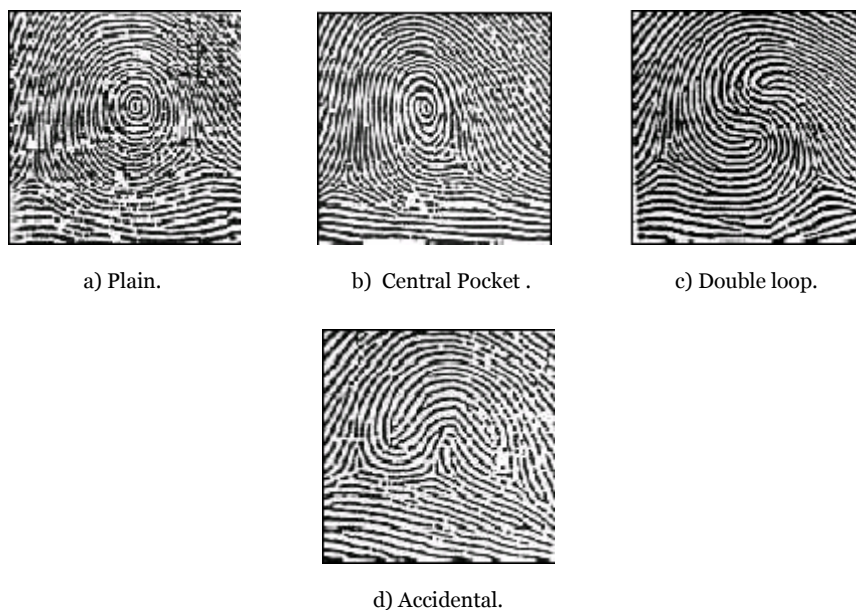


Figura 2.6
Possibili formazioni di whorl in un'impronta digitale [43].

Questa sottoclassificazione è molto facile da apprendere: se l'immagine contiene più di due delta allora è un *accidental loop*, se invece vi sono due delta allora ricadono negli altri tre gruppi. Per quanto riguarda i primi due, tracciando un'ideale linea tra i due punti di delta, si osserva se questa interseca l'area delle creste che formano un cerchio: se la risposta è

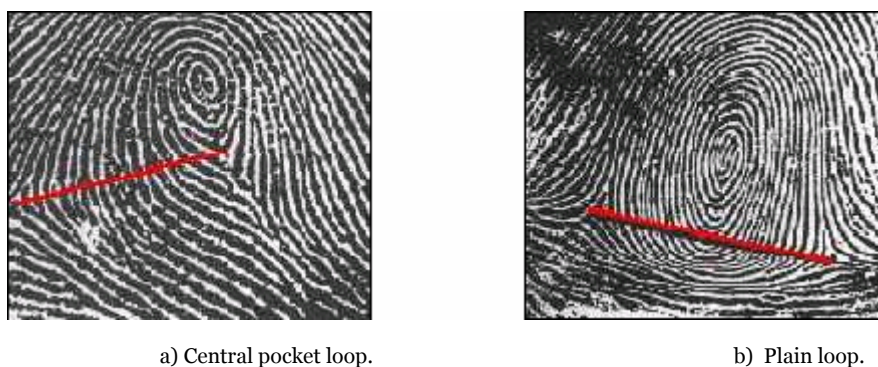


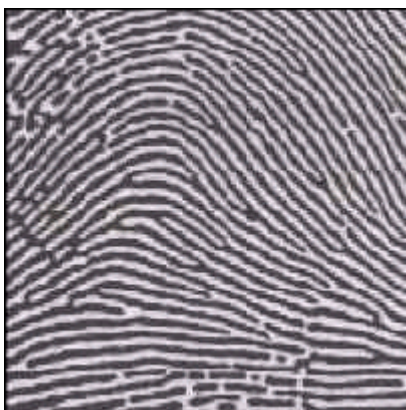
Figura 2.7
Distinzione tra central pocket loop in a) e plain loop in b) [43].

positiva allora si tratta di un *central pocket loop*, se negativa, di un *plain loop* (figura 2.7 tratta da [43]).

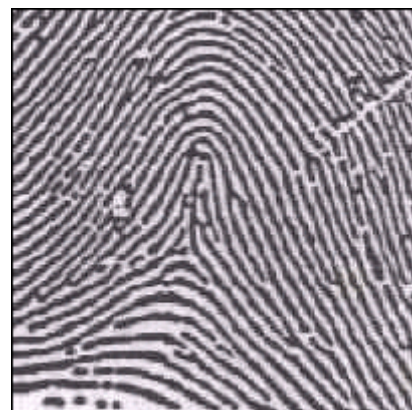
Per quanto riguarda invece il *double loop whorl*, la zona circolare è formata da due completi e distinti loop che “girano” uno dentro l’altro.

Arches:

Plain arch e *tented arch* formano rispettivamente intorno al 6 e 8 per cento dei pattern delle impronte digitali. In questi schemi le creste corrono da parte a parte dell’immagine senza formare curve a “U”; sono caratterizzate dall’assenza di punti di delta, ma qual’ora ve ne fosse una parvenza basta controllare che nessuna cresta “ricurvante” intersechi il segmento tra delta e core. Come si può notare dalla figura 2.8 (tratta da [43]), mentre nel *plain arch* le creste tendono a fluire attraverso l’immagine con curvature molto smussate, mentre in *tented arch* non scorrono così “tranquillamente” come nel primo caso, ma sono presenti curvature significativamente strette che conferiscono all’immagine la forma di una tenda.



a) Plain arch .



b) Tented arch.

Figura 2.8

Esempi di plain arch in a) e tented arch in b) in un’impronta digitale [43].

Come si può controllare sempre dalla figura 2.8, la zona a forma di delta presente negli *arch* in generale, non individua un punto di delta, dato che è formata dalle stesse creste ricurvanti: in un loop tale punto è creato invece da due flussi di creste ben distinti uno dall'altro.

Le cinque classi qui menzionate *whorl*, *right loop*, *left loop*, *arch* (o *plain arch*), e *tented arch*, vengono utilizzate dal *National Institute of Standards and Technology* (NIST) [44], come benchmark per i propri sistemi di classificazione automatica, dei quali rientra per esempio il programma *Pcasys*, un prototipo dimostrativo di cui è disponibile anche il codice sorgente (pubblico dominio) e basato su reti neurali in uno dei passi di computazione che esegue [45]. *Pcasys*, del quale la prima versione pubblica risale al 1995, riesce a classificare anche in una sesta classe non menzionata fino ad adesso e cioè quella di *scar*: in pratica riconosce impronte di dita con notevoli cicatrici sui polpastrelli, anche se in realtà comunque ricadrebbe in una delle altre classi “standard”.

2.5 Acquisizione.

Le immagini riguardanti le impronte digitali possono essere raccolte sia con processi di acquisizione *offline*, sia *online*; i primi sono conosciuti anche come *inked*, mentre i secondi come *live-scan*.

2.5.1 Metodo di acquisizione *inked*.

Nei metodi *inked* solitamente un addetto è incaricato della rilevazione dell'impronta tramite la pressione del dito bagnato di inchiostro dell'utente su un foglio di carta, successivamente acquisito digitalmente attraverso la scannerizzazione con un normale scanner per documenti. La rilevazione dell'impronta può essere effettuata direttamente anche dal solo utente attraverso dei kit specifici spediti a casa dall'azienda a cui viene chiesto un servizio: un esempio è fornito da MasterCard per l'uso sicuro delle carte di credito [49].

A seconda del metodo di rilevazione, le immagini *inked* si dividono in tre diverse sotto classi:

- ◇ *Rolled*: in questo caso l'inchiostro viene applicato al polpastrello e poi il dito viene impresso su un foglio di carta e quindi mosso con un movimento ondulatorio da sinistra verso destra e viceversa, per cercare di trasporre sul documento più impronta possibile. Allo stesso tempo però, questo movimento può provocare deformazioni sulle strutture delle creste molto evidenti e quindi pregiudicare una successiva comparazione con impronte diverse dello stesso dito.
- ◇ *Dab*: in questo caso il dito cosperso di inchiostro viene impresso semplicemente senza ulteriori movimenti; pregi e difetti si scambieranno quindi con il metodo precedente: meno impronta e meno deformazioni.
- ◇ *Latent*: in questo caso, usato dalle polizie scientifiche per l'acquisizione di informazioni sulla scena del crimine, l'impronta viene rilevata dagli oggetti toccati dal "possibile" criminale, e quindi ci troviamo nella mancanza diretta del

dito del quale vogliamo avere l'impronta, a differenza dei primi due casi. Le impronte lasciate dal grasso possono essere quindi molto disturbate, parziali e la qualità può essere pregiudicata dal materiale sul quale sono impresse.

Benché la tesi si rivolga con maggiore insistenza verso le applicazioni civili dell'uso delle impronte digitali, può risultare interessante soffermarsi un attimo sulle tecniche usate dalle polizie scientifiche (o forensi) per la rilevazione delle impronte *latent*.

Per rendere visibili le impronte e quindi utilizzarle in un secondo momento è necessario differenziarle dalla superficie sulla quale si trovano; tradizionalmente questa differenziazione viene ottenuta grazie ad un materiale colorato da applicare sulla zona interessata con un processo fisico o chimico; deve essere quindi prodotto un contrasto tra l'impronta vera e propria e il supporto e che ecceda la soglia rappresentante la sensibilità dell'occhio o del rilevatore fotosensitivo. La quantità minima di residuo latente necessario per effettuare l'operazione chimicamente risulta essere compreso approssimativamente tra i 100 e i 200 ng ($1 \text{ ng} = 10^{-9}$), mentre 500-1000 ng sono richiesti nel complesso per un'aderenza soddisfacente: al di sotto di questi livelli l'impronta può risultare parziale o delineata molto debolmente e quindi inutilizzabile per scopi identificativi.

Tra l'ampia gamma delle tecniche utilizzabili ci sono anche quelle basate sull'interazione tra luce e materia: l'emissione di un'onda elettromagnetica permette infatti una sensibilità nella rilevazione considerevolmente superiore alle tecniche di assorbimento. La fotoluminescenza rientra appunto tra queste, consentendo risultati irraggiungibili fino a circa venti anni fa: essa consiste nell'irradiazione con una luce di

una certa qualità dopo l'esposizione ad una luce di una lunghezza d'onda adeguata. Solitamente queste tecniche sono in grado di rilevare impronte anche direttamente da sole, ma riescono con più successo in concomitanza con altri procedimenti chimici.

La combinazione di metodi ottici (diffusione, luminescenza assorbimento e riflessione di raggi UV), metodi fisici (trattamento con polvere, deposizione del metallo sotto vuoto o VCM), metodi fisico-chimici (cianoacrilato) e metodi basati su reazioni chimiche (nitrato d'argento, ninidrina e suoi derivati, complessazione del metallo) [34] permettono un'alta efficienza nel recupero di impronte lasciate anche su superfici difficili come quelle ruvide e porose, contaminate da altri fattori come per esempio il sangue oppure degradate dal tempo atmosferico o semplicemente dall'età.

2.5.2 Metodo di acquisizione *live-scan*.

Per quanto riguarda il sistema *live-scan* invece, l'impronta è ottenuta senza l'uso intermedio del foglio di carta dato che è necessario solamente premere il dito sulla superficie di acquisizione del sensore e questo ne ottiene un'immagine attraverso differenti tecnologie. Nella più semplice delle ipotesi, basta una sola immagine, ma anche in questo caso è possibile avere immagini *rolled*, e quindi *live-scan* allo stesso tempo: uno scanner un po' più complesso sarà infatti capace di catturare più di una impronta durante il movimento ondulatorio del dito e in seguito ricostruire un'immagine finale attraverso il mosaico di quelle acquisite.

La risoluzione di acquisizione è stata standardizzata da FBI in 500 dot per inch (dpi), e molti scanner sono oggi costruiti con queste caratteristiche [50].

In commercio esistono molte tipologie di scanner; una delle tecnologie sicuramente più diffuse nel mercato è quella ottica, che è anche la più “vecchia” e che si basa su un *Charge Coupled Device* (CCD) usato per esempio anche nelle macchine fotografiche digitali: una luce viene infatti proiettata sul dito e la sua riflessione viene quindi registrata dal CCD e visto che le creste la riflettono maggiormente rispetto alle valli viene ricostruita l'immagine. Problemi di disturbo possono presentarsi nelle situazioni in cui i polpastrelli sono macchiati da inchiostro o nicotina dato che questa tecnologia si basa sulla cattura dell'immagine vera e propria.

La tecnologia basata invece sulla capacità elettrica, si è diffusa alla fine degli anni 90 ed offre invece prestazioni migliori rispetto agli scanner ottici vista la sua tolleranza ai disturbi causati dallo sporco. Il sensore funziona infatti come una piastra del condensatore, mentre il dito rappresenta l'altra: la capacità tra queste due viene poi misurata e convertita in un'immagine digitale. Questi scanner sono però suscettibili alle scariche elettrostatiche.

Altre aziende come Infineon Technologies (sotto divisione di Siemens) e Sony, si sono mosse per contrastare il mercato Veridicom [63], leader in questo settore.

Esistono inoltre scanner termici, tipo quelli basati sul chip AT77C101B di Atmel [61], che acquisiscono l'impronta utilizzando la differenza di temperatura rilevata tra le creste e le valli. Le prestazioni di questi apparecchi non sono tuttavia soddisfacenti per il momento.

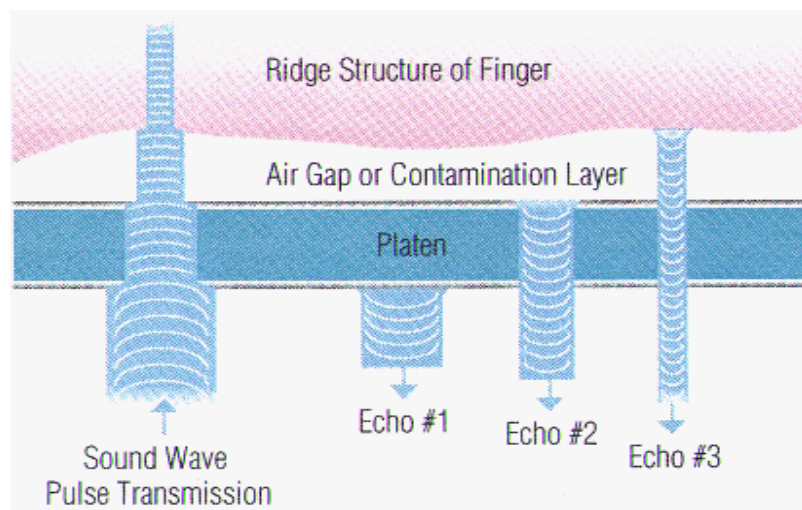


Figura 2.9
Tecnologia a ultrasuoni per il rilevamento delle impronte digitali [61].

Infine esistono anche scanner che utilizzano onde sonore sulle frequenze (da 20 KHz fino ad alcuni GHz) degli ultrasuoni (figura 2.9) che sono capaci di attraversare diversi materiali, escludendo così dall'immagine le contaminazioni dovute allo sporco sulle dita o sul rilevatore. Questi sensori si basano sulla differenza di resistenza acustica tra pelle, aria e piastra di appoggio, che provoca una differente eco di ritorno come mostrato in figura 2.10 e si adatta benissimo anche a condizioni di luce molto intensa, dove i sensori ottici presentano notevoli problemi. La tecnologia ad ultrasuoni è sicuramente la più innovativa adottata in questo campo, ed anche se risulta essere per ora poco diffusa e quindi poco testata, Ultra-Scan Corporation ne commercializza una versione proprietaria [62]. I primi esperimenti sembrano comunque molto promettenti.

2.6 Template e minuzie.

Alla base di qualsiasi sistema di fingerprint matching si trova ovviamente l'informazione che rappresenta l'impronta e su sulla quale l'algoritmo di riconoscimento deve essere applicato: la scelta dei dati da inserire nel template biometrico interessano quindi molto pesantemente la struttura del resto del sistema.

Nel caso delle impronte digitali, i possibili template sono sostanzialmente due e cioè quelli che consistono direttamente nell'immagine in scala di grigio del polpastrello e quelli basati su minuzie.

I primi prevalgono sui sistemi di verifica che utilizzano un matching ottico [46], ma la loro utilità può essere fortemente compromessa da fattori di disturbo come variazioni di luminosità, cicatrici od in generale distorsioni globali dell'immagine; in aggiunta la loro occupazione in termini di spazio può salire vertiginosamente con il numero di impronte memorizzate e non adattarsi a situazioni in cui si ha a che fare con supporti molto limitati (per esempio le smart card). Tale problema può essere parzialmente aggirato utilizzando una piccola porzione dell'immagine, riducendo però anche l'affidabilità della verifica.

D'altro canto l'utilizzo di questo tipo di template molto generale consentirebbe l'applicazione su di esso di algoritmi anche molto differenti tra loro e si potrebbe usare anche su immagini molto degenerate laddove risulterebbe difficile estrarre molte altre informazioni.

La seconda classe di template, che deriva dal modello usato dalle polizie già da molto tempo, utilizza la strutture delle creste nel suo complesso (rappresentazione *ridge-based*) oppure più facilmente, si basa su delle sottostrutture locali dell'impronta: di queste, ne sono state classificate almeno 150 [47]. Data però la difficoltà nella loro rilevazione, visto che alcune sono tra loro molto simili e vista anche la necessità di avere

un'impronta di buona qualità per il loro riconoscimento, la maggior parte dei sistemi utilizza le due strutture più prominenti e cioè *ridge ending* e *ridge bifurcation*, che collettivamente vengono chiamate più precisamente *minuzie* e che corrispondono rispettivamente alle terminazioni e alle biforcazioni delle creste (vedi figura 2.10 tratta da [52] e 2.11 tratta da [51]).



Figura 2.10
Esempi di ridge endings e ridge bifurcation [52]



Figura 2.11
Ridge bifurcation (cerchio) e ridge ending (quadrato) [51]

Queste due classi di punti rappresentano una il duale dell'altra e un cambiamento della pressione del dito durante più scansioni può trasformare una terminazione in una biforcazione; è per questo motivo che numerosi sistemi rappresentano queste

informazioni semplicemente come una lista di punti definiti solamente in base alle loro coordinate spaziali scelte rispetto ad un sistema di coordinate ben fissato.

In realtà, per migliorare le prestazioni di matching, a ciascuna minuzia estratta può essere associata altra informazione come la direzione (per esempio in gradi) della cresta su cui si

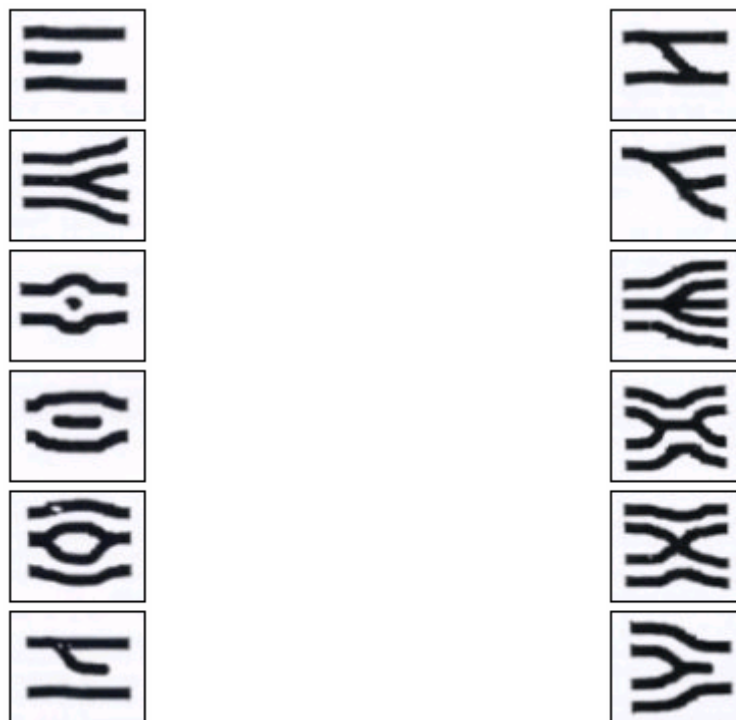


Figura 2.12

Questa figura mostra alcune tipologie di punti di singolarità . Partendo dalla prima colonna in alto a sinistra viene rappresentata una terminazione, una biforcazione, un punto, un'isola, un lago, un uncino, mentre sempre nello stesso ordine, nella seconda colonna ci sono un ponte, una doppia biforcazione, una triforcazione, due biforcazioni opposte, un incrocio di creste ed una biforcazione contrapposta ad una terminazione [43].

trova la minuzia e in quel preciso punto, oppure per ogni coppia di esse può venir associato il numero di creste che le separano e che intersecano il segmento che le congiunge: tale parametro viene chiamato *ridge count*.

La rappresentazione dell'impronta basata su minuzie può comunque contenere informazioni globali che possono aiutare l'algoritmo di verifica o identificazione, come la classe del pattern o le coordinate dei punti di core o delta.

In realtà come accennato precedentemente, la distinzione dei punti di singolarità delle impronte può essere ridotta a punti di biforcazione o di terminazione per rendere più facile la realizzazione del sistema automatico di riconoscimento, anche se nella fisionomia delle creste sono presenti molte più configurazioni maggiormente complesse e che possono venire in aiuto per esempio in un confronto manuale dopo un primo "soltimento" di casi al computer; alcune di queste vengono mostrate nella figura seguente (figura 2.12 tratta da [43]).

I sistemi di verifica includono quindi anche un modulo di *feature extraction*, che ottenuta l'immagine acquisita in ingresso, dopo numerosi passi dedicati alla "pulitura" dell'impronta da disturbi e false minuzie, ottenuta una struttura di creste ben definita, estrae i punti di singolarità con le informazioni aggiuntive necessarie.

Capitolo 3

Matching di impronte digitali.

3.1 Sistemi automatizzati di identificazione di impronte digitali.

Fin dai primi degli anni '60, FBI negli Stati Uniti, Home Office nel Regno Unito e il Dipartimento della Polizia di Parigi hanno investito una grande quantità di sforzi nello sviluppo di un *Automated Fingerprint Identification Systems* (AFIS): in pratica un AFIS consiste in una metodologia di identificazione biometrica basata sulla tecnologia digitale per il trattamento dell'immagine, dalla quale ottenere, memorizzare e analizzare i dati relativi all'impronta.

Gli sforzi compiuti hanno avuto esito così favorevole che un notevole numero di questi sistemi si trova attualmente operativo ed in supporto al mantenimento della legge in molti Stati: il loro uso ha infatti contribuito ad aumentare fortemente la produttività di questi dipartimenti, riducendo al tempo stesso anche i costi per l'assunzione e l'addestramento di esperti di impronte che dovevano svolgere il medesimo lavoro manualmente. Tali successi hanno portato conseguentemente alla migrazione di questi

sistemi da ambiti militari o comunque forensi, ad applicazioni civili ed è per questa loro crescente popolarità che in molti casi sono divenuti sinonimo di sistemi biometrici [129].

Un sistema automatico di verifica di identità si basa essenzialmente su quattro componenti fondamentali:

Acquisizione:

A proposito dei metodi *inked* e *live-scan* per la rilevazione delle impronte si è già discusso esaurientemente nel capitolo 2: una delle tecnologie più diffuse consiste, posto il dito sul vetro del sensore, nell'illuminare il polpastrello con una sorgente di luce LED e tramite un CCD (vedi sempre paragrafo 2.5) catturare le differenze tra la luce riflessa dalle creste e dalle valli.

Rappresentazione (Template):

Chiaramente questo passo è fortemente legato alla fase di verifica, dato che la rappresentazione dell'impronta deve contenere tutte le informazioni necessarie all'algoritmo di matching; vista dall'altra faccia della medaglia, la rappresentazione costituisce l'essenza della verifica. Come già espresso nel capitolo 2, i template possono essere basati sull'intera mappa (o una sua parte di interesse significativo) in scala di grigio, oppure su una lista di minuzie trovate durante la fase di estrazione.

Estrazione delle singolarità:

Un modulo che implementa le funzionalità di estrattore, ricerca nell'immagine i due tipi di minuzie base descritte nel capitolo 2: biforcazioni e terminazioni. Dalla precisione di questa fase dipende pesantemente il risultato finale dell'algoritmo; attraverso alcuni sottopassi consecutivi, eventuali degradazioni presenti nell'immagine vengono rimosse o attenuate, l'immagine viene tagliata in modo da lavorare solamente sulla parte centrale più interessante ai fini del match ed infine l'immagine viene binarizzata come in figura 3.1 (riportata da [28]): in pratica dall'immagine in scala di grigio si decide se un punto appartiene o non appartiene ad una cresta ed il risultato può essere visto come una “mappa” in bianco e nero dell'impronta.

A questo punto viene avviata la fase di estrazione vera e propria che metodicamente scandisce l'immagine binarizzata identificando dei pattern di pixel che indicano la fine o la



Figura 3.1
Binarizzazione di un'impronta digitale [28].

divisione delle creste; come coordinate della minuzia lungo l'asse x e y viene preso in considerazione per esempio il pixel più estremo della cresta binarizzata.

Comparazione (Matching)

Questa è la parte che finalizza tutto il calcolo eseguito precedentemente nelle altre parti del sistema AFIS e quella verso la quale si rivolge il lavoro eseguito per questa tesi.

Il modulo di matching deve determinare se i due template in input rappresentano la stessa impronta digitale. Insieme all'algoritmo di matching vero e proprio deve essere definita anche una sorta di metrica per poter calcolare la "distanza" tra le due impronte ed anche un valore soglia (threshold) per la distanza che stabilisce alla fine il risultato positivo/negativo del match.

In figura 3.2 Viene mostrato come queste componenti possono essere assemblate per realizzare un sistema AFIS: l'utente digita su una consolle il proprio nome o login identificativo e pone poi il dito (per esempio l'indice destro) sul vetro dello scanner. A questo punto, se ci troviamo nella fase di registrazione (enrollment), l'insieme di minuzie rilevate può essere passato ad un ulteriore modulo che ne controlla la qualità, eventualmente scartandolo e richiedendo una nuova registrazione al futuro utente: questo controllo difficilmente può essere effettuato durante la fase di verifica, dato che in questi casi viene richiesta una maggiore velocità di risposta.

Se invece ci troviamo nella fase di verifica, dall'immagine del dito catturata viene estratto un template di minuzie che viene comparato con quello personale mantenuto genericamente in un database e l'algoritmo di matching che ne calcola quindi la distanza. In generale il template di riferimento (quello ottenuto nella fase di registrazione) può essere mantenuto in un database centralizzato, oppure anche su smart card: in questo

secondo caso è necessario che il sistema di interfaccia con l'utente comprenda anche un lettore di carte per poter per lo meno estrarre dal supporto le informazioni necessarie, oppure per poter delegare ad un'applicazione sulla carta di effettuare la fase di matching.

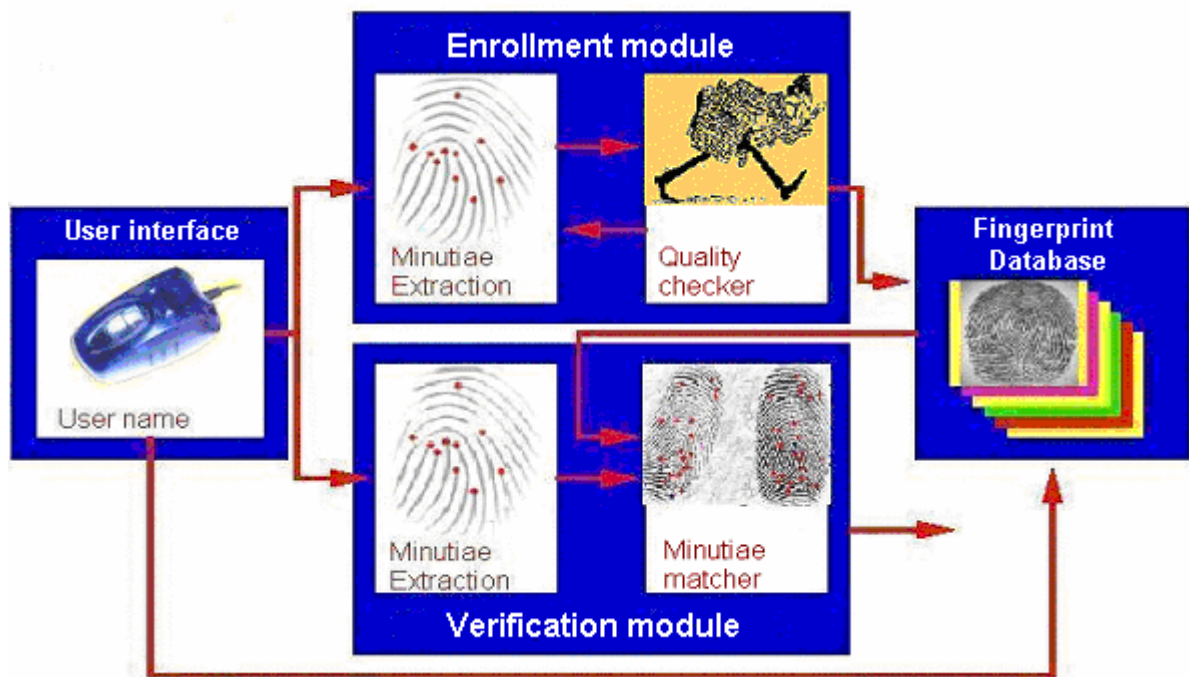


Figura 3.2
Schema del funzionamento di un Automated Fingerprint Identification System (AFIS).

Alcune delle aziende che si muovono nel settore dei sistemi AFIS, sono Biometric Identification [78], Cogent [79], East Shore Technology [80], FingerSec [81], Mentalix [82], NEC [83], NeuroDynamics [84], Optel [85], Sagem Morpho [86].

3.2 Problemi relativi al fingerprint matching.

In questo paragrafo ci concentriamo sulle difficoltà che emergono nella fase di match; ci restringeremo inoltre alle rappresentazioni basate su minuzie, che oltre ad essere usate nella maggior parte dei sistemi di riconoscimento automatici, sono utilizzate anche nel nostro algoritmo di riconoscimento.

Ecco così che il problema di matching di impronte digitali può essere affrontato come problema di matching di punti nel piano, dato che le minuzie altro non sono che punti con delle informazioni aggiuntive in più come ad esempio la direzione della cresta sulla quale si trovano oppure il ridge count (vedi capitolo 5) .

Alla luce dell'ambiente operativo delineato nei capitoli precedenti, la progettazione di un algoritmo di verifica deve quindi tener conto di un modello realistico delle variazioni che possono distinguere due impronte dello medesimo dito; il modello deve infatti affrontare i seguenti punti di interesse:

- ◇ Il dito può essere posizionato in differenti posizioni sul vetro dello scanner (nel caso per esempio di uno scanner ottico) e ciò porta ad una traslazione globale delle minuzie del candidato rispetto al template di riferimento ottenuto all'atto della registrazione dell'individuo.
- ◇ Il dito può essere altresì posizionato sullo scanner con differenti orientamenti che causano rotazioni (di tutte le minuzie) rispetto al template di riferimento.
- ◇ Il dito può esercitare una differente pressione sul vetro causando in questo modo una diversa scala (che interessa l'immagine globalmente) rispetto al template di

riferimento: in pratica le minuzie possono essere tutte più vicine se si è esercitato una pressione maggiore o più lontane altrimenti.

- ◇ Minuzie “false” possono essere presenti sia nel template candidato sia in quello di riferimento: le minuzie false sono minuzie in realtà non presenti sull'impronta digitale, ma comunque evidenziate dal software di rilevazione a causa di disturbi esterni (ad esempio vetro sporco), oppure a causa di difetti più direttamente collegati allo stesso software, che può ad esempio “tagliare” l'immagine dell'impronta formando delle ridge ending inesistenti sul bordo.
- ◇ Allo stesso modo anche delle minuzie “genuine” (presenti effettivamente sull'impronta), possono essere non rilevate sia sul template candidato sia su quello di riferimento; quindi dell'informazione “buona” può essere persa durante il processo di calcolo.
- ◇ Accanto a queste primi ostacoli dovuti a differenti rotazioni, traslazioni, cambi di scala, presenza di informazione fuorviante e assenza di informazione corretta si affiancano anche delle perturbazioni non-lineari dovute all'elasticità intrinseca dello strato epidermico. Queste perturbazioni influenzano negativamente le prestazioni di matching dato che a differenza delle prime sono meno intuitivamente eliminabili: non riguardano le minuzie globalmente, ma ogni zona può essere diversamente interessata. Una loro caratteristica che risulta utile nella definizione dell'algoritmo è rappresentata dal fatto che all'interno di una piccola porzione di immagine queste perturbazioni rimangono pressoché identiche. Considerando le figure 3.3 e 3.4 (tratte ambedue da [69]) si può osservare come un 10% di distorsione in un quadrato sia poco evidente, ma che se questa deformazione si ripete in più quadrati (una zona più ampia), porti a effetti molto vistosi.

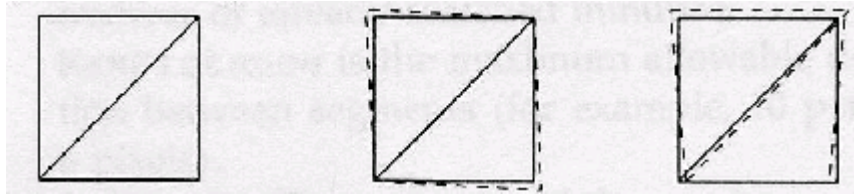


Figura 3.3
Esempi di piccole distorsioni in un quadrato [69].

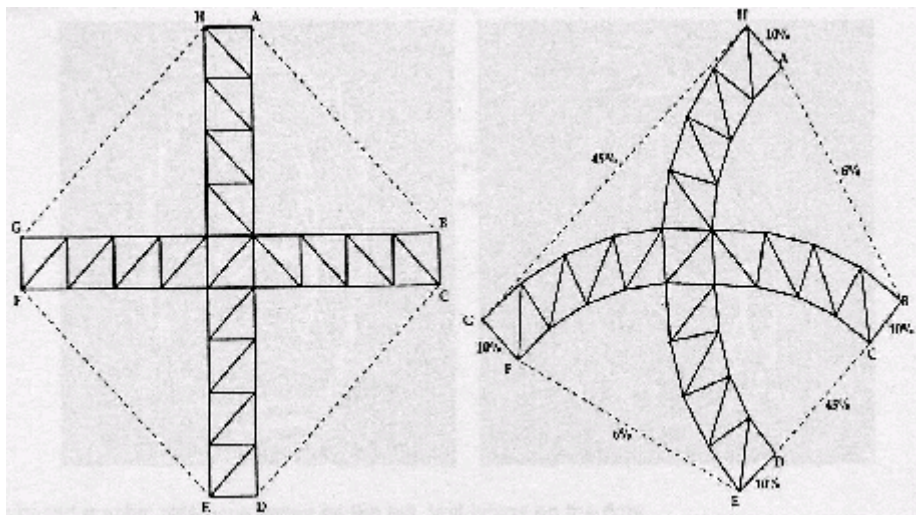


Figura 3.4
Esempio di come una piccola distorsione della figura 3.3 possa portare a vistosi effetti se ripetuta [69].

A seconda di come un algoritmo si comporta nei confronti delle precedenti deformazioni, viene indicato dalla letteratura in modi differenti: al gradino più basso della scala si trovano gli algoritmi “Euclidei”, che sono in grado di affrontare solo trasformazioni rigide tra le rappresentazioni di riferimento e candidata. All’altro capo si trovano gli algoritmi “topologici”, che sono in grado di tollerare in pratica qualsiasi trasformazione, dato che le minuzie dei due insiemi si possono trovare in un ordine spaziale completamente diverso.

Nella realtà implementativa, la maggior parte degli algoritmi è in grado di affrontare solo le deformazioni lineari e la presenza/assenza di alcune minuzie, anche se esistono dei matcher (come il nostro) in grado di sostenere piccole perturbazioni locali, ma non grandi scarti spaziali delle minuzie dalla loro vera posizione: questi algoritmi vengono definiti dalla letteratura come “Elastici” [35, 64].

Il problema quindi di trovare un “buon” allineamento tra due diverse immagini della stessa impronta non è comunque facilmente risolvibile, dato che una passabile sovrapposizione in una certa zona, come quella centrale, può portare a forti discrepanze nelle zone periferiche.

Altri fattori che possono perturbare ulteriormente le prestazioni dell’algoritmo di matching possono essere rappresentati dallo “sporco” sul vetro dello scanner (dipende dalla tecnologia implementata nel sensore) dovuto ad un uso frequente, a macchie (per esempio di inchiostro) sulla pelle del polpastrello oppure anche altre “interferenze”, come timbri o firme solitamente presenti su documenti o schede informative della polizia. In figura 3.5 sono riportati i più significativi fattori che possono influire sulla qualità dell’immagine dell’impronta digitale, con il loro impatto negativo su tale qualità così come risulta in [51].

Fattore	Impatto
Lavoro manuale	Medio
Condizioni metereologiche	Medio
Posizionamento del dito sul sensore	Alto
Sporco o grasso sul polpastrello	Medio
Tagli e abrasioni sul dito	Alto

Figura 3.5
Fattori che influiscono la qualità dell'immagine. [51]

3.3 Valutazione delle prestazioni dei sistemi biometrici.

Nei sistemi di riconoscimento automatico la corrispondenza tra due impronte digitali è espressa in termini di un *coefficiente di similarità* che corrisponde alla probabilità che le due impronte appartengano allo stesso individuo. Per determinare tale valore si definiscono una *metrica di similarità* e una soglia (*threshold*): la prima stabilisce le regole mediante le quali calcolare il coefficiente di similarità e tipicamente tale valore è proporzionale al numero di minuzie (vedi paragrafo 2.6) corrispondenti nelle due impronte; la seconda indica il valore minimo al di sopra del quale due impronte sono considerate “simili”.

Esiste una importante distinzione tra i tradizionali metodi di autenticazione come le password o i PIN e la biometria: mentre per i primi, una volta digitati, il risultato del controllo assume valori “vero” o “falso” con estrema precisione vista la facilità dell'algoritmo, per la seconda tecnica di riconoscimento il valore deve essere

ulteriormente giudicato per poter stabilire definitivamente la positività o la negatività della verifica.

L'algoritmo può quindi sbagliare con una certa percentuale dato che possono sempre esistere due (o più) persone con caratteristiche fisiche o comportamentali molto vicine tra loro, che il procedimento di calcolo non è in grado di distinguere a causa del suo grado di approssimazione; parimenti, un utente può non essere riconosciuto positivamente a causa della scarsa tolleranza verso disturbi nel template appena acquisito durante la scannerizzazione (per esempio a causa di un leggero raffreddore nel riconoscimento vocale).

Queste misure sopra introdotte vengono chiamate tolleranza di Tipo I (*False Rejection Error*) e di Tipo II (*False Acceptance Error*) come già accennato precedentemente ed il loro settaggio stabilisce pesantemente le prestazioni del sistema: FRR causa frustrazione, mentre FAR lascia aperta la porta alla frode.

Gli errori di Tipo I e di Tipo II possono essere tradotti graficamente nelle curve di *false acceptance* e *false rejection* che servono per stabilire la sensitività del sistema: idealmente queste due curve dovrebbero entrambe tendere a zero per qualche valore soglia (threshold) e quindi le prestazioni dovrebbero mostrare un forte incremento generale.

È piuttosto evidente che le distribuzioni dei due errori FAR (*False Acceptance Rate*) e FRR (*False Rejection Rate*) sono due quantità legate da una corrispondenza inversa: maggiore è la soglia imposta per il riconoscimento positivo, minore è la probabilità di accettare erroneamente un impostore, ma maggiore è anche la possibilità di rifiutare per errore il legittimo proprietario del template.

Realisticamente invece queste curve mostrano un punto di intersezione (*equal error rate*) nel quale si “scavalcano” e che indica il valore nel quale i due errori si

presentano con la stessa percentuale di frequenza. Nella figura A.3 viene mostrato l'andamento di FAR e FRR nei sistemi di riconoscimento: in applicazioni dove la sicurezza rappresenta l'obiettivo primario sarà molto importante avere un valore di falsa accettazione molto basso (FAR bassa), come nel caso ambienti militari; questo andrà a discapito della percentuale dei falsi rifiuti, che sarà elevata. Nelle applicazioni forensi invece, risulterà preferibile l'esatto contrario: avere un FRR basso, quindi di conseguenza un FAR alto, per cercare di non escludere nessun criminale che ha un'impronta digitale simile a quella trovata sul luogo del crimine. La diminuzione degli errori del TIPO I porta quindi ad un aumento di quelli del TIPO II e viceversa.

Per il resto delle applicazioni civili invece è più logico tarare il sistema nel suo punto di equilibrio tra FAR e FRR.

La curva nella figura 3.6 viene chiamata *Receiver Operating Curve* (ROC) e per la prima volta venne usata per le rilevazioni dei segnali radar nella seconda guerra mondiale: serve quindi per poter osservare con facilità il tradeoff tra FAR (chiamata anche *False Match Rate*: FMR) e FRR (chiamata anche *False Non Match Rate*: FNMR). La curva di ROC viene trovata utilizzando i valori di FAR e FRR trovati per la stessa soglia.

Risulta chiaro che per lo studio delle prestazioni del sistema e la sua messa a punto sarà necessario tracciare più curve di ROC, una per ogni configurazione di importanza rilevante.

Inoltre, la presenza di due fattori per la misurazione delle performance rende spesso poco chiare le dichiarazioni da parte delle aziende produttrici che alcune volte rendono pubblico il migliore dei due dati; sovente poi questi valori soglia sono poi modificabili tramite configurazioni diverse e quindi l'informazione può essere "confusa" ulteriormente.

Benché questi valori siano considerevolmente migliorati nel corso dell'evoluzione di tutti i sistemi conosciuti, esistono ancora differenze importanti da un sistema all'altro, in parte dovute al metodo di identificazione prescelto. Fino ad ora non esistono standard per i test. Molto è stato fatto per arrivare alla standardizzazione ma sono state riscontrate varie difficoltà per via della differente natura dei sistemi.

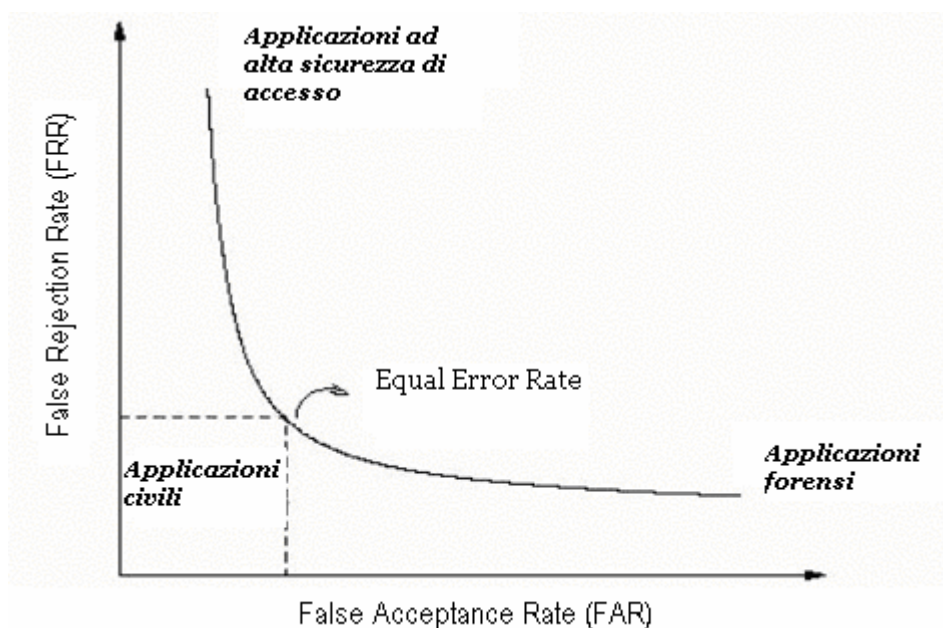


Figura 3.6
FRR, FAR e applicazioni biometriche nella curva di ROC.

Dato che, come già accennato in precedenza, la decisione del riconoscimento può decretare come risultato che l'individuo sia “genuino” oppure un “impostore”, ciò può essere rappresentato da due distribuzioni chiamate appunto distribuzione genuino e distribuzione impostore. La misura statistica d' le utilizza entrambi per valutare la distanza tra di esse e quindi risulta utile come ulteriore valore indicante le prestazioni; d' viene calcolata con la seguente formula [22]:

$$d' = \frac{|| M_{impostore} - M_{genuino} ||}{\sqrt{(SD_{impostore}^2 - SD_{genuino}^2)}}$$

dove $M_{impostore}$, $M_{genuino}$, $SD_{impostore}$, $SD_{genuino}$ rappresentano i valori della media e della deviazione standard delle due distribuzioni.

Nella valutazione di un sistema biometrico ricopre molta importanza anche la velocità di verifica, dato che l'utente è costretto ad attendere la sua terminazione per poter usufruire del servizio controllato da tale sistema; ecco perché per esempio risulta difficile l'implementazione di algoritmi efficienti su piattaforma smart card, visto i limiti di risorse nei quali è costretto a muoversi lo sviluppatore.

Un sistema biometrico si può basare interamente su di un processo di verifica biometrica *monomodale*, ovvero incentrata sull'analisi di un unico insieme di dati biometrici (ad esempio l'elenco delle minuzie di un'impronta digitale). Soluzioni teoricamente più "precise" possono venire sicuramente dall'applicazione di sistemi di verifica *multimodali*, realizzati cioè utilizzando due o più metodologie biometriche, ad esempio analizzando iride e impronta digitale, oppure impronte digitali di più dita dello stesso soggetto; questo aumento di dati può essere molto utile per raggiungere un FAR molto basso in applicazioni che richiedono estrema sicurezza [135]. Chiaramente, un utilizzo di più tecnologie biometriche può comportare un aumento del codice da eseguire, con conseguente aumento dei tempi necessari per ottenere il risultato della verifica: questo aumento può essere incompatibile con la natura stessa del sistema di sicurezza come nel caso di sistemi molto lenti come ad esempio smart card.

3.4 Template biometrici.

Con template intendiamo la registrazione della misurazione di una caratteristica biometrica dell'utente. Questo dato può chiaramente essere memorizzato su supporti diversi, a seconda dell'applicazione che ne fa uso ed anche della quantità di dati che ne fanno parte:

Memory of Biometric Device:

Il template viene memorizzato nella memoria del Biometric Device, che può avere una capacità anche per qualche migliaio di utenti, variando naturalmente da modello a modello. Questa soluzione risulta essere sicura, dato che nessun dato viene trasmesso e quindi non può essere soggetto a intercettazione e risulta essere anche economico, dato che non sono necessari costi aggiuntivi per la distribuzione di strumenti di riconoscimento agli utenti (per esempio, smart card). Non costituiscono comunque una buona scelta laddove sia richiesto un elevato numero di utenti (come un sistema di schedamento di criminali), oppure l'utente debba essere autorizzato in diverse locazioni differenti (per esempio, porte di accesso ad aree con livello di sicurezza più o meno stringente di banche o aeroporti, ma anche uffici statali sparsi sul territorio).

Data Base Centrale:

I template possono essere mantenuti in un database centralizzato se il numero degli utenti richiesto dall'applicazione è molto grande, oppure nei casi in cui la verifica debba essere eseguita remotamente.

Come però in generale tutti i sistemi centralizzati, l'aspetto riguardante la sicurezza non può essere facilmente messo da parte: infatti la privacy dei template viene compromessa nei casi di abuso o errore da parte degli amministratori del sistema, ma anche da eventuali intrusi interni o provenienti dall'esterno della propria rete e anche la sicurezza del canale di trasmissione tra apparati di scannerizzazione e sistema centrale rappresenta un punto focale per stabilire la reale impenetrabilità del database.

Plastic Cards o Tokens:

Quest'ultimo metodo consente l'utente di tenere sempre con sé i dati personali necessari per il riconoscimento effettuato dai veri strumenti di identificazione compatibili; questo metodo risulta essere il più appropriato nei seguenti casi:

- ◇ Il numero di dati da registrare, o più chiaramente, il numero di utenti, è molto grande per un database centrale: le prestazioni di riconoscimento risulterebbero essere rallentate dalla ingente mole di comparazioni per quanto riguarda per esempio la verifica.
- ◇ Gli utenti devono essere verificati remotamente.
- ◇ Il trattamento sicuro dei dati rappresenta una priorità.

La paura del "Grande Fratello" che può minare in partenza l'accettabilità delle tecnologie basate sulla biometria è aggirabile anche grazie allo spostamento del template dal database centrale alla carta personale: in questo modo l'utente ha la sicurezza che i

propri dati segreti rimangano sempre con sé e quindi non esista la possibilità che vengano divulgati volontariamente per rintracciare gli spostamenti oppure involontariamente in seguito a attacchi o guasti del sistema. In questo senso anche l'educazione sulla sicurezza e riservatezza delle nuove tecnologie può venire in aiuto per rendere il cliente più disponibile verso questi nuovi sistemi dato che infatti, per esempio per quanto riguarda le impronte digitali, l'accostamento a sistemi simili adottati per il riconoscimento dei può sorgere immediatamente.

3.5 Stato dell'arte.

Data la maggiore popolarità di questa tecnica biometrica basata su impronta digitale, esistono numerosi realizzazioni di sistemi per la classificazione e la successiva verifica. Anche se il *National Institute of Standards and Technology* (NIST) fornisce alcuni database per la valutazione delle performance sul banco di prova, molte compagnie riportano spesso risultati ottenuti sui loro database proprietari e quindi tali risultati non possono essere indipendentemente verificati e comparati da più laboratori scientifici diversi. Alcune di queste prove su benchmark proprietari si segnalano per prestazioni molto buone, alcune delle quali sono riportate nella tabella 3.1 da [135]; nella seconda tabella invece (la 3.2 sempre da [135]) sono riportati i risultati di alcuni algoritmi ottenuti mediante dei test su database comuni di impronte digitali nel corso di una competizione, le caratteristiche dei quali verranno approfondite meglio nel capitolo 6: come si può notare le prestazioni sono notevolmente peggiori, come ci si poteva chiaramente aspettare. Informazioni maggiori al riguardo possono essere recuperate in

[65]. Molti di questi algoritmi si basano sull'uso delle minuzie e di altri punti caratteristici delle impronte, come punti di core o delta, per il loro funzionamento.

Numerosi documenti riguardanti progetti biometrici, tra cui molti sulle impronte digitali, possono essere reperiti da [87], dalla pagina internet sulla ricerca biometrica mantenuta dall'università dello stato del Michigan negli Stati Uniti: nello svolgimento del nostro lavoro ha rappresentato una delle fonti principali di documentazione. La maggior parte dei sistemi di verifica automatizzati di identificazioni sono costituiti da soluzioni proprietarie di aziende o comunque di progetti governativi sui quali vengono divulgate poche notizie per tutelarne la sicurezza.

Compagnia (sito web)	Sensore	FAR (%)	FRR (%)
Biolink USA (biolinkusa.com)	Ottico	0.0000001	0.01
BiometricId (biometricid.com)	Ottico	0.01	0.01
Startek (startek.com.tw)	Ottico	0.001	3.3
IOSoftware (iosoftware.com)	Ottico	0.1	1
Identix (identix.com)	Ottico	0.0001	1
NEC (necotech.com)	Stato solido	0.0002	0.05
Biometrix Int. (biometrix.at)	Stato solido	0.001	0.0001
Pollex (pollex.ch)	Stato solido	0.001	1
Sony (sony.com)	Stato solido	0.001	1

Tabella 3.1

Alcune delle prestazioni di algoritmi rilevate dai siti web delle compagnie. Nessuna di esse specifica il database sul quale sono state effettuate le prove e così le prestazioni non possono essere comparate in maniera equilibrata [135].

Algoritmo	EER (%)	T medio registr. (secondi)	T medio verifica (secondi)	Reject rate (%)
Sag1	3.64	5.70	2.13	0.00
Sag2	4.01	1.94	1.94	0.00
Cspn	5.33	0.35	0.35	1.81
Cetp	8.29	1.49	1.66	0.00
Cwai	5.90	0.46	0.57	20.86
Krdl	8.03	1.48	1.60	11.98

Tabella 3.2

Nella tabella sono riportati i risultati dei test di alcuni algoritmi di fingerprint matching effettuati su un database comune; maggiori informazioni sono disponibili su [65].
Nella prima colonna viene riportata la sigla dell'algoritmo in competizione corrispondente in generale alla sigla delle compagnie che lo presenta [135].

Gli algoritmi di verifica di impronte digitali si dividono sostanzialmente in due grandi categorie dipendenti dall'informazione sfruttata: negli algoritmi basati su minuzie vengono sfruttati punti singolari, come le terminazioni e le biforcazioni delle creste, che sono state descritte nel paragrafo 2.6. Questo approccio alla verifica presenta alcuni problemi generali come l'essere in grado di trovare tali punti con una certa affidabilità, la sensibilità alla distorsione dell'immagine e il costo computazionale; d'altra parte l'accuratezza della verifica risulta essere molto alta perché basata su micro caratteristiche dell'impronta. In [139] vengono mostrati tre principali sotto tipi di algoritmi basati su minuzie:

- ◇ Basati sulla trasformazione di Hough: è composto da tre principali passi, nel primo dei quali si cercano i parametri di trasformazione tra i due template come la traslazione sull'asse x, su y, la rotazione e il fattore di scalatura. Nel secondo passo si allineano i due insiemi di minuzie secondo i valori calcolati nel precedente passo e si contano le minuzie che possono essere fatte corrispondere

tramite l'applicazione di una zona di delimitazione centrata sulla minuzia chiamata *bounding box*. Nel terzo passo si ripetono i primi due per l'insieme discretizzato delle possibili trasformazioni. Il valore di matching finale viene riportato in un valore tra 0 e 99; un esempio viene descritto in [64].

- ◇ Basati sulla distanza tra stringhe: ciascun insieme di minuzie viene trasformato in coordinate polari facendo riferimento ad un certo punto base (punto d'ancora); le caratteristiche in due dimensioni delle minuzie sono quindi trasformate in una stringa unidimensionale concatenando i punti in ordine crescente rispetto all'angolo radiale delle coordinate polari. A questo punto basta calcolare la distanza di editazione tra le due stringhe per ottenere il valore del match. Un esempio di questa classe di algoritmi ([35]) viene descritta più approfonditamente nel seguito.
- ◇ Basati sulla programmazione dinamica in due dimensioni: questo tipo di algoritmo rappresenta una generalizzazione di quello basato sulla distanza tra stringhe e cerca di migliorarne le prestazioni evitando la perdita di informazioni costituita dal passaggio da due a una dimensione. In [77] viene data una dimostrazione di come questo possa accadere mantenendo anche una buona efficienza nel costo computazionale: come nel precedente caso si stima la differenza di rotazione tra le due impronte e si adattano i due insiemi di minuzie secondo questo valore. A questo punto si calcola il numero di minuzie accoppiate secondo una tecnica di programmazione dinamica in 2D; intuitivamente, in questo passo si cerca di deformare un insieme di minuzie per allinearle con l'altro e massimizzare il numero di minuzie accoppiate. Il valore di output (normalizzato), si basa solamente sul numero di minuzie che coincidono

applicando anche in questo caso una *bounding box* e applicando infine un termine di penalizzazione basato sulle minuzie che non sono state accoppiate.

Il secondo tipo di informazione utilizzabile è rappresentata direttamente dal flusso delle creste, chiamato in letteratura *oriented texture* e più solitamente usato invece nella classificazione delle impronte come descritta in 2.4; gli algoritmi basati su questa macro informazione presentano generalmente forte sensibilità alla perdita di parti dell'immagine che possono pregiudicare l'affidabilità della funzione di verifica. Inoltre si basano sull'identificazione di alcuni punti particolari dell'impronta (come quello di *core* descritto in 2.4) che sono altrettanto difficili da trovare rispetto alle minuzie e possono non essere presenti nell'immagine per colpa del processo di acquisizione o semplicemente a causa della natura dell'impronta.

Esiste poi una terza tipologia di algoritmi che cerca di fondere insieme le caratteristiche dei primi due per cercare chiaramente di unificarne i lati positivi e migliorarne quindi le prestazioni.

Alcuni algoritmi importanti esaminati oltre a quelli descritti nel paragrafo 5.1 [72] e [69], sono stati inoltre [35], [135] e [75]; rappresentano un valido esempio per ciascuna delle tre tipologie di verifica presentate precedentemente.

L'algoritmo descritto nel documento "*An Identity Authentication System Using Fingerprints*" [35] utilizza le minuzie per confrontare due impronte; la fase di matching si compone di due sottofasi: nella prima fase di allineamento vengono stimate le trasformazioni come le traslazioni, le rotazioni e le differenze di scalatura tra i due template e conseguentemente le minuzie nel template in input vengono allineate con quelle del template registrato precedentemente nel database seguendo le indicazioni

trovate. Nella seconda fase invece, quella di matching vero e proprio, le minuzie dei due template vengono convertite nel sistema di coordinate polari in due poligoni, ai quali viene applicato poi un algoritmo elastico di matching tra stringhe.

Per quanto riguarda la seconda tipologia di algoritmi e cioè quelli basati su texture, uno dei migliori algoritmi per prestazioni e velocità, allo stato della nostra conoscenza risulta essere quello descritto in “*Finger classification and matching using a filterbank*” [135]: nel primo passo viene costruito un vettore di feature dall’applicazione di un insieme di filtri di Gabor; il vettore viene chiamato *finger-code*. Più precisamente viene prima localizzato il punto di *core* dell’impronta, poi attorno a questo punto viene considerato un cerchio suddiviso in N settori; conseguentemente viene filtrata l’immagine grazie ad un insieme di M filtri di Gabor che evidenziano maggiormente il flusso delle creste. A questo punto viene calcolata la deviazione standard dei valori di livello di grigio in ciascuna immagine filtrata e questi calcoli vengono salvati in un vettore $N \times M$ chiamato *finger-code*. Il matching tra due *finger-code* (e quindi due impronte) avviene semplicemente calcolando la distanza euclidea tra i due. Nel documento si fa riferimento ad un vettore di dimensione 640 e cioè 80 settori per 8 filtri e se ciascuna delle feature può essere quantizzata in un valore da 1 a 266 (1 byte), il *finger-code* occupa solamente 640 byte; chiaramente questi parametri possono variare in funzione della risoluzione e della dimensione dell’immagine.

In “*Fusion of Multiple Matchers for Fingerprint Verification*” [75] invece si cerca di ovviare ai problemi introdotti dagli algoritmi basati su minuzie o su texture cercando di fondere insieme le caratteristiche delle due tipologie; viene così utilizzato un algoritmo del primo tipo ed uno del secondo: partendo dalla rappresentazione delle due impronte in minuzie e texture, vengono calcolati i punteggi (un valore nell’intervallo $[0,1]$) di somiglianza tra le due impronte per tutti e due gli algoritmi. Dopodiché viene calcolato

$$s = f(s_m, s_t)$$

in pratica un nuovo punteggio ottenuto applicando una funzione ai due punteggi provenienti dal primo e dal secondo algoritmo. Infine se s è strettamente maggiore di una certo valore soglia il match è ritenuto positivo, altrimenti, negativo.

Gli algoritmi utilizzati per i match sono [52] per quello basato su minuzie (il cui punteggio finale è rappresentato da s_m) e [41] per quello basato su texture (il cui punteggio finale è rappresentato da s_t); i valori dei punteggi dei due algoritmi vengono poi normalizzati nell'intervallo $[0,1]$ e servono infine da ingressi per la funzione di “mediazione” che può essere per esempio la media o il prodotto dei due.

Anche i lavori [139] e [76] rappresentano altri sistemi nei quali si cerca di integrare tipologie di verifica diverse.

Allo stato della nostra conoscenza quindi, nessun sistema per la verifica di impronte digitali si basa sull'uso di algoritmi di ricerca operativa rivolti alla determinazione dell'accoppiamento di costo minimo su grafi come quello da noi realizzato e descritto nella trattazione di questa tesi.

Capitolo 4

Smart card e Java Card™.

4.1 Introduzione.

In questo capitolo viene introdotto l'ambiente nel quale è stato sviluppato il nostro nuovo algoritmo di fingerprint matching, che verrà a sua volta descritto nel capitolo successivo.

I primi paragrafi si rivolgeranno ad uno studio della tecnologia smart card in generale e all'ambiente basato sul linguaggio Java installabile su di essa, mentre successivamente verranno presentate le API introdotte da Java Card™ Forum per la realizzazione di algoritmi di riconoscimento basato su caratteristiche biometriche della persona. Seguono alcune immagini di smart card che mostrano il loro attuale impiego in alcuni settori e per servizi differenti.



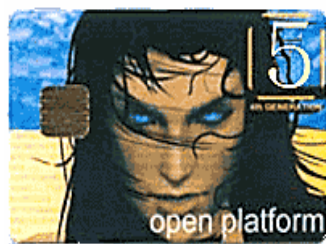
a)



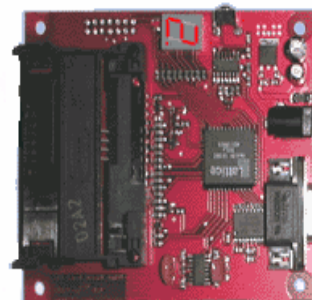
b)



c)



+



d)

Figura 4.1

Alcuni esempi riguardanti l'uso di smart card: a) una carta di credito bancario, b) carta telefonica prepagata, c) un validatore di carte *contactless* per il trasporto urbano, d) kit per "costruirsi" in casa un accesso alla TV satellitare.

4.2 Smart Card.

Identica nelle dimensioni rispetto ad una normale e onnipresente carta magnetica tipo carta di credito (85.6mm x 53.98mm x 0.76mm), una smart card memorizza e processa informazione grazie ai suoi circuiti elettronici protetti da un involucro di plastica.

La figura 4.2 mostra inoltre come i contatti siano presenti esternamente, mentre il microprocessore sia immerso all'interno dell'involucro di plastica tramite l'uso di una resina epossidica.

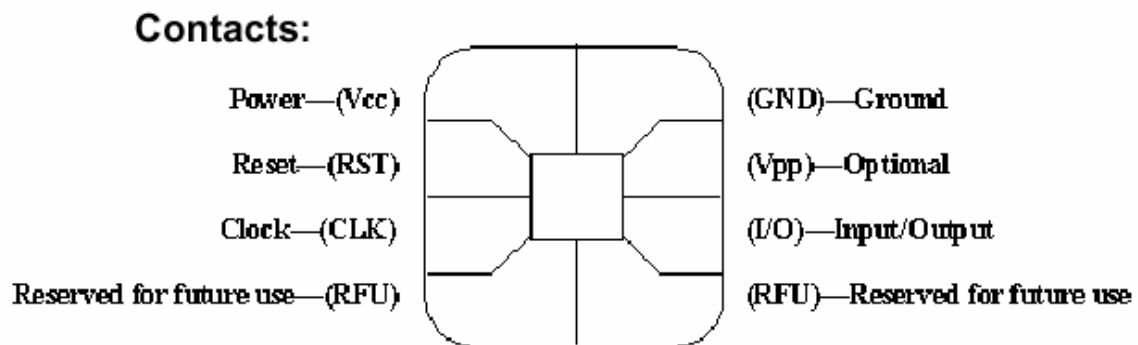


Figura 4.2
Contatti esterni di una smart card.

Il nome non deriva dalle capacità di calcolo interne e che la potrebbero quindi distinguere fortemente dalle altre tecnologie basate su banda magnetica, bensì dal rafforzamento delle regole di controllo per gli accessi alla memoria: per esempio, le zone contenenti il nome del possessore della carta possono essere rese di sola lettura dopo una prima loro scrittura. Il controllo degli accessi viene affidato ad un semplice processore o anche ad una circuiteria più semplice nel caso di carte di basso prezzo [103].

La tecnologia fu indipendentemente sviluppata in Germania (1967), Giappone (1970), Stati Uniti (1972) e Francia (1974), anche se si ritiene comunque che l'invenzione sia da attribuire a Roland Moreno proprio in quest'ultimo paese, a cui si deve anche il suo nome: la frase "smart card" è stata infatti coniata nel 1980 per descrivere al grande pubblico questa tecnologia durante una grande campagna pubblicitaria ideata da Roy Bright, direttore del marketing della società francese Intelmatique. Anche se il termine

smart card cattura facilmente l'immaginazione, risulta comunque essere allo stesso tempo ambiguo ed è usato in diversi modi differenti.

International Organization for Standardization (ISO) [100], alla quale si devono gli standard più importanti che riguardano questa tecnologia, adotta il termine *Integrated Circuit Card* (ICC) per inquadrare tutti quei dispositivi che contengono un circuito integrato all'interno del rettangolo di plastica individuato nelle misure dal formato ISO ID1.

Le ICC si dividono in due famiglie: le *contact* (a contatto), fornite di pin che hanno bisogno di entrare in contatto con il lettore/scrittore di smart card (che può essere definito in letteratura anche con il nome di *Card Acceptance Device* o CAD) per acquisire dati e ricevere corrente per il chip (figura 4.5) e le *contactless* (a prossimità).

4.2.1 Smart Card contactless.

Le smart card contactless differiscono da quelle a contatti perché lo scambio delle informazioni non avviene attraverso il canale fisico realizzato dai contatti della carta e del lettore, ma per trasmissione in radio frequenza; nel seguito del capitolo utilizzeremo il termine lettore al posto del più prolisso lettore/scrittore. Nella plastica della carta è presente una minuscola antenna in grado di rilevare un segnale emesso a distanza da una antenna presente su un dispositivo fisso. La lettura/scrittura della carta può avvenire ad una certa distanza dall'apposito lettore dipendente dallo standard usato: omm-10mm (Close-Coupled Cards), standard ISO/IEC 10536, 10mm-10cm (Proximity cards) secondo ISO/IEC 14443 oppure 10cm-50cm (Vicinity cards) secondo ISO/IEC 15693; queste ultime consentono all'utente di passare accanto al lettore senza esplicitamente rivolgere la carta verso di esso. Standard sulle smart card *contactless*

possono essere reperiti sul sito del *Working Group 8* (WG8) [101], uno dei Working Group (attualmente in totale sono 10) all'interno di ISO/IEC JTC1/SC17 [102] organizzato nel 1988 col fine unico di sviluppare standard per questa specifica tecnologia [101].

L'alimentazione e la comunicazione avviene per mezzo di un campo elettromagnetico di bassa potenza generato dal lettore (figura 4.3).

Oltre alla praticità di utilizzo dovuta al fatto che non occorre inserire la carta nel lettore, in quanto è sufficiente avvicinarla, le carte a prossimità risultano praticamente indistruttibili poiché, non presentando contatti esterni che si possono danneggiare in seguito a graffi o a piegature ed inoltre non sono soggette ad usura.

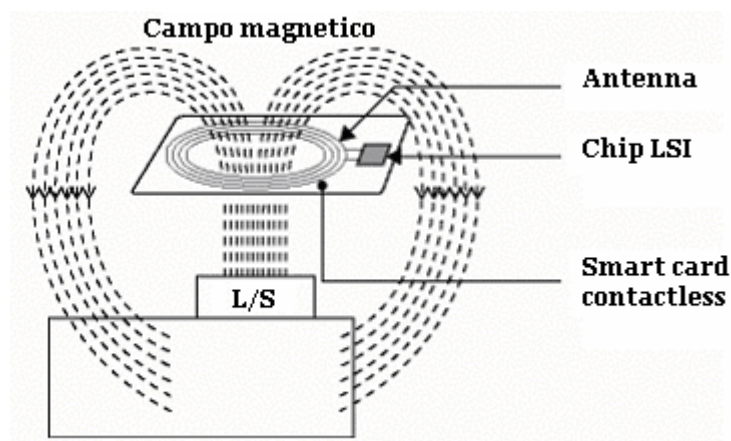


Figura 4.3
Trasmissione nella tecnologia contactless.
L/S rappresenta il lettore/scrittore di carte.

Le carte a prossimità sono particolarmente adatte nei contesti in cui è indispensabile uno scambio di dati rapido: un'applicazione tipica di queste carte è infatti quella dei trasporti pubblici, per il pagamento di biglietti o validazione di abbonamenti

su autobus, metropolitane, oppure il controllo degli accessi e/o il pagamento ad aree controllate come uffici, parcheggi, centri storici, ecc....

4.2.2 Super Smart Card.

In realtà, tra le due grandi famiglie *contact* e *contactless*, si pone una terza tecnologia più ambiziosa, già testata (per esempio da Visa ai giochi olimpici di Atlanta del 1996), ma per la quale non esiste ad oggi in mercato degno di nota: mentre le prime due vengono ritenute sostanzialmente passive, le *Super Smart Card* (figura 4.4) incorporano una tastiera ed un display direttamente sulla superficie e sono dotate di una propria sorgente di energia (tipicamente una batteria). Anche se possono essere presenti comunque dei contatti come sulle *contact* card (vedi in seguito le *CombiCard*) per avere la possibilità di essere usate in un normale lettore/scrittore, il principale punto di forza delle Super Smart Card (carte “attive”) rimane quello di poter funzionare come unità stand-alone in qualsiasi locazione; come controparte rimane lo svantaggio del costo di produzione superiore a quella delle carte passive.



Figura 4.4
Super Smart Card.

Questa tecnologia rimane tuttora in fase di sviluppo e prova ne è il fatto che non sia stato stabilito ancora nessuno standard a riguardo, al contrario delle *contact* e *contactless*.

Oltre alle *Super Smart Card*, come ulteriore tecnologia esistono anche le *CombiCard* che in realtà non aggiungono però niente di veramente nuovo includendo sia le caratteristiche delle *contact*, sia delle *contactless*, ma anche una banda magnetica ed un codice a barre per poter usare la carta in ambienti molto diversi tra loro.

4.2.3 Smart Card contact.

Dopo aver presentato brevemente alcune tecnologie meno diffuse, ma con un futuro probabilmente entusiasmante, ci occuperemo adesso più approfonditamente delle smart card *contact*, le più diffuse e popolari e che portano con sé alcuni svantaggi rispetto alle carte senza contatti fisici: questi si possono infatti usurare o semplicemente sporcare riducendo caratteristiche come l'affidabilità e la robustezza e pregiudicando così il tempo di vita del supporto. Naturalmente in contrapposizione a questi esistono anche alcuni vantaggi come la velocità e il basso costo della loro produzione ed un grado di sicurezza potenzialmente maggiore.

Come già accennato, queste carte devono essere inserite in un CAD per poter scambiare dati e ricevere voltaggio attraverso i contatti (o pin), che vanno un minimo di 5 ad un massimo di 8 e la cui definizione è oggetto del documento standard ISO 7816, sia nelle loro caratteristiche fisiche, sia nelle loro posizioni e dimensioni.

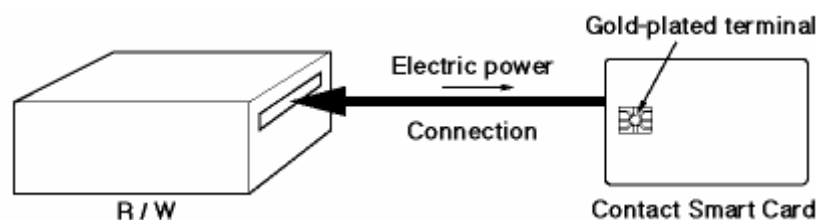


Figura 4.5
Contact card.

Come si può notare anche molto schematicamente dalla figura 4.2, i contatti solitamente presenti consistono nel segnale di clock per pilotare la logica della carta (CLK), il pin per fornire energia (Vcc), quello per scambiare informazioni con il CAD, quello per fornire il segnale ad alto voltaggio necessario per programmare la memoria EPROM (Vpp),

il contatto per indicare al microprocessore di avviare la sequenza di reset delle istruzioni (RST), la terra elettrica tra il lettore e la carta (GND) ed infine due contatti riservati per uso futuro (RFU); tutta questa specifica è contenuta nell'ISO 7816-parte 2 [100].

Mentre nelle prime carte il voltaggio fornito era di 5 Volt, sfruttando i passi in avanti nella tecnologia dei semiconduttori, consentendo una minore dissipazione di energia; per quanto riguarda la frequenza del ciclo di clock invece, la maggior parte dei processori in commercio supporta meno di 10Mhz e possiede un'architettura ad 8 bit, anche se esistono realizzazioni a 16 e 32 bit.

Passiamo adesso all'architettura tipica di una smart card (figura 4.7): dato che l'utilizzo primario delle ICC si basa sulla facilità di trasporto e sul recupero delle informazioni, il componente fondamentale consisterà quindi in almeno un modulo di memoria; tipicamente saranno presenti però più moduli di memoria, ciascuno diverso tecnologicamente e sfruttato per un bisogno differente. Potrà essere presente quindi una

memoria ROM (Read Only Memory), che una volta scritta nel processo di costruzione della compagnia di semiconduttori, non potrà più essere modificata e quindi sarà adatta per contenere ad esempio il sistema operativo della carta. Accanto a questa prima memoria poco costosa può essere affiancata una EEPROM (Electrically Erasable Programmable ROM) per poter contenere le applicazioni scaricate sulla carta in secondo momento: questa memoria infatti ha la capacità di effettuare da 10,000 a 1,000,000 riscritture e quindi risulta essere adatta per ambienti multi applicazione. Proprio in questi ambienti vengono aggiunti nel tempo nuovi servizi o vecchie loro versioni vengono sostituite: questa è proprio una delle fondamentali direzioni verso le quali si sta spostando il mondo smart card.

Infine sarà presente una memoria RAM (Random Access Memory) che rappresenterà l'unico modulo di memoria volatile; questa tecnologia è quindi adatta per contenere i dati necessari solamente durante la sessione d'uso della carta e non per memorizzarli tra sessioni diverse, come potrebbe risultare indispensabile per un credito in denaro; si presta quindi a contenere il codice in esecuzione o lo stack del programma.

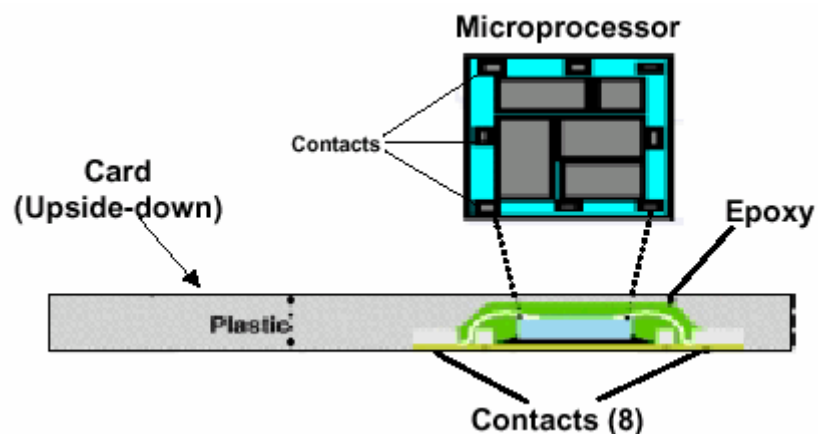


Figura 4.6
Sezione di una smart card [30].

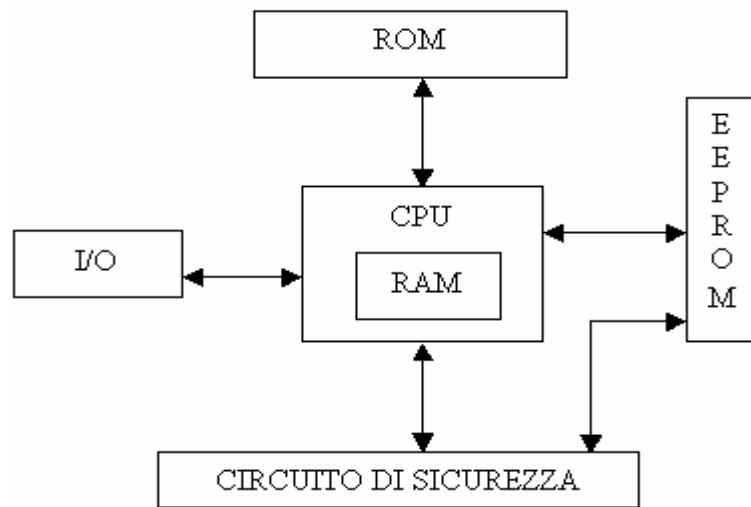


Figura 4.7
Struttura hardware di una smart card intelligente (con processore).

Tanto per rendere un'idea sulle capacità molto ristrette della tecnologia rispetto ad un normale PC di fascia anche bassa, tipicamente la RAM non supera i 4-8Kb, mentre la EEPROM i 64Kb.

4.2.4 Architetture delle Smart Card.

Dal punto di vista delle funzionalità, le smart card si dividono in due grandi architetture: le *Memory Smart Card* (o *Synchronous Card*) e le *Intelligence Smart Card* (*Asynchronous Card*).

Memory Smart Card:

Questo tipo di carte è puramente rivolta solamente alla memorizzazione di dati visto che non possiede né un microprocessore, né un sistema operativo: una volta collegata con il mondo esterno, quei chip che includono una logica per il controllo degli accessi operano

semplicemente settando un interruttore on/off in base alla comparazione delle informazioni riguardanti il codice PIN con un'area di memoria riservata: una volta che il test è stato superato la memoria è leggibile o modificabile a seconda delle proprie caratteristiche, indirizzandola direttamente.

Intelligence Smart Card:

A differenza delle precedenti, le carte “intelligenti” contengono, oltre alla memoria, una Central Processing Unit (CPU) [104] con capacità di calcolo e di lettura/scrittura autonome. Una volta interfacciato con il lettore/scrittore e fornito il voltaggio, la carta lavora sotto il controllo di un sistema operativo definito dall'acronimo SCOS (*Smart Card Operating System*).

Vista questa capacità di calcolo in più, nasce conseguentemente il bisogno di introdurre un protocollo di comunicazione più complesso per assolvere al compito di scambiare comandi e dati tra i due interlocutori: l'accordo su tale protocollo tra ICC e CAD avviene durante il primo flusso di dati scambiati non appena la carta viene inserita nel lettore: il processore invia infatti questa informazione, assieme ad un identificatore del tipo del chip, nello stream *Answer To Reset* (ATR) non appena gli viene fornita energia. Il canale di comunicazione risulta essere una linea half-duplex sulla quale, o il lettore parla e la carta ascolta, oppure viceversa il lettore riceve ciò che la carta gli ha inviato. Sopra questo canale fisico si trova logicamente un protocollo a livello di link grazie al quale il lettore può inviare alla carta un comando che scatena una serie di operazioni interne; alla fine questa può rispondere con dei dati rappresentanti il risultato ed un codice di status del comando. Anche se lo standard elenca più di quindici protocolli, ci sono attualmente due sole principali varianti di questo protocollo usate nei sistemi smart card: il protocollo T=0 e quello T=1. Il primo dei due è stato anche il primo

ad essere ideato e definisce la natura delle richieste e delle risposte che attraversano il canale di comunicazione basandosi sul carattere: in pratica il byte rappresenta l'unità di informazione trasferita e conseguentemente anche la gestione dell'errore si rivolge a questo "corto" raggruppamento di bit; per inviare un byte di dati occorrono in pratica 10 bit come mostrato nella figura 4.8, comprendendo infatti un bit di inizio sequenza e uno di parità per controllare eventuali errori nella trasmissione. Nel protocollo T=1 viene scambiato invece un blocco di dati e quindi, oltre a risultare più complessa in generale la sua definizione, anche la gestione dell'errore si deve basare su delle tecniche più articolate come il *longitudinal redundancy character* o un *cyclic redundancy check* (CRC) come riferito nello standard ISO 3309 [100]; nonostante questo però, la maggior parte delle carte esistenti oggi sul mercato usa ancora il protocollo T=0.

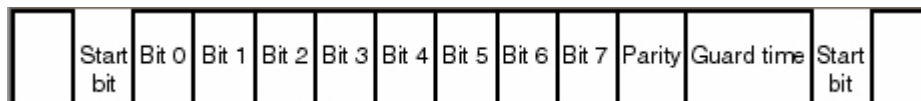


Figura 4.8
Bit trasmessi per byte di informazione nel protocollo T=0 [30].

Come in tutti i protocolli di comunicazione, anche nel caso delle smart card viene definito un formato per lo scambio di messaggi che nel caso specifico prende il nome di *Application Protocol Data Unit* (APDU) e che viene trasportata sul protocollo T=0; con questo termine viene indicato sia il formato del messaggio di comando inviabile dal CAD alla carta, sia il formato della risposta (differente dall'altro) elaborata da quest'ultima: il sistema si presenta quindi in configurazione master/slave, con la carta che gioca sempre il ruolo passivo.

Alcune carte, come quella utilizzata nel kit di sviluppo di *Cyberflex Access Developer 32K card* [130] adottato per implementare su carta l'algoritmo oggetto di questa tesi, possono supportare tutti e due i protocolli di comunicazione T=0 e T=1.

4.2.5 Standard ISO su Smart Card.

Per quanto riguarda l'intera tecnologia *contact* può risultare utile riassumere le varie parti dello standard che la riguarda, l'ISO 7816, il cui nome formale è *Integrated Circuit Cards with Electrical Contacts*, dimostrando quindi di rivolgersi proprio alle carte con contatti elettrici.

L'inizio della stesura del documento risale al 1987, anche se nel tempo sono state aggiunte numerose informazioni e correzioni: le migliorie vengono indicate da ISO con il nome di *amendment*.

- ◇ Parte 1: definisce le dimensioni fisiche dei contatti sulla smart card, la loro resistenza all'elettricità statica, alle radiazioni elettromagnetiche e agli stress meccanici. Riguarda inoltre la locazione fisica della banda magnetica (che può essere presente).
- ◇ Parte 2: definisce la locazione, lo scopo e le caratteristiche elettriche dei contatti metallici della carta.
- ◇ Parte 3: definisce il voltaggio ed i requisiti per i contatti elettrici definiti nella parte 2 ed inoltre anche il protocollo di trasmissione dati asincrono e half-duplex a caratteri (T=0). La modifica 1:1992 introduce anche il protocollo sincrono half-

duplex a blocchi. Un'altra modifica è stata aggiunta nel 2002 e riguarda le caratteristiche dei circuiti elettrici operanti a 5V, 3V e 1,8V.

- ◇ Parte 4: riguarda la standardizzazione di un insieme di comandi per fornire accesso, sicurezza e trasmissione dei dati sulla carta.
- ◇ Parte 5: stabilisce gli standard per l'attribuzione degli *Application Identifiers* (AIDs) per identificare univocamente un'applicazione presente sulla carta. Un identificatore di questo tipo è formato da due parti: la prima parte di 5 byte si chiama *Registered Application Provider Identifier* (RID) ed è unico rispetto al produttore di applicazioni, mentre la seconda parte, che ha una lunghezza variabile fino a 11 byte, si chiama *Proprietary Application Identifier Extension* (PIX) e serve al produttore per identificare applicazioni specifiche all'interno del dominio acquisito. I RID sono distribuiti dalla compagnia Copenhagen Telephone Company Ltd. (KTAS), presente in Danimarca [121].
- ◇ Parte 6: descrive le regole di codifica per i dati utilizzati da applicazioni di uso comune, come il nome e la fotografia dell'utente, le lingue scelte, etc.
- ◇ Parte 7: descrive come trattare i dati sulla carta per gestirli come un database SQL o meglio, *Structured Card Query Language* (SCQL).
- ◇ Parte 8: aggiunge ai comandi introdotti nella parte 4 delle estensioni per la crittografia a chiave simmetrica e asimmetrica.
- ◇ Parte 9: aggiunge i comandi necessari alla personalizzazione, come Create File e Delete File a quelli presentati nella parte 4.
- ◇ Parte 10: Definisce le basi per il protocollo di comunicazione denominato T=14, riservato per le realizzazioni proprietarie delle aziende.

Altri documenti non hanno raggiunto ancora lo stadio finale di standard, ma permangono ancora nello stato provvisorio di bozza (Draft del 2002), come *ISO 7816-11: Personal verification through biometric methods*, che intende occuparsi della verifica biometrica su smart card dotate di contatti elettrici, oppure *ISO 7816-15: Cryptographic information application* che riguarda metodi standard per accedere a certificati e chiavi pubbliche memorizzati sulla carta. In ISO 7816-11 delinea uno standard che incorpora gli oggetti biometrici che rappresentano l'informazione e che incontrano i requisiti proposti da *Common Biometric Exchange File Format (CBEFF)* [93]: CBEFF definisce i campi base dell'informazione biometrica, come per esempio le componenti delle minuzie, senza vincolare la loro codifica.

Genericamente, CBEFF descrive un insieme di elementi per supportare una tecnologia biometrica (qualsiasi) in modo comune consentendo in pratica l'interoperabilità di applicazioni e sistemi di produttori differenti; il formato di ANSI/NIST risulta completamente compatibile con questo standard.

ISO7816-11 stabilisce così un formato per queste informazioni che devono risiedere su smart card, lasciando al suo interno anche lo spazio per tipi di informazione proprietari.

4.3 Tecnologia Java Card™.

Descrivendola in poche parole, una Java Card™ è una smart card sulla quale possono essere eseguiti programmi scritti in un sottoinsieme del linguaggio Java [107, 110]. La scelta in questa direzione può essere giustificata da due principali punti di forza, che sono la semplificazione nello sviluppo di applicazioni grazie ad un linguaggio ad alto

livello che si basa su delle API standardizzate e la ricerca dell'adattamento della filosofia WORA ("Write Once, Run Anywhere") al mondo delle smart card; alternative possono essere rappresentate da sistemi operativi proprietari, lo standard MULTOS mantenuto attualmente dal consorzio MAOSCO (Europay, Mastercard, American Express, etc) [109] e piattaforme Microsoft per le quali le applicazioni vengono scritte in Visual Basic.

Questi benefici rivolti al mondo dei programmatori si sono rivelati molto invitanti, tanto che oggi sono numerose le aziende che producono carte con piattaforme Java Card™ come Gemplus [112], Giesecke & Devrient [113], Oberthur Card Systems [114], Orga [115], e Schlumberger [116].

Le specifiche e il kit di sviluppo per questa tecnologia è giunto ormai alla versione 2.2 rilasciata pubblicamente nel Giugno del 2002 [106] (con una recente revisione, la 2.2.1 ancora ancora allo stato di "beta" [125]), anche se risulta essere "backward-compatible": programmi sviluppati sotto la precedente versione 2.1.1 possono essere eseguiti senza modifica sulla piattaforma 2.2. Le più importanti funzionalità offerte dalla nuova "release" riguardano l'introduzione di un sottoinsieme della tecnologia *Remote Method Invocation* (RMI), il supporto a nuovi algoritmi crittografici come l'*Advanced Encryption Standard* (AES), una migliorata compatibilità, supporto verso le smart card contactless di prossimità ed una gestione più ottimizzata della memoria.

In realtà le piattaforme Java e Java Card™ differiscono profondamente l'una dall'altra visto che le API Java Card™ sono state interamente riviste per la piattaforma con l'eccezione di poche classi contenute nel package *java.lang*, ridotte comunque nelle funzionalità. Allo stesso tempo la JCVM (*Java Card™ Virtual Machine*) non implementa concetti di programmazione come dynamic class loading, bytecode verification, security manager, thread, object cloning, garbage collection, sincronizzazione e tipi primitivi che occupano molto spazio di memoria: sono supportati

solamente *boolean* (1 byte), *byte* (1 byte) e *short* (2 byte). Per questi motivi, è sicuramente più corretto definire Java Card™ come una realizzazione di Java orientata alle smart card e non un suo sottoinsieme. Ciò nonostante, il programmatore abituato al fratello più grande si troverà molto a suo agio anche in questa piattaforma dai limiti certamente più stringenti, dato che la sintassi di base è pressoché identica e per la prima fase della scrittura del programma è necessario un normale compilatore Java.

I componenti fondamentali della tecnologia Java Card™ comprendono le specifiche Java Card™ e il kit di sviluppo, che possono essere scaricate dal sito di Sun Microsystems [107]. Tali specifiche si suddividono in:

- ◇ *Java Card™ 2.2 Virtual Machine Specification, Revision 1.0, Sun Microsystems, Inc.*: definiscono le caratteristiche, il comportamento ed i servizi richiesti da una implementazione della tecnologia Java Card™: a sua volta viene definita dal set di istruzioni accettato dalla JCVM, il sottoinsieme supportato del linguaggio Java ed i formati dei file usati per installare applet e librerie all'interno della carta.
- ◇ *Java Card™ 2.2 Runtime Environment (JCRE) Specification, Revision 1.0, Sun Microsystems, Inc.*: definiscono il comportamento richiesto obbligatoriamente da qualsiasi implementazione di Java Card™ da parte dell'ambiente runtime: essa include una implementazione della JCVM, le classi Java Card™ Application Programming Interface (API) ed i servizi del supporto runtime, come la selezione e la deselection degli applet.
- ◇ *Java Card™ 2.2 API Specification, Revision 1.1, Sun Microsystems, Inc.*: contiene le definizioni delle classi per supportare JCVM e JCRE.

Oltre alle specifiche, la tecnologia Java Card™ contiene un kit di sviluppo [107] che riunisce un set di tool utili nello sviluppo di applet come il *C-JCRE*, che consiste in una realizzazione in linguaggio C del JCRE, contenente anche un interprete JCVm, un insieme di componenti off-card come *Java Card™ Converter* e *Java Card™ Verifier*, che servono nella catena di sviluppo dell'applet. Nel kit sono presenti poi altri strumenti per scrivere e testare i propri programmi, come *apdutool*, il cui scopo è quello di costruire i messaggi da inviare alla carta per verificare la corretta risposta al comando esterno.

I passi per lo sviluppo di un applet Java Card™, come specificati da Sun, sono riassunti graficamente nell'immagine 4.9 e possono essere suddivisi temporalmente nei sei seguenti punti:

- 1) Scrivere il codice Java con un normale editor.
- 2) Compilare il codice sorgente con un semplice compilatore Java per ottenere dei file *.class*.
- 3) Compilare i file *.class* ottenuti dalla compilazione in un file *Converted Applet* (CAP) grazie al tool *Java Card™ Converter*.
- 4) Verificare che il file CAP sia valido tramite lo strumento *Java Card™ Verifier* (questo passo è opzionale).
- 5) Installare il file CAP sulla carta.

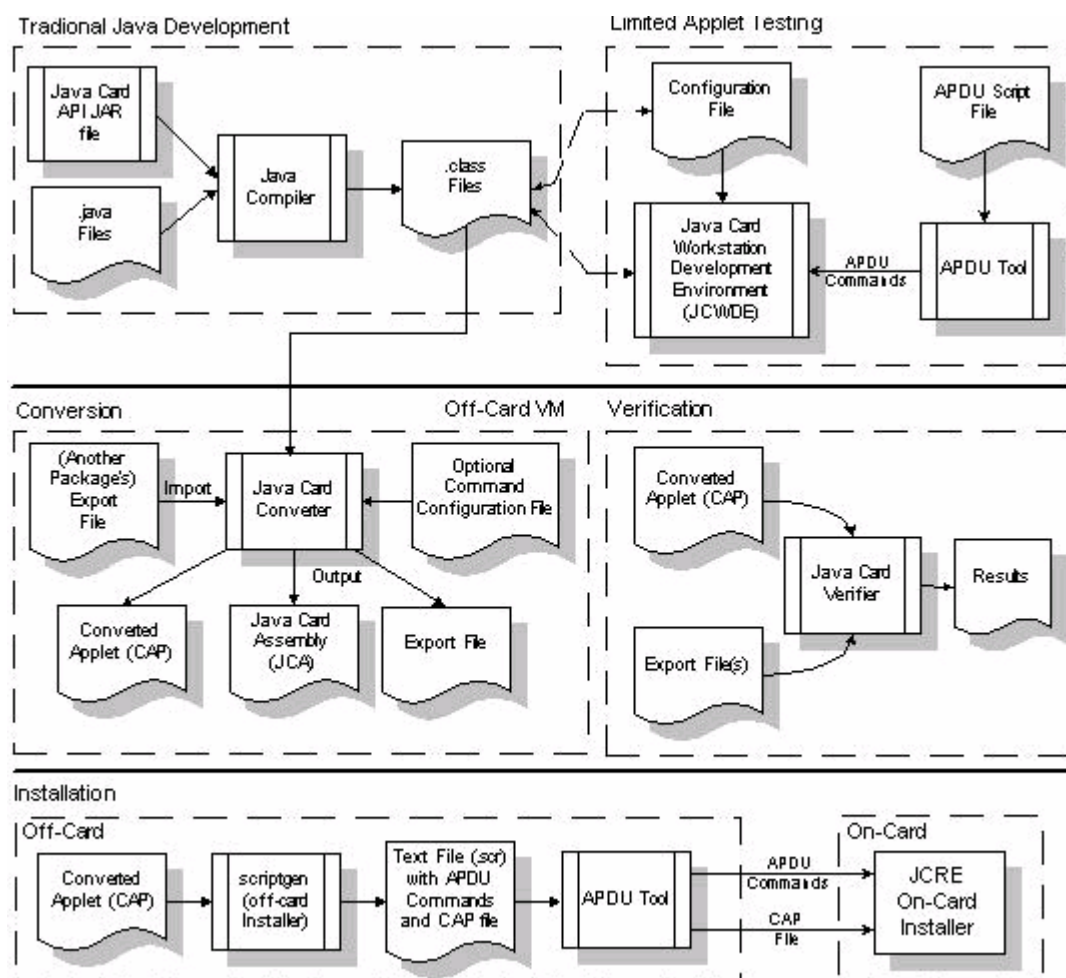


Figura 4.9
Passi per lo sviluppo di un applet Java Card™ [122].

Mentre i primi due passi sono del tutto identici a quelli seguiti durante lo sviluppo di applicazioni Java, gli altri sono del tutto innovativi rispetto alla tecnologia “padre”. Questi nuovi passi derivano dal fatto che le funzionalità della Virtual Machine Java vengono suddivise in una parte su carta (*on-card JVM*) e una parte al di fuori della carta (*off-card JVM*), cioè su PC, per concentrare le operazioni più pesanti dal punto di vista del calcolo e della memoria al di fuori delle limitate risorse della carta; per spezzare queste funzionalità, nasce così il bisogno di introdurre nuovi passi di sviluppo.

Prima che l'applet venga installato sulla carta, il passo di conversione trasforma ciascun package Java in un file CAP che contiene la rappresentazione binaria combinata di tutte le classi e le interfacce del package (operazione off-card). Come mostrato anche in figura 4.11, oltre ai file *.class* il *converter* può ricevere in input anche dei file *export* che contengono informazioni riguardanti altri package importati dalla applicazione, che vengono a loro volta caricati sulla carta; il *converter* può restituire in output anche file *export*.

Di seguito può succedere un ulteriore passo di verifica del file CAP per validarne la struttura, il bytecode e le dipendenze tra i package. Questo è un passo tipicamente off-card, anche se alcune carte possono contenere al loro interno un verificatore on-board. Gli strumenti utili in questa fase sono gli script (a loro volta contenuti nel kit di sviluppo come gli altri [123]) di *verifycap*, per controllare i file CAP, *verifyexp* per i file *export* e *verifyrev* per verificare la compatibilità dei binari tra revisioni di package.

Il ciclo di sviluppo di un applet, gli strumenti e la spiegazione del loro funzionamento sono descritti approfonditamente in [111].

Infine l'applet viene installata sulla carta e istanziata con un proprio identificatore unico denominato *Application Identifier* (AID), approfondito nel paragrafo 7.1.

I package che fanno parte della specifica delle API [108] sono *java.lang* che fornisce le classi fondamentali del linguaggio (tipo *Object* e *Throwable*), *javacard.framework* che fornisce le funzionalità centrali di un applet (vedi dopo) come la classe *APDU*, *javacard.security* che aggiunge un framework rivolto al mantenimento della sicurezza e *javacardx.crypto*, un'estensione rivolta alla crittazione/decriptazione dei dati.

Le applicazioni Java Card™ vengono chiamate con il nome di applet ed anche se coincidono nel nome, l'unica caratteristica in comune che hanno con i più conosciuti applet Java, è che sono ambedue oggetti. Un numero arbitrario di classi e interfacce può costituire un applet, anche se obbligatoriamente una delle classi deve estendere *javacard.framework.Applet* ereditando i seguenti metodi:

- ◇ *deselect()*: invocata dal JCRE per informare l'applicazione in esecuzione che ne verrà selezionata un'altra.
- ◇ *getShareableInterfaceObject()*: chiamata dal JCRE per ottenere un'interfaccia di un oggetto condivisibile da questo applet server, per conto di un applet client.
- ◇ *install()*: JCRE chiama questo metodo per creare un'istanza della sottoclasse di *Applet* e viene chiamato quindi all'installazione dell'applet sulla carta.
- ◇ *process()*: viene chiamato da JCRE e serve all'applet per processare appunto il comando contenuto in un messaggio ricevuto dal lettore/scrittore.
- ◇ *register()*: usato dall'applet per registrare il proprio identificatore unico dell'applicazione (AID) ed informare JCRE.
- ◇ *selec()*: invocato da JCRE per informare l'applet che è stato selezionato.
- ◇ *selectingApplet()*: questo metodo viene usato dal metodo *process* per distinguere la APDU che ha selezionato l'applet da altre APDU con comandi di selezione che riguardano file o stati interni dell'applet.

In realtà nello sviluppo della propria applicazione è veramente necessario definire solo i metodi *install* e *process*, dato che gli altri hanno un comportamento di default per il quale più difficilmente si sente il bisogno di una modifica.

Il ciclo di vita di una smart card inizia con la scrittura nella memoria ROM del sistema operativo, della Java Card™ VM e delle librerie API di classi: questo processo di “burning” di informazioni permanenti in una memoria non modificabile per il trattamento dei comandi viene chiamato *masking*. Dopo questa fase, e sempre prima che la carta arrivi nel portafoglio dell’utente, seguono le tappe di inizializzazione e di personalizzazione; nella prima si caricano ancora dati generali non pertinenti al singolo utente nella memoria non volatile, come per esempio il nome del costruttore. Nella fase di personalizzazione invece, si attribuisce una persona alla carta e si può suddividere nella parte fisica, per esempio scrivendo direttamente sulla plastica il nome, oppure (per noi più interessante) nella parte elettronica, aggiungendo sempre il nome, un login, una chiave personale, un PIN o informazioni biometriche.

A questo punto possono essere aggiunte con il tempo anche nuove applicazioni grazie all’ambiente Java Card™ e la carta rimane utilizzabile fino alla data di scadenza oppure fino ad errori che la bloccano definitivamente, come per esempio il fallimento nello stabilire un canale sicuro di comunicazione tra il lettore/scrittore e la carta per un certo numero consecutivo di tentativi.

Ognuna delle applicazioni deve essere installata con una sequenza di numeri lunga da 5 a 16 byte definita con il nome di AID, per poterla identificare univocamente all’interno della carta: non possono esistere due applet con lo stesso AID sullo stesso supporto; questo codice si suddivide in una parte indicante il package ed una riservata a ciascun applet facente parte del package (anche uno solo) [118].

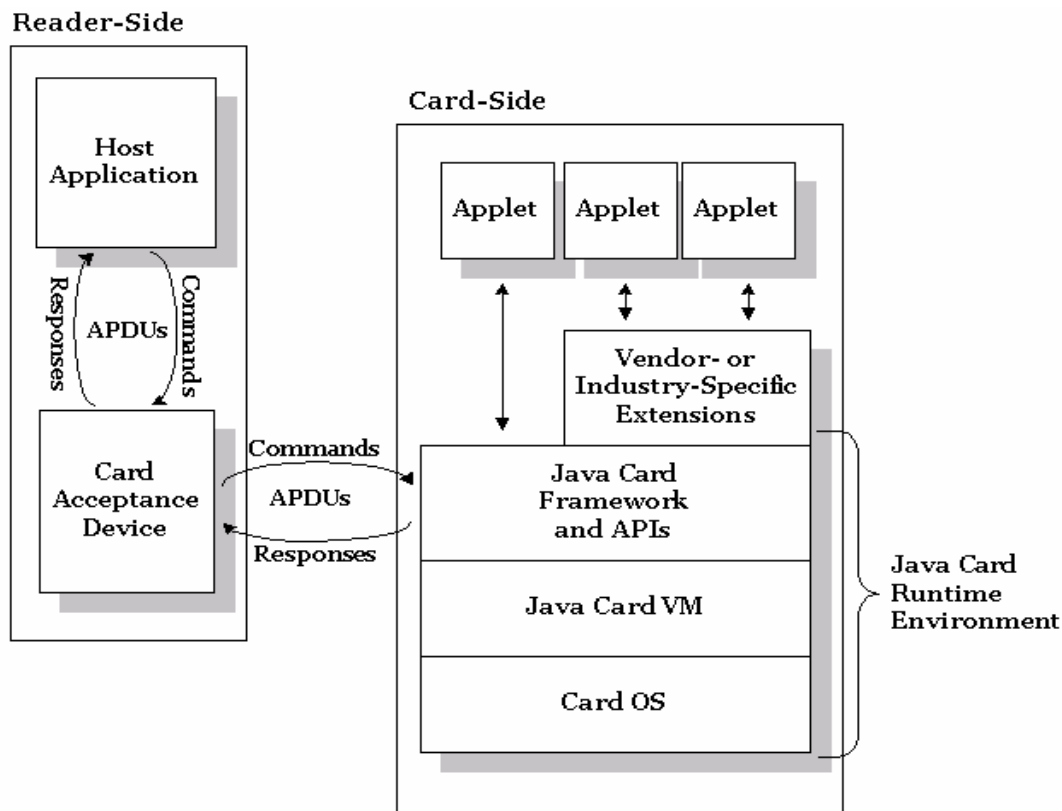


Figura 4.10
Architettura di un'applicazione Java Card™ [124].

Nell'architettura mostrata in figura 4.10 (tratta da[124]), la Java Card™ Virtual Machine si trova sopra il livello dell'hardware e quello strettamente legato ad esso rappresentato dal sistema operativo; la JVM nasconde quindi la realizzazione proprietaria del costruttore con in linguaggio ed un'interfaccia verso il sistema comuni. Al di sopra, il framework Java Card™ definisce ed offre un insieme di API per lo sviluppo di applicazioni e fornire ad esse funzionalità di sistema; sopra a queste possono essere aggiunte librerie specifiche verso servizi offerti da aziende commerciali e quindi non standard. Infine, in cima alla pila s trovano le applicazioni Java Card™ (applet). La figura mostra anche che Java Card™ Framework, Java Card™ Virtual Machine e il sistema operativo della carta formano il Java Card™ Runtime Enviroment.

Il ciclo di vita della Java Card™ Virtual Machine coincide in pratica con quello della stessa carta su cui si trova: comincia poco dopo che la carta è stata costruita e testata, prima che venga rilasciata al possessore finale e termina alla sua distruzione. La JCVM non si ferma infatti quando viene interrotta la corrente e cioè tipicamente quando viene rimossa dal lettore/scrittore, dato che il suo stato viene mantenuto nella memoria non volatile della carta, mentre ciò che è contenuto nella RAM viene irrimediabilmente perso; quando poi viene fornita nuovamente energia è possibile quindi riottenere lo stato della JVM e degli altri oggetti salvati nella EEPROM e l'esecuzione riparte normalmente. Alla sua partenza iniziale la JCVM inizializza il JCRE e crea tutti gli oggetti necessari del JCRE framework che persisteranno per tutta la vita della JCVM.

Per quanto riguarda il ciclo di vita dell'applet invece, esso comincia dopo che viene scaricato sulla carta e dopo che JCRE invoca il metodo statico dell'applet *Applet.install()* e dopo che l'applet si auto registra con il JCRE tramite il metodo *Applet.register()*; dopo questa fase l'applet è nello stato di *non selezionato*, ma pronto per ricevere messaggi. I metodi appena descritti sono riassunti nella figura 4.11 tratta da [124].

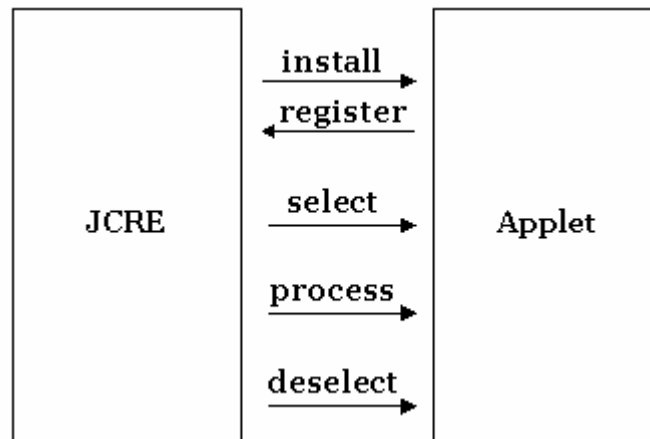


Figura 4.11
Interazione tra applet e JCRE nel ciclo di vita dell'applet [124].

Mentre si trova nello stato di *non selezionato*, l'applet risulta inattivo. Passa invece nello stato attivo (per processare messaggi) di *selezionato* quando l'applicazione lato PC ordina al lettore/scrittore di inviare alla carta un messaggio che la seleziona (*SELECT APDU*); in pratica questo messaggio arriva al JCRE che a sua volta invoca il metodo *select()* dell'applet che tipicamente inizializza il suo stato e si prepara quindi a ricevere altri messaggi. A questo punto JCRE passa all'applet selezionato tutti i successivi messaggi che provengono dal lettore/scrittore e raccoglie tutte le eccezioni che non vengono raccolte dall'applet durante la sua esecuzione. La deselection avviene quando l'applicazione lato PC seleziona un'altra applicazione sulla carta: JCRE avverte l'applet invocando il suo metodo *deselect()* che tipicamente ha il compito di lasciare l'applicazione in uno stato consistente; l'applet torna infine nello stato di *non selezionato*.

4.4 Smart card e biometria.

A seconda di quante funzionalità vengono implementato sulla smart card possono essere definite diverse architetture nelle quali può essere organizzato un sistema biometrico di verifica: ciascuna di esse è caratterizzata da differenti livelli di sicurezza, costi e problemi implementativi.

La prima soluzione prende il nome di *Template On Card* (TOC) e costituisce la configurazione più semplice possibile realizzabile sulla carta per quanto concerne un sistema biometrico di verifica. In questo caso sulla smart card costituisce solamente un deposito sicuro dei propri dati che possono venir protetti anche con un protocollo di criptazione tra carta e lettore. Dato che la verifica viene effettuata sul lato “fisso”, la sicurezza può venir notevolmente minacciata modificando per esempio il software di matching.

Nella seconda soluzione rappresentata dal *Match On Card* (MOC) la situazione riguardante la sicurezza migliora, dato che la procedura di verifica delle due impronte si sposta sulla carta, che a questo punto deve essere dotata di risorse aggiuntive di calcolo e di memoria: ci dovrà essere infatti spazio per due template e non più uno solo ed un ambiente in grado di eseguire calcoli necessari ad un algoritmo abbastanza complesso.

Allo stesso tempo sul “lato workstation” rimane solamente da realizzare la lettura del template, che quindi non ci assicura ancora a priori una acquisizione live (live-scan): i dati che arrivano alla smart card possono così essere modificati o recuperati da una sessione precedente (replay attack) in modo da “truffare” il sistema.

L'ultima soluzione è costituita dal *System On Card* (SOC), che nella pratica corrisponde al sistema MOC con in più anche la presenza del lettore biometrico sulla stessa carta: chiaramente i costi aumentano drasticamente rispetto alle prime due architetture e si presentano anche notevoli problemi di integrazione; *System On Card*

diventa così possibile solo per alcune rilevazioni biometriche nelle quali lo scanner da utilizzare non sia di grandi dimensioni (come appunto le impronte digitali) e sia realizzabile sull'area della carta. D'altro canto, la sicurezza offerta aumenta ulteriormente dato che tutto il sistema viene realizzato su un unico supporto, evitando così pericolose comunicazioni di template passibili di intercettazione. Sul mercato esistono ad oggi alcune implementazioni di SOC, anche se sono tutte lontane dalla tipica smart card: ad esempio Sony FIU-710 [117] (chiamata amichevolmente "Puppy") ottiene corrente dal collegamento seriale con l'host visto che non possiede batterie interne, anche se le sue dimensioni gli permettono di poter stare sul palmo di una mano.

4.5 Introduzione a Java Card™ Forum.

Lo scopo principale di Java Card™ Forum [88], nel seguito chiamato per semplicità JCF, e' quello di promuovere Java, linguaggio di programmazione per sviluppare applicazioni su smart card. Alla base di questa incentivazione si trovano due caratteristiche ereditate da Java "originale" e presenti anche nel suo sottoinsieme rappresentato da Java Card™ e che risultano essere importanti per questa tipologia di mercato: il modello di security, importante per un ambiente multi-applicazione che deve gestire spesso dati personali e informazioni bancarie e, secondo punto, la rapidità con cui queste applicazioni possono essere sviluppate per un mercato dato in forte crescita per il futuro.

La possibilità infatti di aggiungere, aggiornare o togliere applicazioni dalla stessa smart card rappresenta nel prossimo futuro un forte incentivo per questo mercato, dato che un unico supporto comune si ripercuote inevitabilmente al ribasso sui costi.

JCF ha come attività quella di promuovere le specifiche di Java Card™ API (*Application Programming Interface*) come standard industriale, prestando attenzione alle necessità dei costruttori di carte e dei loro clienti, rappresentati da compagnie finanziarie, di telecomunicazioni (specialmente reti di comunicazioni mobili GSM), che offrono servizi sanitari e dell'Information Technology in generale. Proprio per mantenere l'obiettivo che l'interesse delle aziende sia presente, JCF ha istituito lo *Strategic Partner Program*, un programma nel quale le compagnie interessate per vari motivi a questa tecnologia, sono invitate al processo di definizione dei requisiti. Attualmente, ben 18 compagnie partecipano a questo programma.

Il Forum e' stato fondato il 27 Aprile del 1997 da Schlumberger [89], Gemplus [90] e Bull CP8 Transac (ultimo membro ad aderire al progetto) dopo meno di un anno (Ottobre 1996) dal rilascio delle specifiche API Java Card™. A questo primo gruppo fondatore poi si sono aggiunte tra le altre Visa, American Express, Citibank, Hitachi, NEC e la stessa Sun a cui appartengono le specifiche di Java Card™ API.

Il Forum e' organizzato nella *Business Committee* e nella *Technical Committee* che si preoccupano rispettivamente delle necessità del mondo aziendale e degli aspetti tecnologici riguardanti questa tecnologia; questi due organismi si riuniscono all'incirca ogni due mesi e, come frutto di questo lavoro, hanno preparato anche documenti di raccomandazioni per le specifiche di Java Card™ 2.0 API di SUN Microsystems, attualmente rilasciate nella versione 2.2.

In questa ultima versione non sono fanno però parte le API introdotte da JCF.

Come viene direttamente annunciato nel *White paper* [95] che porta la data del 7 Agosto 2002, le API supportano il Match-On-Card sicuro, raccogliendo così sulla carta sia i dati privati, sia il sistema di sicurezza che li "sorveglia" e gestisce.

L'atto istitutivo di JCF prevede quindi di:

- ◇ Dare visibilità verso il mercato di nuove soluzioni riguardanti sistemi operativo per smart card, esaminare insistentemente gli standard già proposti, fornire nuove soluzioni per ambienti multi-applicazione .
- ◇ Promuovere l'interoperabilità di prodotti e applicazioni su smart card.
- ◇ Facilitare la creazione di prodotti per l'intera catena riguardante quindi smart card, terminali e applicazioni.

I membri di JCF hanno scelto di sviluppare le loro nuove famiglie di prodotti multi-applicazione utilizzando l'ambiente di sviluppo individuato dalle API Java Card™ con i seguenti obiettivi:

- ◇ Promuovere il linguaggio Java Card™ e gli standard associati verso il mercato delle smart card e verso gli sviluppatori di nuove specifiche.
- ◇ Preparare documenti tecnici per supportare Java Card™, come specifiche, API e così via.
- ◇ Scambiare informazioni tecniche riguardanti Java Card™ e sue implementazioni tra i membri.
- ◇ Proporre a SUN Microsystems qualsiasi nuova miglioria rispetto allo standard di specifiche alle API Java Card™.
- ◇ Fornire input tecnici, tramite implementazioni Java Card™, ai costruttori di chip per smart card.

- ◇ Incoraggiare il dialogo tra i fornitori di software che utilizzano Java Card™ API.

4.5.1 Caratteristiche delle API.

In questa sezione verranno presentate le Java Card™ Biometric API [95], che sono state sviluppate e in seguito accettate dai membri di *Java Card™ Forum Biometric Task Force*, un sotto gruppo di lavoro di JCF; queste API sono state inoltre presentate e hanno ricevuto un feedback da parte del *NIST Biometric Consortium Working Group* [94].

Queste API sono state utilizzate all'interno dell'algoritmo MOC da noi sviluppato e verranno perciò riprese nel capitolo 7.

Nella sostanza, sono state introdotte per l'interoperabilità di applet Java diversi verso algoritmi di riconoscimento biometrico: costituiscono in pratica lo strato di interfacciamento tra le applicazioni e gli algoritmi che gestiscono i dati biometrici personali sulla smart card; in questo caso si cerca appunto di standardizzare le specifiche per lo strato intermedio, facilitando e rendendo indipendente lo sviluppo di questi programmi che chiamano e offrono metodi comuni.

Applicazione e algoritmo di matching possono così essere eseguiti su qualsiasi card in grado di ospitare una JCVM.

Lo scopo è quindi quello di supportare lo sviluppo di algoritmi di MOC sulla carta raggiungendo un alto livello di sicurezza.

La documentazione disponibile diffusa da JCF non è ancora terminata del tutto ed alcune sezioni mancano ancora completamente, come viene indicato anche dal nome del *White Paper* che porta la dicitura finale *Working Document*.

Alcuni requisiti che sono stati individuati prima della realizzazione delle API da parte di JCF e che sono serviti da linee guida per la loro definizione, sono i seguenti:

- ◇ Supportare la capacità di registrare i dati biometrici (per esempio un template di una impronta digitale) in modo sicuro e di poter effettuare la verifica dell'identità senza esporre questi dati all'esterno della carta.
- ◇ Essere al tempo stesso semplici e compatte, ma devono poter adattare al loro interno un'ampia varietà di algoritmi di enrollment e matching realizzati dal mondo dell'industria.
- ◇ Consentire di registrare all'interno della carta più tipi di dati biometrici come impronte digitali, iride, geometria dell'orecchio, della mano, schema del DNA (ancora futuristico) ecc....
- ◇ Proteggere i dati biometrici per esempio controllando il numero di match falliti consecutivi come nelle *PIN API* di Java Card™.
- ◇ Riusare componenti già presenti in altre interfacce come per esempio la sopra citata PIN API, per mantenere un approccio simile ed ereditarne i punti di forza per quanto riguarda la sicurezza.
- ◇ Rispettare le specifiche già esistenti di GlobalPlatform [91], il cui maggior intento è rappresentato dal cercare di definire specifiche e infrastruttura per un ambiente multi-applicazione per le smart card (ad esempio *Cardholder Verification Method* o più brevemente CVM).
- ◇ Dove possibile, devono rispettare gli standard biometrici già esistenti come *BioApi* [92, 119, 120] o *Common Biometric Exchange File Format* (CBEFF) [93].

Fino a qualche anno fa le applicazioni biometriche venivano realizzate sulla base di specifiche API proprietarie, fornite dal costruttore dello strumento di acquisizione, insieme ad altre API necessarie al trattamento dell'informazione biometrica (costruzione del template, confronto, ecc...). In tale scenario, sistemi biometrici basati sulla stessa tecnologia, ma realizzati con dispositivi di acquisizione e algoritmi provenienti da fornitori distinti, erano totalmente incompatibili nel senso che le informazioni acquisite in un sistema non erano utilizzabili nell'altro. Per ovviare a questa mancanza si è cercato di creare un'interfaccia standard che consentisse di svincolarsi dal particolare dispositivo e proponesse una metodologia comune per il trattamento dell'informazione.

Le API in questione nascono con l'intento di agire da interfaccia tra uno o più algoritmi di matching e i vari applet on-card che fanno uso di biometria come viene mostrato nella parte destra della figura 4.12, tratta da [95]: questa figura riassume le interazioni tra diversi standard biometrici esistenti e la loro disposizione all'interno di un sistema completo di sensore di acquisizione, una postazione PC, un lettore di smart card ed infine una Java Card™. Nella figura si possono riconoscere alcuni standard come *ISO 7816-11: Personal verification through biometric methods* oppure BioApi [92, 119, 120], che propone un modello di programmazione ad "alto livello" adatto a tutte le tecnologie biometriche e basato sul concetto di *Biometric Service Provider* (BSP, vedi figura 4.13); esso fornisce le principali di acquisizione, enrollment, verifica e identificazione ed offre un insieme di primitive per la realizzazione di protocolli di riconoscimenti biometrico specializzati. Le informazioni biometriche provenienti dal dispositivo di acquisizione sono rappresentate nel BSP sotto forma di *Biometric Information Record* (BIR); il dispositivo restituisce un BIR "rozzo" (raw) che viene trattato dal BSP mediante una serie di algoritmi di ottimizzazione ed estrazione delle informazioni significative, per

produrre una serie di BIR intermedi fino al BIR finale pronto per l'uso (il formato del record di dati è standard, seguendo le regole di CBEFF).

Nello sviluppo delle Java Card™ Biometric API, la più grande sfida è stata quella di cercare di supportare il più alto numero di tecnologie e allo stesso tempo cercare di mantenerle semplici e con un consumo limitato di risorse, per consentire il loro impiego all'interno di una smart card: per questo le API realizzano solo le funzioni biometriche centrali.

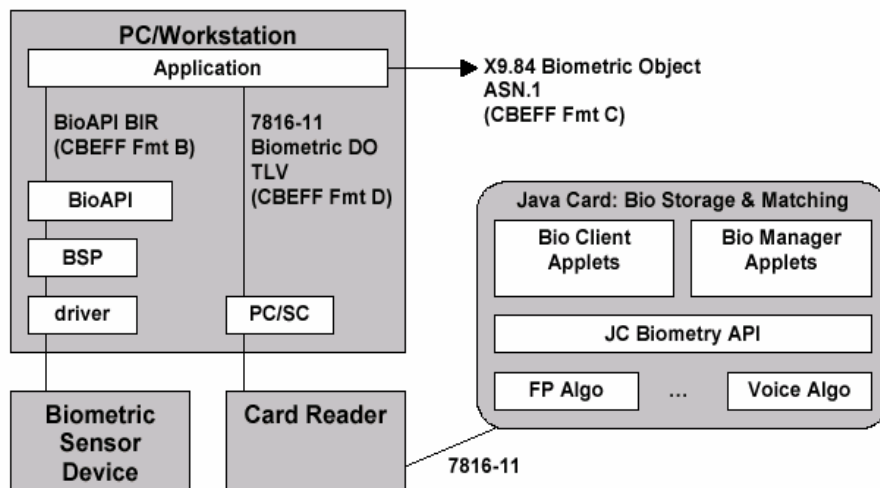


Figura 4.12
Interazione tra diversi standard biometrici esistenti [95].

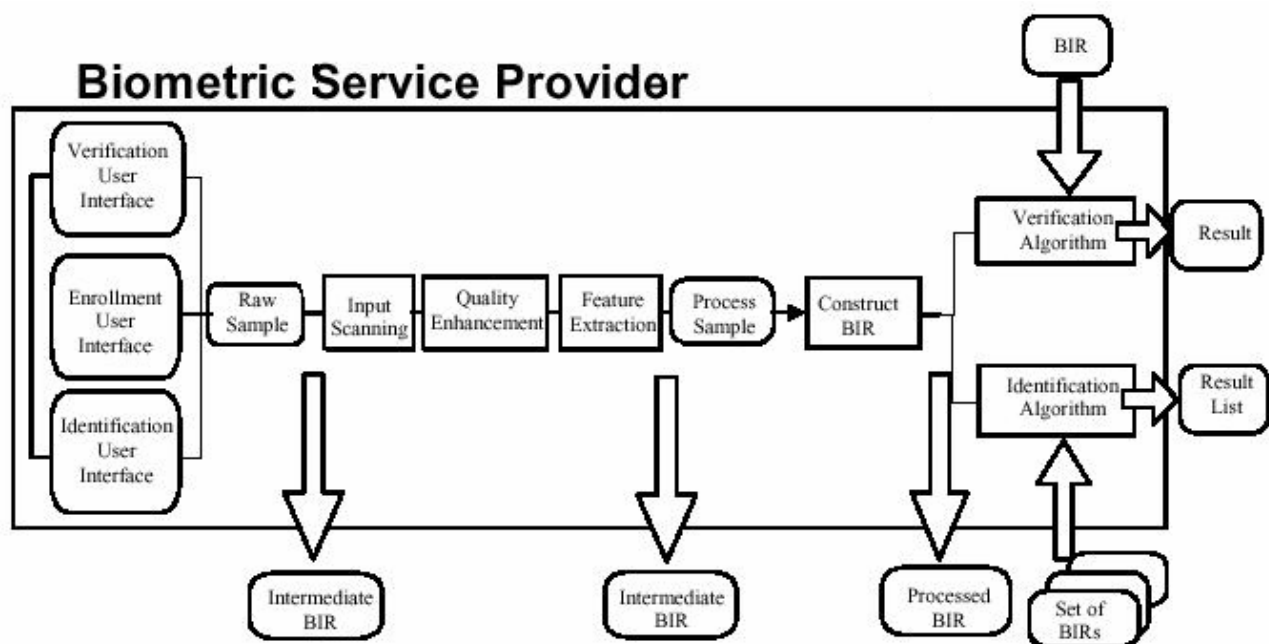


Figura 4.13
Architettura del Biometric Service Provider [120].

Capitolo 5

Un nuovo algoritmo di fingerprint matching.

5.1 Alcuni algoritmi esaminati.

Tra gli algoritmi che sono stati esaminati nella fase di raccolta informazioni scientifiche per questa tesi, oltre agli algoritmi ed ai lavori in generale del gruppo di A.K. Jain, in particolar modo [35], due si sono rilevati preziosi alla luce del lavoro svolto.

Il primo algoritmo serve per risolvere problemi di *Point Pattern Matching* [70]; più precisamente tale problema può essere descritto con la seguente definizione: dati due insiemi di punti in uno spazio d -dimensionale, deve essere determinato se esiste una trasformazione nello spazio che mappi il primo insieme sul secondo insieme di punti, anche se non precisamente, ma limitando le discrepanze. Il più delle volte le trasformazioni spaziali da utilizzare per il matching riguardano traslazioni o comunque movimenti “rigidi” e per quanto riguarda i metodi utilizzati per affrontare il problema, questi devono essere il più possibile robusti nei confronti di perturbazioni casuali, presenza o assenza di punti tra i due insiemi e differenze rispetto alla scala di grandezza.

Come si può quindi evincere dalle proprie caratteristiche, questo problema si incarna perfettamente nel nostro rappresentato dal matching di impronte digitali, anche se più in generale può essere incontrato in un'ampia scelta di casi, come nella biologia computazionale (come nell'allineamento della struttura delle proteine), nella chimica computazionale, nell'analisi di documenti ed infine anche in ambito aerospaziale: l'Istituto Aerospaziale e Astronautico della Università di Berlino, utilizza il matching di punti per stabilire il comportamento dei satelliti della serie TUBSAT, portati nello spazio con i progetti Ariane. Il satellite è equipaggiato con una telecamera che raccoglie immagini del cielo stellato, che vengono in seguito comparate con altre immagini di catalogo per decidere la posizione e l'orientamento da seguire.

Inoltre tale problema, che non è esclusivamente rivolto a due sole dimensioni, ma anche a punti collocati nello spazio, può trovare applicazione anche nel campo del riconoscimento dei caratteri [71], come illustrato in figura 5.1 (da [71]).

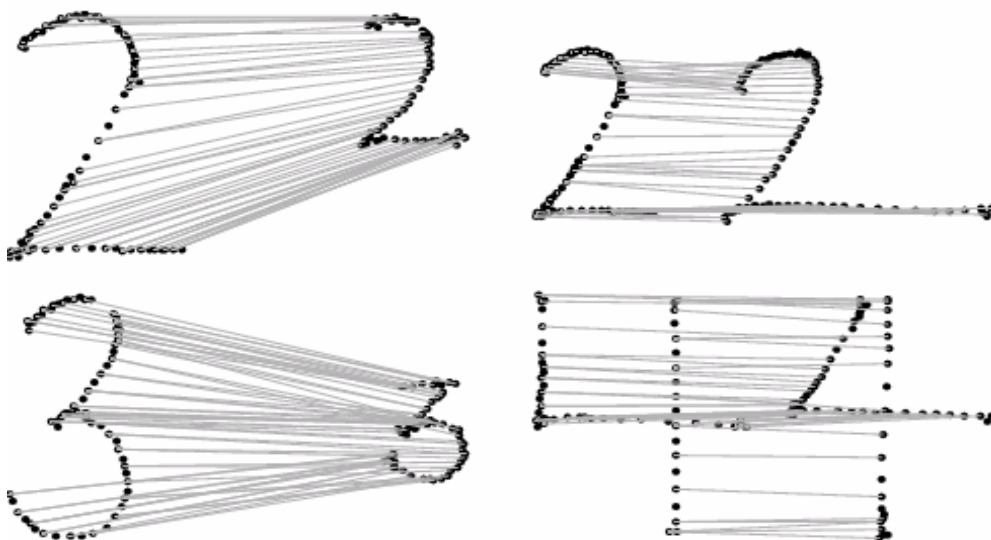


Figura 5.1
Esempio di riconoscimento dei caratteri [71].

Un algoritmo che abbiamo analizzato più approfonditamente per la risoluzione del point pattern matching e quindi anche del matching di impronte digitali, è descritto in [72] ed è rivolto alla comparazione di due “liste di stelle” rappresentate dalle loro coordinate in due dimensioni; questo matching tiene conto dei problemi che interessano la collocazione delle minuzie nelle impronte. Applicato al problema dell’accoppiamento di minuzie l’algoritmo affronta infatti traslazioni, rotazioni, differenze di scala, perturbazioni casuali e punti di singolarità in più o in meno.

Anche in questo algoritmo, come in quello citato successivamente, viene introdotto il concetto di vista del mondo circostante (“world view”) rispetto ad un determinato punto preso in considerazione. Per ogni punto viene determinato “l’effetto” che tutti gli altri punti determinano su di esso; questo effetto può essere rappresentato dalla distanza euclidea tra i due e, nel caso di liste di stelle, pesata in un certo modo con la magnitudine della stella.

A questo punto, tutti gli $n-1$ valori ottenuti (se la lista di punti è di n elementi), dopo la normalizzazione, vengono interpolati insieme se cadono all’interno spicchio di visuale: questa viene così campionata in 360 intervalli di 1 grado ciascuno e così viene definita da un vettore di 360 posizioni. A questo, ripetendo anche il procedimento per la seconda lista, le due impronte digitali possono essere comparate insieme tramite la comparazione delle viste dei propri punti.

Dal secondo algoritmo [69], che a sua volta deriva da [73, 74], è stato invece derivato il concetto di “panorama” che vede una minuzia centrale rispetto ad alcuni suoi vicini, che sono in realtà anch’esse delle minuzie: questo punto è stato infatti rivisto e utilizzato anche per la definizione del nostro algoritmo.

Nel primo stadio del calcolo infatti, viene tracciato un cerchio di diametro 300 pixel che ha per centro una minuzia (in questa situazione ritenuta centrale); all’interno

di quest'area vengono individuate le 4 minuzie più vicine e vengono marcate come vicini della corrispettiva minuzia centrale. Se all'interno del cerchio ci sono meno di 4 vicini, la minuzia non viene più presa in considerazione; tale procedimento viene ripetuto per ciascun punto singolare del template di riferimento, mentre per quello candidato si segue lo stesso principio, ma vengono rilevati tutti i vicini e quindi non c'è limitazione a soli 4.

Le caratteristiche con le quali vengono descritti i vicini di una minuzia centrale sono raccolte in un vettore (il vettore delle feature) di 4 posizioni, che serviranno poi negli stadi successivi; in [73] si utilizzano solamente le prime 3 delle successive caratteristiche, ma l'aggiunta della quarta si è rivelata importante sperimentalmente per l'abbassamento della False Acceptance Rate.

- ◇ Tipo della minuzia vicino (terminazione o biforcazione).
- ◇ Distanza euclidea tra la minuzia centrale ed il vicino.
- ◇ *Distance-relative angle*: questo è l'angolo tra la linea congiungente la minuzia centrale ed il suo vicino, e l'angolo della cresta su cui si trova la minuzia centrale.
- ◇ *Relative ridge angle*: angolo ottenuto sottraendo l'angolo della cresta su cui si trova la minuzia centrale e quello su cui si trova la minuzia/vicino.

Questo vettore di feature risulta quindi essere immune da rotazioni e traslazioni ed inoltre le stesse caratteristiche ottenute per minuzie vicine sono meno affette da distorsioni, data la breve distanza dalla minuzia centrale.

Della seconda parte dell'algoritmo, nel nostro riprendiamo soltanto lo “spirito” dell'obiettivo da raggiungere, ma non sicuramente i calcoli per realizzarlo: si cerca infatti di comparare in qualche modo tra loro le visioni che le minuzie dei due template hanno dei propri vicini per cercare di accoppiare quelle che lo hanno simile, per confrontare appunto quanto le due impronte siano differenti. Nel sopra citato algoritmo questo stadio inizia decidendo se una certa minuzia centrale di riferimento può essere ritenuta una papabile candidata al match con un'altra nell'altro template, se almeno uno dei propri vicini ha caratteristiche (feature) vicine a uno dell'altra: viene trovata la minuzia di riferimento nell'altro template che ha il vettore delle feature di un vicino più simile al vettore di un vicino della minuzia nel primo template (quello di riferimento); in questo modo, una minuzia nel template candidato può essere fatta corrispondere a più minuzie del template di riferimento.

A questo punto ci troviamo con due paia di punti per ogni minuzia del riferimento: la minuzia centrale nel test (X_1, Y_1) e il proprio miglior vicino corrispondente (X_2, Y_2) formano la prima coppia, mentre la seconda raggruppa ovviamente la minuzia centrale nel riferimento (x_1, y_1) ed ancora il proprio miglior vicino corrispondente (x_2, y_2), questa volta a quello della minuzia centrale nell'immagine di test.

Adesso, ciascuno degli altri tre vicini (x_3, y_3) della minuzia centrale nel template di riferimento (minuzia riferimento) verrà deformato in base al vicino (X_3, Y_3) della minuzia centrale nel template candidato (minuzia candidata), questo perché supponiamo appunto che le due precedenti coppie di punti siano corrispondenti. Le formule per questo passo sono del tipo (coefficienti A, B, GX, GY sono precisati in [69]):

$$X_3 = A \cdot x_3 + B \cdot y_3 + GX \quad Y_3 = -B \cdot x_3 + A \cdot y_3 + GY$$

Dopo questo passo è possibile cercare di accoppiare le due minuzie in maniera più accurata: basta osservare se le tre coppie di vicini delle due minuzie sono “abbastanza” vicine tra loro (threshold da stabilire) e attribuire in tal caso un punto, fino al massimo quindi di tre punti per coppia di minuzie. Contando alla fine il totale di questi punti è possibile determinare, attraverso un altro threshold, se le due impronte possono rappresentare la stessa identità.

5.2 Descrizione dell’algoritmo.

5.2.1 Teoria di base.

Il nostro algoritmo di matching si basa su delle caratteristiche misurabili dei vicini delle minuzie estratte attraverso l’uso di un precedente passo di calcolo estraneo alla trattazione di questo paragrafo. Nella nostra ricerca abbiamo individuato quattro elementi descrittivi che delineano come la minuzia vicina viene vista dalla minuzia centrale che stiamo prendendo in considerazione in una certa fase; la collezione di queste *feature* ripetuta per un certo numero di vicine, rappresenta una visione del mondo circostante che può essere comparata con quella di altre per cercare di fare matching di minuzie dalle quali si osserva un “panorama” simile; questa idea di “visione del mondo” circostante è stata ispirata dall’algoritmo in [72], mentre l’aiuto delle *feature* è provenuto da [73], anche se tali caratteristiche non coincidono completamente con quelle del nostro algoritmo.

È quindi chiaro che più i panorami di due minuzie prese da due immagini diverse sono simili, più è possibile che le due minuzie corrispondano per l'impronta dello stesso polpastrello; cercando di trovare questa similarità in tutte le coppie di minuzie delle due immagini, è possibile ottenere una misura della probabilità di quanto le due impronte possano coincidere. Queste quattro *feature* sono formate da:

- ◇ La distanza euclidea tra minuzia centrale A e la minuzia vicina B, rappresentata nella figura 5.2 dal segmento D .
- ◇ L'angolo compreso tra la direzione del *ridge* (o cresta) su cui si trova la minuzia centrale e la retta congiungente tale minuzia e il suo vicino: α figura 5.2.
- ◇ La differenza tra gli angoli che individuano la direzione della cresta su cui si trova la minuzia centrale e la direzione della cresta su cui si trova il vicino: nel caso della figura 5.2, è calcolata la differenza tra θ_1 e θ_2 (possiamo chiamarla β).
- ◇ Il numero di *ridge* che intercorrono tra minuzia centrale e vicino: un esempio, con valore uguale a uno, è mostrato in figura 5.2 e 5.2.

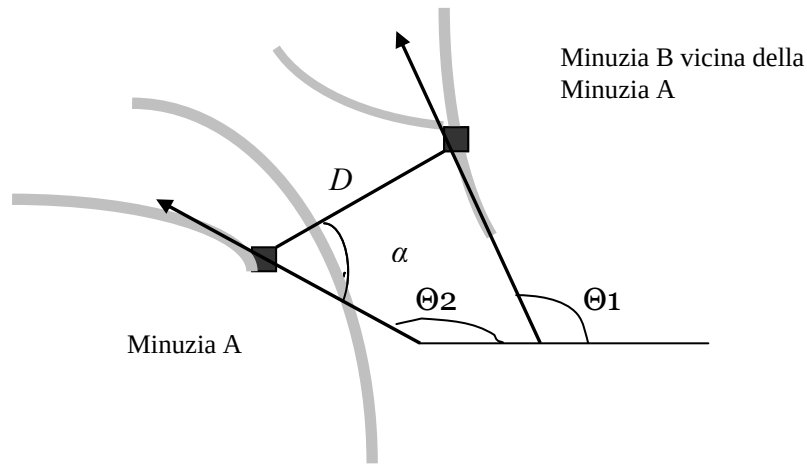


Figura 5.2
Rappresentazione grafica delle quattro feature.

Nelle figure 5.2 e 5.3, i due quadratini indicano la minuzia centrale A e uno dei suoi vicini rappresentato dalla minuzia B. Nella figura 5.3 sono mostrate solo le creste e le minuzie di figura 5.2 per rendere più chiara la spiegazione delle feature.

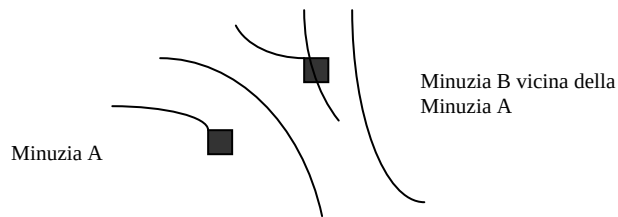


Figura 5.3
Rappresentazione di una minuzia e di una suo vicino.

L'uso di queste feature rende il problema immune da traslazioni in quanto la visione del mondo circostante traslata in un'altra posizione dello spazio rimane esattamente uguale; inoltre le quattro feature rimangono invarianti anche rotazione,

visto che distanza e ridge count ruotando non cambiano e neanche i due angoli, che ruotano insieme alla rotazione globale. Inoltre, visto che queste caratteristiche vengono calcolate per minuzie vicine rispetto alla centrale, eventuali distorsioni globali dell'immagine vengono smorzate: localmente alla minuzia centrale, il “rumore” provocato dalla distorsione è notevolmente inferiore.

Il problema di matching di impronte digitali è stato impostato in questo caso attraverso un algoritmo di accoppiamento di costo minimo tra i nodi di un grafo bipartito [30, 31, 32]. Su tale grafo bipartito possono essere affrontati tre tipi di problemi molto comuni dei quali uno risulterà utile nel nostro caso:

- ◇ *Accoppiamento di massima cardinalità* : si vuole determinare un accoppiamento di massima cardinalità in G e cioè col numero di archi, e conseguentemente di nodi, più alto possibile.
- ◇ *Assegnamento di costo minimo*: si vuole determinare, tra tutti gli accoppiamenti perfetti in G , uno che abbia di costo minimo. Tra le sue varianti è possibile ricercare sul grafo l'assegnamento di costo minimo di massima cardinalità: in questo caso si cerca di accoppiare il numero massimo di nodi possibile, minimizzando contemporaneamente il costo dell'accoppiamento.
- ◇ *Assegnamento di massima cardinalità bottleneck*: si vuole determinare, tra tutti gli accoppiamenti di massima cardinalità in G , uno che abbia valore bottleneck minimo, cioè tale che il massimo costo degli archi sia minimo.

I concetti esposti precedentemente sono quindi adattabili ad un problema di accoppiamento di minuzie: i due insiemi di nodi del grafo bipartito rappresentano i due insiemi di minuzie del template di reference, contenente i dati registrati durante l'enrollment, e del template candidato per il quale vogliamo misurare la “somiglianza” con il primo. Il nostro algoritmo cercherà di affrontare e risolvere il problema posto al secondo punto e cioè quello rappresentato dal trovare un assegnamento di costo minimo e cardinalità massima sul grafo che rappresenta i due template: oltre ad ottenere un costo basso, miriamo anche a cercare di accoppiare il più alto numero di minuzie possibile, poiché è necessario per ottenere una vera identificazione.

Il primo passo da affrontare è costituito dalla costruzione di tale grafo: abbiamo già detto che i nodi rappresentano nel nostro caso le minuzie dei due template che vogliamo comparare e quindi molte possibilità sono lasciate nella fase di implementazione che viene trattata nei capitoli finali. Chiaramente in questa prima fase è possibile risparmiarsi del calcolo in quelle successive cercando di inserire archi solo tra minuzie che hanno una qualche possibilità di essere accoppiate, per esempio, minuzie dello stesso tipo: terminazioni di creste con terminazioni e biforcazioni di creste con biforcazioni. Per continuare nella descrizione teorica però, è sufficiente pensare che ci sia semplicemente un arco tra qualsiasi coppia di minuzie dei due template.

Adesso rimane solo da attribuire un costo a ciascun arco inserito nel grafo. Dato che vogliamo eseguire sul grafo un algoritmo di accoppiamento di costo minimo, il costo di un arco deve rappresentare una “distanza” tra le due minuzie-nodi in cui incide e quindi più queste minuzie saranno “simili”, più questo costo dovrà essere basso.

Per trovare tale distanza, eseguiamo accoppiamento perfetto di costo minimo, ma questa volta, su un grafo bipartito nel quale i due insiemi rappresentano i vicini della prima e della seconda minuzia che stiamo prendendo in considerazione. Dato il basso

numero di questi, viene inserito per ogni nodo un arco che lo connette a ciascuno nel secondo insieme: in totale, dato che per ogni minuzia vengono trovati cinque vicini, vengono raggiunti i 25 archi; mettere meno archi in questo caso non avrebbe molto senso.

Il risultato alla fine è quindi un accoppiamento perfetto di costo minimo sui vicini delle due minuzie: in pratica stimo in un numero (la somma dei costi degli archi dell'accoppiamento) quanto i due “vicinati” siano simili tra loro e quindi quanto le due minuzie che possiedono tali vicini possano coincidere.

Come ultimo passo, rimane da stabilire solo come viene trovato il costo dell'arco che unisce due vicini di minuzie diverse: questo viene calcolato grazie alle differenze tra le corrispettive feature (istanza euclidea con distanza euclidea, ecc..) dei due vicini descritte ad inizio capitolo, che ripetiamo, sono la distanza tra minuzia centrale e vicino, i due angoli che individuano l'orientamento del vicino rispetto alla minuzia e il numero di creste che intercorrono tra le due (ridge count). Queste quattro differenze costituiscono quindi una misura di quanto differiscono le visioni delle due minuzie centrali rispetto ai propri vicini; queste informazioni possono essere a questo punto trattate in maniera differente: per esempio possono essere lasciate in un vettore di quattro posizioni e questi vettori possono essere comparati tra loro con un concetto di ordinamento totale per decidere quale siano le visioni più simili tra loro (c'è bisogno in realtà anche di un concetto di somma tra questi vettori per poter trovare i cammini minimi necessari all'algoritmo). Invece di lasciarle in un vettore, le quattro differenza possono invece essere normalizzate ed in seguito, grazie a dei pesi, sommate in un unico valore rappresentativo della distanza tra le due visioni. Il passo di normalizzazione (per esempio dividendo gli angoli per 360 gradi e gli altri valori per i massimi riscontrati) è necessario per rendere tutte le grandezze tra loro omogenee, mentre quello di pesatura

serve ad attribuire maggiore rilevanza alle differenze che rendono il match più affidabile; in questo secondo caso, il costo dell'arco congiungente i due vicini è rappresentato da un unico valore invece di un vettore di quattro posizioni.

Ricapitolando, questa volta partendo nella descrizione dal basso verso l'alto, viene eseguito l'algoritmo di assegnamento perfetto di costo minimo sul grafo che rappresenta i due vicinati delle due minuzie che stiamo prendendo in considerazione: il risultato è la somma dei costi degli archi che formano l'assegnamento, ed è anche una stima della somiglianza dei due vicinati (che può essere calcolata in maniera differente), dato che più che il risultato è basso, più che i due vicinati possono appartenere alla stessa minuzia.

Questo risultato rappresenta anche il costo dell'arco tra le due minuzie di cui abbiamo esaminato i vicinati, nel grafo delle minuzie e viene ripetuto più volte per completare l'intero grafo.

Una seconda esecuzione dello stesso algoritmo di assegnamento di costo minimo di massima cardinalità, restituisce come risultato un valore che rappresenta il costo dell'assegnamento e quindi anche una stima della somiglianza delle due impronte digitali: a noi interessa infatti un riconoscimento di tipo "uno a uno".

Modifiche alla procedura possono consistere in uno "sfoltimento" di archi nei due tipi di grafi presenti (minuzie-minuzie e vicini-vicini) oppure trovando altre feature da aggiungere o sostituire alle quattro già delineate, come per esempio l'area formata poligono individuato dai vicini (figura 5.4), o anche la distanza euclidea tra i vicini delle due minuzie riportati nello stesso sistema di riferimento.

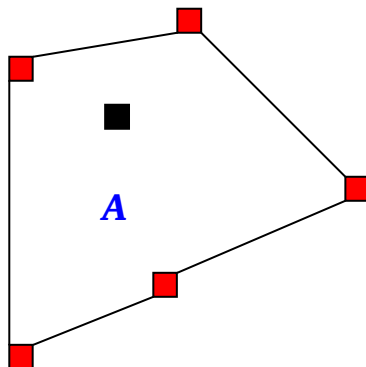


Figura 5.4
Possibile ulteriore feature: area del poligono individuato dai vicini.
Quella dentro il poligono è la minuzia centrale dei cinque vicini.

Capitolo 6

Implementazione e prestazioni su PC.

Nel corso di questo capitolo parleremo della fase implementativa della tesi: in pratica discuteremo del programma MINDTCT usato per la rilevazione delle minuzie, delle scelte fatte per la realizzazione dell'algoritmo di fingerprint matching in linguaggio Java™, delle caratteristiche di alcuni database di impronte ed infine illustreremo le prestazioni ottenute dalla nostra implementazione.

6.1 MINDTCT.

Nel corso della documentazione sull'argomento effettuata nella prima parte del progetto di tesi, sono state analizzate con attenzione anche le funzionalità offerte dal programma MINDTCT dell'*Image Group* del NIST.

Dato che questo software rappresenta l'unico esempio di codice di pubblico dominio per la rilevazione dei punti di singolarità (minuzie) all'interno delle impronte

digitali, lo abbiamo adottato nel nostro progetto per ricavare le informazioni necessarie alla fase di matching, verso la quale si rivolge più strettamente questa tesi.

Più precisamente il software rilasciato prende il nome di *Nist Fingerprint Image Software* (NFIS) e contiene la tecnologia sviluppata per il *Federal Bureau of Investigation* (FBI) [39] americano; lo scopo di questo pacchetto è quello di supportare e facilitare la manipolazione automatica e l'elaborazione delle immagini riguardanti impronte digitali: viene messo quindi a disposizione il codice sorgente di più di 50 utility diverse attraverso un supporto Compact Disc da richiedere direttamente al NIST [126].

Più dettagliatamente, le utility facenti parte del NFIS possono essere classificate in 4 categorie generali:

- 1) Un classificatore di impronte basato su reti neurali e che prende il nome di PCASYS: il suo compito è quello di assegnare all'impronte una classe tra arch, loop sinistro o destro, tented arch oppure whorl (vedi capitolo 2).
- 2) Un rilevatore di minuzie chiamato appunto MINDTCT, che automaticamente rileva e registra le biforcazioni e le terminazioni delle creste all'interno dell'impronta; per ognuna di queste fornisce anche una stima dell'affidabilità della minuzia, basata sulle caratteristiche locali dell'immagine. Questo software viene usato da *Universal Latent Workstation* (ULW) [127], che descrive una stazione di lavoro (basata su Windows) per effettuare ricerche di impronte digitali. Insieme al precedente PCASYS, questo programma rappresenta l'unica alternativa possibile (secondo le nostre conoscenze) nel panorama dei prodotti di pubblico dominio.

- 3) Un'implementazione di riferimento dello standard ANSI/NIST-ITL 1-2000 "*Data Format for the Interchange of Fingerprint, Facial, Scar Mark & Tattoo (SMT) Information*" [128]. Questa implementazione contiene una suite di utilità per permettere letture, scritture, modifiche ed in generale manipolazioni su file formattati secondo questo standard per lo scambio di informazioni e approvato nel Luglio del 2000 aggiornando e revisionando le precedenti versioni ANSI/NIST-CSL 1-1993 e ANSI/NIST-ITL 1a-1997: rispetto a queste sono state introdotte o aggiornate alcune parti del file come i Record Type 9, 13, 14 e 15.
- 4) Un'ampia collezione di utility per lavorare sulle immagini, come encoder e decoder per formati di tipo Baseline JPEG, Lossless JPEG o *Wavelet Scalar Quantization* (WSQ) di FBI.

Il documento ANSI/NIST-ITL 1-2000 è stato approvato nel Luglio del 2000 da *American National Standards Institute* (ANSI) come un *American National Standard*. Questo standard definisce il contenuto ed il formato per lo scambio di impronte digitali, impronte palmari, impronte facciali, cicatrici, voglie e tatuaggi che possono essere utilizzati nel processo di identificazione di un soggetto; questa informazione consiste in una varietà di campi obbligatori e opzionali che includono parametri di scansione, parametri che descrivono l'informazione contenuta e anche immagini compresse o non compresse. L'obiettivo di questa informazione è quello di essere scambiata tra le numerose stazioni che amministrano la giustizia o comunque tra organizzazioni che utilizzano sistemi automatici di verifica dell'identità (AFIS) basati sulle sopra citate tecnologie biometriche.

I dati vengono suddivisi in record logici (in totale sedici dei quali due riservati per uso futuro) e il primo contiene informazioni sulla transazione del file che deve avvenire tra due stazioni remote differenti; in alcuni record, come il primo, l'informazione viene codificata in ASCII, mentre in altri come il quarto, che contiene l'immagine ad alta definizione e in scala di grigio dell'impronta digitale, i dati sono rappresentati in binario.

I valori delle feature utilizzate dal nostro algoritmo di matching (vedi paragrafo 5.2) sono calcolati a partire dalle informazioni estratte dall'applicazione MINDTCT modificata e salvate da questa, oltre che nel record Type-9 (il nono record) [29] conservato in un file conformato dello standard ANSI/NIST, anche in un file di testo ASCII. In pratica l'applicazione prende in input da riga di comando un file *an2k* contenente l'immagine in binario da cui estrarre le minuzie e salva in un nuovo file *an2k* le vecchie informazioni aggiungendo in un altro campo le minuzie estratte. Il record Type-9 (*Type-9 minutiae data record*) rappresenta lo standard per la memorizzazione dell'informazione riguardante un'impronta digitale all'interno del file di estensione *an2k* (in pratica, le sue minuzie).

I file con estensione *an2k* rappresentano l'implementazione di riferimento dello standard ANSI/NIST-ITL 1-2000; NFIS contiene anche diversi tool per leggere, scrivere e manipolare il contenuto di questi file: per esempio l'applicazione *an2k2txt* converte il contenuto testuale di un file ANSI/NIST in un rapporto in testo formattato. L'informazione per minuzia ricavata dal programma segue il seguente schema:

MN : MX, MY : DIR : REL : TYP : FTYP : FN : NX1, NY1; RC1 :

Dove:

MN è l'intero che identifica, univocamente rispetto all'estrazione,

la minuzia rilevata.

MX è la coordinata x della minuzia rilevata .

MY è la coordinata y della minuzia rilevata.

DIR l'angolo tra la direzione della minuzia rilevata e l'asse verticale.

REL è una misura dell'affidabilità assegnata alla minuzia rilevata.

Il risultato consiste in un reale con valore compreso tra 0.0 e 1.0
che rappresentano rispettivamente il minimo ed il massimo.

TYP è il tipo della minuzia rilevata

bifurcation = "BIF"

ridge ending = "RIG".

NX1 è la coordinata lungo x della prima minuzia del vicinato.

NY1 è la coordinata lungo y della prima minuzia del vicinato.

RC1 è il numero di creste che separano dalla prima minuzia del
vicinato.

..... I parametri NX, NY e RC vengono ripetuti per ogni minuzia
del vicinato; di default l'algoritmo di MINDTCT trova 5 vicini
per minuzia .

FTYP e FN sono altri parametri interni all'algoritmo MINDTCT e non sono richiesti dall'algoritmo di match.

Avendo la posizione della minuzia e del suo vicino è quindi facile trovare la prima *feature* della distanza euclidea D in figura 5.2, mentre la seconda e la terza (α e β in figura 5.2) sono calcolabili a partire dall'orientamento DIR della minuzia centrale e quella del vicino. La quarta feature, il ridge count, è direttamente calcolato da MINDTCT.

La locazione di ciascuna minuzia è rappresentata dalle coordinate all'interno dell'immagine dell'impronta. Lo standard ANSI/NIST specifica la distanza a partire dal vertice in alto a sinistra in unità di 0.01 millimetri. Per esempio un'immagine con 500x600 pixel scannerizzata a 19.69 ppmm risulta avere dimensioni di 25.39x20.47 millimetri che in termini di unità di 0.01 mm diventano

$$2539 \times 3047 = \frac{500}{19.69 \times 0.01} \times \frac{600}{19.69 \times 0.01}$$

con la coordinata y crescente verso il basso e la x verso destra.

L'orientamento della minuzia calcolato dall'applicazione *MINDTCT* è rappresentato in gradi, con l'angolo nullo collocato orizzontalmente e con verso rivolto a destra e aumentante in senso anti-orario. L'orientamento per una minuzia di tipo "ridge ending" è calcolato misurando l'angolo tra l'asse orizzontale e la linea che parte dalla minuzia e che corre verso il ridge. L'orientamento per una minuzia "bifurcation" è

calcolato misurando l'angolo tra l'asse orizzontale e la linea che parte dalla minuzia e che corre verso la valle individuata dalla biforcazione nelle due creste. In pratica l'angolo calcolato corrisponde a quello etichettato con B (che equivale anche agli angoli θ_1 e θ_2 di figura 5.2) nelle due figure 6.1 *a)* e *b)*, che rappresentano rispettivamente una minuzia di tipo “ridge ending” e una “bifurcation”.

Nelle immagini sottostanti (tratte dal manuale del MINDTCT) sono rappresentate una minuzia di tipo “ridge ending” e una di tipo “bifurcation”; la “coda” che si diparte dal cerchietto nel primo caso e dal quadrato nel secondo, indica l'orientamento della minuzia e gli angoli segnati con la lettera B rappresentano gli angoli calcolati da MINDTCT.

Nella terza figura è mostrata un'impronta intera con tutte i punti di singolarità trovati.

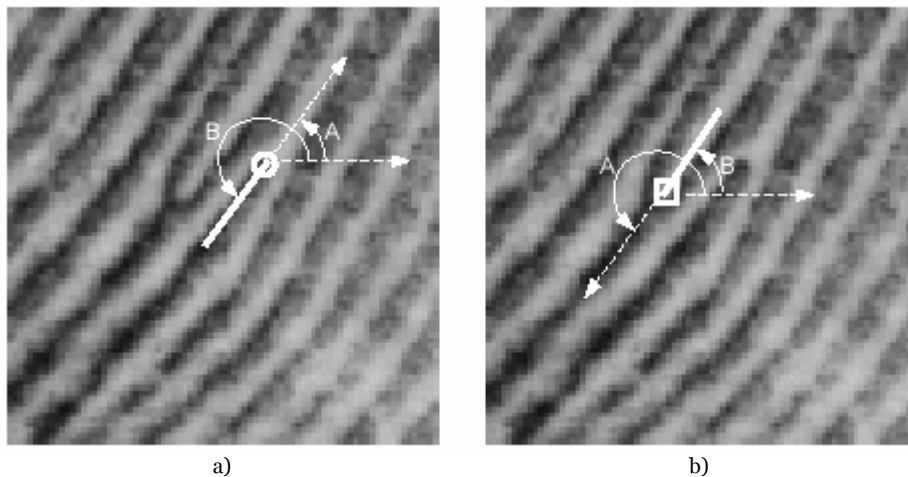


Figura 6.1
Angoli trovati dal programma MINDTCT e che rappresentano la direzione di due minuzie di tipo “bifurcation” e “ridge ending” [45].



Figura 6.2
Tutte le minuzie trovate da MINDTCT su una impronta digitale
con la direzione segnata da un segmento [45].

In figura 6.2 viene mostrato un esempio di tutte le minuzie trovate dal programma: i quadratini evidenziano il punto della minuzia, mentre la “coda” del quadrato mostra la direzione della cresta relativa.

Nella figura 6.3 sono riportate due impronte diverse dello stesso polpastrello con alcune minuzie “false” rilevate da MINDTCT. Le minuzie false non sono vere minuzie caratteristiche dell'impronta digitale, ma vengono create durante il loro processo di acquisizione: possono derivare per esempio dal taglio dell'immagine che viene effettuato per evitare di elaborare parti di essa non significative.

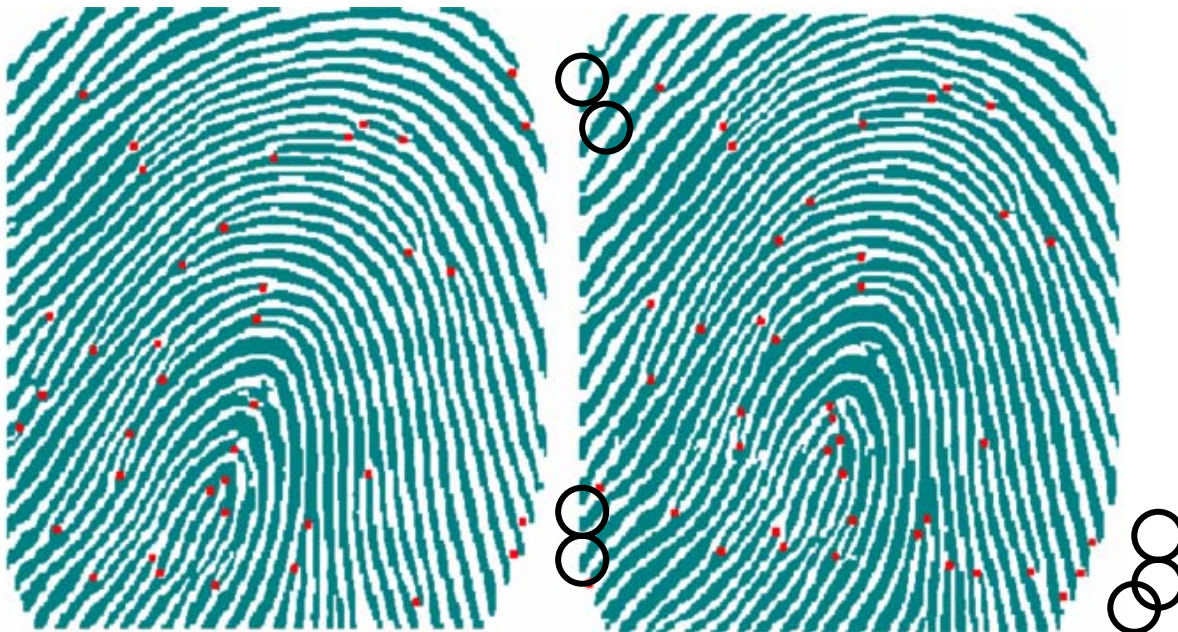


Figura 6.3
 Due impronte diverse dello stesso dito con alcune minuzie rilevate dal programma MINDTCT: si possono notare come vengano rilevate anche delle “false” minuzie sul bordo dell’immagine tagliata; la loro affidabilità sarà comunque più bassa rispetto alle altre. Le immagini sono due acquisizioni diverse dello stesso dito: si può notare come le minuzie “false” possono non ripetersi.

Per poter capire di quali informazioni aggiuntive potevano usufruire rispetto a quelle già fornite direttamente dal programma MINDTCT, è stato necessario esaminare approfonditamente il codice, che è stato successivamente modificato in base alle nostre esigenze.

Il principale motivo che ci ha imposto delle modifiche, si può attribuire al fatto che originariamente i vicini di una minuzia venivano scelti solo tra le minuzie successive nella lista trovata e per questo le ultime ne avevano un numero più basso: dalla quint’ultima che ne aveva quattro, all’ultima che ne aveva zero; nel nostro algoritmo abbiamo invece bisogno che la procedura trovi i 5 vicini più prossimi in assoluto. Ecco quindi che è stato necessario modificare il codice per inserire nella lista dei vicini i primi 5 con distanza minore dalla minuzia centrale.

Inoltre è stato modificato il codice per ottenere solo vicini con un'affidabilità maggiore di un certo valore, oppure modificare il loro numero dagli standard 5: in questo modo è possibile alleggerire il lavoro del nostro algoritmo di matching scartando direttamente nel programma MINDTCT i vicini con una bassa affidabilità oppure aumentare il loro numero sopra 5, in modo da rendere l'algoritmo più preciso a scapito del costo computazionale. In Appendice B sono riportati i dettagli della modifica al codice.



Figura 6.4
Diagramma del processo di rilevamento delle minuzie in MINDTCT [45].

In figura 6.4 sono riportate le fasi del calcolo di MINDTCT così come vengono descritte nel manuale del NIST [45].

6.2 Scelte implementative dell'algoritmo su PC.

Come già accennato nel paragrafo 5.2.1, in fase di implementazione è stato possibile modificare numerosi parametri per cercare di rendere il nostro algoritmo più veloce e più affidabile. A tal fine è stata creata la classe (ricordiamo che su PC l'algoritmo è stato programmato in linguaggio Java) *checkEdge*, il cui compito è proprio quello di decidere se inserire un arco tra due minuzie avendo come input proprio le due minuzie: diminuendo il numero degli archi infatti, si diminuisce di conseguenza anche il calcolo necessario per trovarne il costo, che corrisponde ad una esecuzione dell'algoritmo di assegnamento di costo minimo sul grafo dei vicini (vedi paragrafo 5.2.1); oltre a questo, si risparmia tempo anche nell'esecuzione dell'assegnamento di costo minimo anche sul grafo delle minuzie, dato che ne viene diminuito il numero degli archi e quindi la sua esplorazione risulta più semplice. La decisione se includere l'arco o meno può dipendere dal fatto che le due minuzie appartengano allo stesso tipo (terminazione o biforcazione), non siano troppo distanti tra loro rispetto alle coordinate trovate, non differiscano troppo rispetto all'affidabilità, dato che rispecchia quanto MINDTCT sia sicuro della loro esistenza, oppure altre varianti o altri fattori secondari. Nell'implementazione finale dell'algoritmo è stato scelto di non inserire l'arco se l'affidabilità di una delle due minuzie è inferiore a 0.45.

D'altra parte, nell'eliminare archi e quindi possibili accoppiamenti tra minuzie c'è anche la possibilità di eliminarne anche alcuni che connettono minuzie tra loro

corrispondenti e quindi pregiudicare la precisione della verifica biometrica. Questo trade-off deve essere affrontato anche nel caso della scelta del numero di minuzie massimo da considerare nel funzionamento dell'algoritmo: nella scelta finale da noi effettuata vengono prese le prime 25 minuzie per affidabilità da tutte quelle trovate dal programma MINDTCT: in questo modo si riducono notevolmente i tempi di calcolo e si cerca di mantenere una certa affidabilità da parte dell'informazione utilizzata. Da notare che MINDTCT può rilevare anche un numero di minuzie molto basso (tipo 8-10) se l'immagine dell'impronta è molto compromessa.

Per quanto riguarda la normalizzazione delle differenze delle feature, abbiamo deciso di utilizzare il massimo valore di ridge count rilevato da MINDTCT per quanto riguarda la differenza tra ridge count, il massimo di tutte le distanze minuzia centrale-vicina per la distanza euclidea ed il valore di 360 per quanto riguarda le due differenze tra angoli; per i primi due parametri è stato utilizzato il massimo tra quello rilevato sulla minuzia di riferimento e quello rilevato sulla minuzia candidata per il matching.

Altri parametri che riguardano pesantemente l'ottimizzazione delle prestazioni sono rappresentati dai pesi da attribuire alle sopra citate differenze tra feature (dopo la normalizzazione) nella somma che porta alla definizione di un solo valore di riassunto delle visioni dei vicinati e che costituisce in definitiva il peso dell'arco nel grafo dei vicini: modificando questi parametri è possibile dare più importanza a feature che riescono a rendere il match più affidabile (vedi paragrafo 6.3).

Per quanto riguarda il valore che l'algoritmo di assegnamento trova per il grafo dei vicini, che costituirà poi il valore dell'arco congiungente due minuzie, abbiamo deciso di dividerlo per 5 mediandolo appunto con il numero di nodi presenti in un insieme del grafo bipartito.

A parte la scelta di alcune costanti è stato esattamente implementato in [30], anche se è stato introdotto un parametro per tollerare l'errore della macchina nel decidere in *Shortest Path Tree* se il costo di un nuovo cammino è inferiore a quello precedentemente individuato: tale errore impediva a SPT di trovare addirittura l'albero, entrando in un falso ciclo negativo [30].

Infine, per quanto riguarda il risultato finale dell'algoritmo, ovvero il valore che rappresenta la somiglianza tra i due template comparati, abbiamo deciso di dividerlo per il numero di minuzie realmente accoppiate che, ricordiamo, può essere anche in numero minore di 25, il massimo possibile, visto che ci possono essere nodi nel grafo ausiliario oggetto dell'assegnamento non connessi con altri data la loro bassa affidabilità. Risulta chiaro quindi che il valore di match deve essere mediato per il numero di minuzie accoppiate, visto che un basso valore finale può non indicare una probabile somiglianza tra le due impronte, se ottenuto in base ad un basso numero di minuzie.

Molti altri dettagli implementativi dell'algoritmo sono descritti direttamente nei commenti del codice da cui è possibile generare la documentazione Javadoc [137] (vedi Appendice C).

6.3 Caratteristiche di alcuni database di impronte.

La scelta dei database da utilizzare per controllare le prestazioni del proprio algoritmo di matching di impronte digitali, riveste un ruolo molto importante, visto che i risultati possono essere pesantemente influiti dalla qualità delle immagini collezionate. In questo senso, i database realizzati dal NIST possono rappresentare un valido "benchmark" sul quale testare il programma [131]: uno di questi, il *NIST Special Database 9* [132], è stato

utilizzato anche nel test dell'algoritmo del gruppo di A. K. Jain [35]. Attualmente esso è suddiviso in 5 diversi volumi, ciascuno contenente 3 CD-ROM (in pratica 900 coppie di immagini di impronte per CD-ROM); ciascuna di queste immagini ha una risoluzione di 832X768; un campione di immagini di questo database può essere ottenuto all'indirizzo [133], mentre, sempre nelle stesse pagine web [131], sono segnalati altri database del NIST acquisiti in maniera differente.

La collezione di impronte digitali che abbiamo però scelto per effettuare i test sul nostro algoritmo, proviene dalla competizione per algoritmi di verifica di impronte digitali *Fingerprint Verification Competition*, che si presenta come la prima competizione a livello nazionale per questo tipo di algoritmi e di cui si sono già svolte con crescente successo le prime due edizioni (2000 [66] e 2002 [67]): per questo motivo verrà organizzata anche una prossima edizione nel 2004 [68] per la quale gli eseguibili degli algoritmi devono essere presentati entro il 31 Ottobre 2003 nella forma di Win32 Console Application.

Lo scopo principale di questa manifestazione, a detta anche degli organizzatori, è quello di misurare con la frequenza di due anni i progressi effettuati nel campo dei sistemi biometrici rivolti alle impronte digitali ed in particolar modo alle prestazioni degli algoritmi di matching.

Tuttavia il risultato della competizione non deve essere vista come certificazione ufficiale di tali prestazioni, dato che

- ◊ I database utilizzati in questo contesto non sono stati necessariamente acquisiti in un ambiente applicativo reale e le loro immagini non sono state collezionate secondo un protocollo formale.

- ◇ Sono testate solo alcune parti software del sistema, senza prendere in considerazione i sensori nativi dei singoli sistemi.

Nonostante questo, i risultati della competizione servono come utile panoramica dello stato dell'arte in questo campo e possono servire anche come punto di confronto alle aziende per valutare e migliorare i propri algoritmi.

Gli organizzatori fanno parte del Biometric System Lab dell'università di Bologna, del Pattern Recognition and Image Processing Lab dell'università dello stato del Michigan e dell'università di San Jose (due università degli Stati Uniti); i partecipanti, provenienti da tutto il mondo, possono appartenere sia alla realtà accademica sia a quella industriale, e quest'ultima presenza è notevolmente aumentata nella seconda edizione: nel 2000 si erano presentati 4 partecipanti del mondo industriale e 7 di quello accademico, mentre nel 2002 i due gruppi erano saliti rispettivamente a 21 e 6, con 4 partecipanti di natura differente dalle prime due (per esempio sviluppatori indipendenti).

Per concludere con FVC, l'edizione 2004 porta con sé un'interessante novità: questa volta le categorie nelle quali poter competere sono due e non più una sola. Infatti esiste una sezione *open* adatta ad algoritmi senza particolari obblighi riguardanti le dimensioni dei template e con dei limiti riguardanti rispettivamente il tempo di enrollment e di matching, di 10 e 5 secondi. La categoria *light* è stata creata invece per algoritmi realizzati per architetture leggere e quindi obbligati ad essere eseguiti in ambienti dalle capacità di calcolo limitate; per esempio i template delle impronte non possono superare i 2Kb ed il tempo necessario al matching non deve superare 0,3

secondi (quello per l'enrollment 0,5 secondi), misurati su un PC AMD ATHLON 1600+ (1.41 GHz) con Windows XP come sistema operativo.

Le impronte digitali utilizzate come benchmark in FVC, sono sempre state suddivise in quattro sotto-collezioni, ciascuna ottenuta a partire da una differente fonte o attraverso diversi metodi di acquisizione; di seguito riportiamo le caratteristiche dei quattro insiemi di impronte adottate nella futura edizione 2004:

- ◇ CrossMatch V300 (sensore ottico) con risoluzione dell'immagine 640x480.
- ◇ Digital Persona U.are.U 4000 (sensore ottico) con risoluzione dell'immagine 328x364.
- ◇ Atmel FingerChip FCD4B14CB (sensore termico) con risoluzione dell'immagine 300x480.
- ◇ Il quarto database è generato con un programma generatore di impronte, chiamato SFinGe [134] e approfondito nel seguito di questo paragrafo, con risoluzione dell'immagine 288x384.

I risultati degli algoritmi vengono in seguito pubblicati per ognuno dei differenti db, ma viene anche fornito anche un riassunto complessivo delle prestazioni come è mostrato, limitatamente ai primi 10 partecipanti indicati con una sigla (viene consentito anche l'anonimato), nella tabella 6.1 tratta dal sito dell'edizione 2002 e da noi modificata: tali prestazioni sono quelle ottenute mediamente su l'insieme di tutti e quattro i database.

Algorithm	Avg EER	Avg FAR100	Avg FAR1000	Avg Zero FAR	Avg REJ _{ENROLL}	Avg REJ _{MATCH}	Avg Enroll Time	Avg Match Time
<u>PA15</u>	0.19%	0.15%	0.28%	0.38%	0.00%	0.00%	0.11 sec	1.97 sec
<u>PA27</u>	0.33%	0.28%	0.56%	1.44%	0.00%	0.00%	2.12 sec	1.98 sec
<u>PB27</u>	0.41%	0.34%	0.59%	1.29%	0.00%	0.00%	1.23 sec	1.13 sec
<u>PB15</u>	0.77%	0.77%	1.04%	1.29%	0.00%	0.00%	0.07 sec	0.22 sec
<u>PB05</u>	0.92%	1.46%	1.87%	2.29%	0.00%	0.00%	0.48 sec	0.52 sec
<u>PA08</u>	0.99%	1.12%	2.07%	3.11%	0.00%	0.00%	0.56 sec	0.56 sec
<u>PA35</u>	1.18%	1.32%	2.71%	4.21%	0.00%	0.00%	4.05 sec	1.65 sec
<u>PA02</u>	1.31%	1.34%	1.80%	2.22%	0.32%	0.20%	0.81 sec	1.23 sec
<u>PB35</u>	1.42%	1.51%	2.71%	4.60%	0.00%	0.00%	0.77 sec	0.66 sec
<u>PA13</u>	2.18%	2.49%	3.48%	10.29%	0.00%	0.14%	0.17 sec	0.48 sec

Tabella 6.1
Prestazioni dei primi 10 algoritmi rilevate durante la competizione FVC2002:
FAR100 : la più bassa FRR per FAR≤1%
FAR1000: la più bassa FRR per FAR≤0.1%
ZeroFAR: la più bassa FRR per FAR=0% [67].

I parametri per la misurazione delle performance adottati per valutare gli algoritmi sottoposti alla gara sono esposti nel seguito. I primi due seguenti punti, rappresentano i due più famosi indicatori di prestazioni e sono stati definiti anche nel paragrafo 3.3:

Nella descrizione seguente verrà fatto riferimento ai match “genuini”, come quelli alla fine dei quali viene ottenuto un risultato positivo e quindi le due impronte sono ritenute appartenenti allo stesso dito, mentre in quelli “impostori” il risultato è negativo.

Per ciascun algoritmo e database vengono riportati i seguenti indicatori delle performance:

- ◇ REJENROLL (numero di impronte respinte durante l’enrollment).
- ◇ REJNGRA (numero di impronte respinte durante i match genuini).
- ◇ REJNIRA (numero di impronte respinte durante i match impostori).

- ◇ Distribuzioni dei match “genuini” e “impostori”.
- ◇ Le curve di $FAR(t)/FRR(t)$, dove t è la soglia di accettazione.
- ◇ Curve di $ROC(t)$.
- ◇ EER o equal-error-rate.
- ◇ EER^* (il valore che EER assumerebbe se i fallimenti nel matching fossero esclusi dal calcolo del FAR e FRR).
- ◇ FAR100 (il più basso FRR per $FAR \leq 1\%$).
- ◇ FAR1000 (il più basso FRR per $FAR \leq 0.1\%$).
- ◇ ZeroFAR (il più basso FRR per $FAR = 0\%$).
- ◇ ZeroFRR (il più basso FAR per $FRR = 0\%$).
- ◇ Tempo medio speso per la sessione di enrollment.
- ◇ Tempo medio speso per la sessione di matching.
- ◇ Dimensione massima e media del template.
- ◇ Massima quantità di memoria allocata.

Questi altri indicatori di performance vengono indicati invece per tutti i database e riguardano quindi le prestazioni medie acquisite:

- ◇ EER medio.
- ◇ FAR100 medio.
- ◇ FAR1000 medio.
- ◇ ZeroFAR medio.

- ◇ REJENROLL medio (numero medio di impronte respinte durante la fase di enrollment).
- ◇ REJMATCH medio (numero medio di impronte respinte durante match genuini e impostori) .
- ◇ Tempo medio della fase di enrollment.
- ◇ Tempo medio della fase di matching.
- ◇ Dimensione media del template (calcolata sulla dimensione media del template per ogni database) .
- ◇ Memoria media allocata (calcolata sulla base della massima quantità di memoria allocate per ciascun database).

Può essere infine utile spendere due parole a riguardo del metodo per la generazione del quarto database: in questo caso le immagini delle impronte vengono generate grazie ad un programma chiamato SFINGE [134] (Synthetic *FIN*gerprint *GE*nerator), ideato principalmente dagli stessi organizzatori del FVC e facenti parte del *Biometric Systems team* dell'Università di Bologna, attivo fin dal 1993. Per quanto riguarda la generazione, vengono utilizzati i filtri di Gabor per espandere iterativamente l'immagine inizialmente vuota e contenete uno o più semi; i passi seguiti durante la creazione sono i seguenti:

- a Generazione della mappa direzionale.
- b Generazione della mappa di densità.
- c Generazione dello schema delle creste.

d Fase di disturbo e renderizzazione.

Nel passo *a* si parte da dalla posizione dei punti di core e di delta e tramite un modello di flusso matematico viene generata una mappa direzionale delle creste basata sui suddetti punti caratteristici. Nel passo *b* viene generata una mappa di densità in base a criteri euristici. Al passo seguente (il *c*), vengono generate le creste e le minuzie tramite il filtraggio di Gabor, mentre nel passo *d* viene disturbata e renderizzata in scala di grigio per renderla più realistica.

SFINGE può essere quindi usato per creare grandi database di impronte per testare le performance dei propri algoritmi di matching, dato che, per esempio, una collezione di 10000 impronte può essere “costruita” in circa 10 ore di computazione di un semplice PentiumIV.

La figura 6.5 mostra un esempio di due impronte dello stesso dito create attraverso il programma.



Figura 6.5
Due immagini dello stesso dito create con SFINGE [134].

6.4 Test e prestazioni rilevate sul database utilizzato.

Per testare le prestazioni del nostro algoritmo abbiamo utilizzato i quattro database della competizione FVC2002 gentilmente messo a disposizione dal dipartimento di scienze dell'informazione di Cesena appartenente all'università di Bologna; vogliamo ringraziare quindi tutti gli appartenenti al *Biometric Systems Lab* [136].

Le caratteristiche di acquisizione di questi quattro database sono riassunte di seguito:

- ◇ DB1: sensore ottico "TouchView II" di Identix.
- ◇ DB2: sensore ottico "FX2000" di Biometrika.
- ◇ DB3: sensore capacitativo "100 SC" di Precise Biometrics.
- ◇ DB4: un database ottenuto attraverso *Synthetic FINGERprint Generation* (SFINGE), con il quale termine si indica la costruzione di impronte digitali artificiali [134].

Ciascuno dei quattro db è formato da 110 diverse impronte digitali, ciascuna di esse ripetuta in 8 differenti immagini, per un totale quindi di 880 immagini; una parte di questi db e cioè l'insieme *B* di figura 6.5 consistente in 10 impronte ciascuna ripetuta 8 volte, è stata resa pubblica anche prima della competizione per permettere ai partecipanti di tarare i loro algoritmi in base alle differenti caratteristiche delle immagini sulle quali devono essere eseguiti, lasciando quindi anche un minimo di adattabilità nel funzionamento del programma. Nei nostri test abbiamo utilizzato l'insieme *A* e l'insieme

B di figura 6.5 come un unico grande insieme, ed è per questo che abbiamo effettuato un numero maggiore di test rispetto a quello sfruttato in FVC2002 che utilizzava chiaramente solo la parte tenuta riservata per la gara (l'insieme A): abbiamo effettuato 9075 test contro i 7750 di FVC2002.

La tabella 6.2, tratta dal sito internet dell'edizione 2002 [67], riassume tutte le caratteristiche dei quattro database, mentre nella figura 6.6, tratta dalla stessa fonte, sono mostrate esempi di immagini per ciascuno dei quattro db.

	Tipo sensore	Dim. immagine	Insieme A	Insieme B	Risoluzione
DB1	Sensore ottico	388x374 (142 Kpixels)	100x8	10x8	500 dpi
DB2	Sensore ottico	296x560 (162 Kpixels)	100x8	10x8	569 dpi
DB3	S. capacitativo	300x300 (88 Kpixels)	100x8	10x8	500 dpi
DB4	SFinGe v2.51	288x384 (108 Kpixels)	100x8	10x8	circa 500 dpi

Tabella 6.2
Tabella riassuntiva delle caratteristiche dei quattro database di test [67].

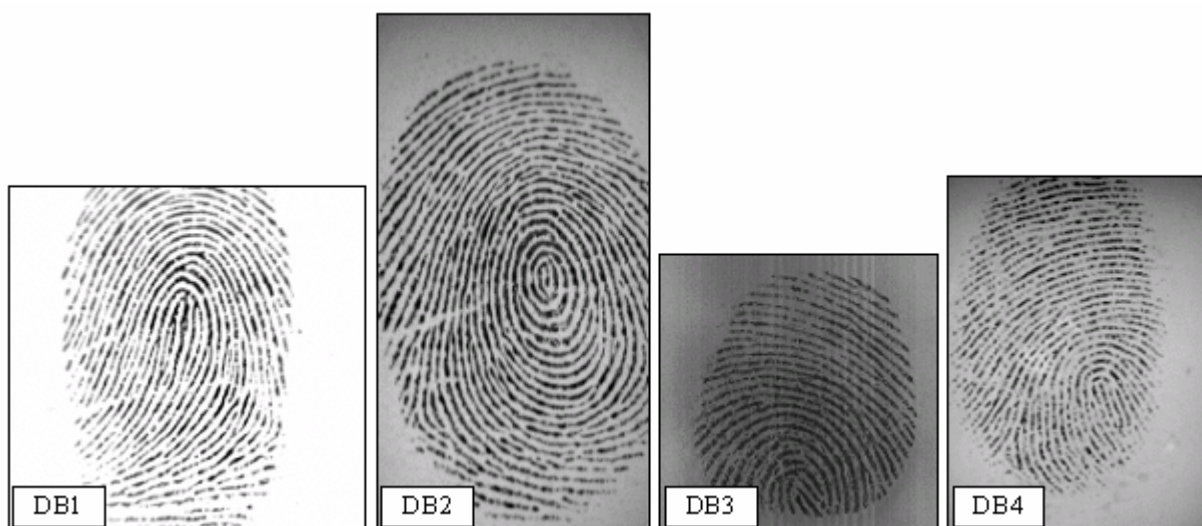


Figura 6.6
Esempi di immagini di impronte digitali per ciascuno dei quattro database FVC2002 [67].

Il nostro test sulle performance consiste nell'unione delle due seguenti prove, per un totale di 9,075 comparazioni tra impronte:

- ◇ Ciascun campione del database viene comparato (match) con i rimanenti campioni dello stesso dito per calcolare il *False Rejection Rate* (FRR). Se viene eseguito il match di g nei confronti di h , il simmetrico (h con g) non viene effettuato. Quindi il numero totale di test (nel caso in cui nessuna impronta venga respinta durante l'enrollment) è di $((8*7) / 2) * 110 = 3,080$
- ◇ Il primo campione di ciascun dito nel database è comparato (match) con le rimanenti dita in A per calcolare il *False Acceptance Rate* (FAR). Allo stesso modo del punto precedente, vengono evitati i match simmetrici delle impronte. Il numero totale di test (nel caso in cui nessuna impronta venga respinta durante l'enrollment) è quindi $((110*109) / 2) = 5,995$.

In un primo tempo abbiamo scelto di esaminare le prestazioni sulle singole feature, che ricordiamo di aver normalizzato in un valore tra 0 e 1 (paragrafo 6.2); abbiamo deciso di utilizzare un peso pari a 1000 con ogni feature per eseguire il passo successivo: nei primi quattro test eseguiti abbiamo impostato a zero tutti i pesi delle feature eccetto rispettivamente quello della distanza euclidea, quello dell'angolo relativo alla distanza, quello della differenza tra gli angoli delle creste ed infine quello relativo alla distanza in creste tra le minuzia centrale e vicina (vedi le quattro feature descritte in 5.2.1). Questi test sono serviti per capire quali prestazioni offriva l'algoritmo sulle singole feature e quindi utilizzare i risultati per poter combinare insieme i loro pesi in maniera

opportuna al momento di utilizzarle tutte. I quattro punti di Equal Error Rate dove le due curve di FAR e FRR si incontrano, sono calcolati per i test descritti precedentemente in:

- ◇ 34,5% per la distanza euclidea tra minuzia centrale e vicina (nel seguito *Ed*), con soglia a 50.5.
- ◇ 27,8% per l'angolo relativo alla distanza tra minuzia centrale e vicina (nel seguito *Dra*), con soglia a 136.5.
- ◇ 33,7% per l'angolo-differenza tra le direzioni della minuzia centrale e vicina (nel seguito *Ra*), con soglia a 121.5.
- ◇ 39%% per la distanza in creste tra minuzia centrale e vicina (nel seguito *Rc*) con soglia a 55.5.

Nelle figure 6.7, 6.8, 6.9 e 6.10 sono riportati i grafici in cui sono mostrate le curve di FAR e FRR (ed il loro punto di intersezione in EER) per i test effettuati pesando a 1000 rispettivamente la feature *ed*, *dra*, *ra* e *rc* e a 0 le altre 3. Le curve sono riportate per un valore massimo di soglia pari a 400, mostrandone la parte più interessante

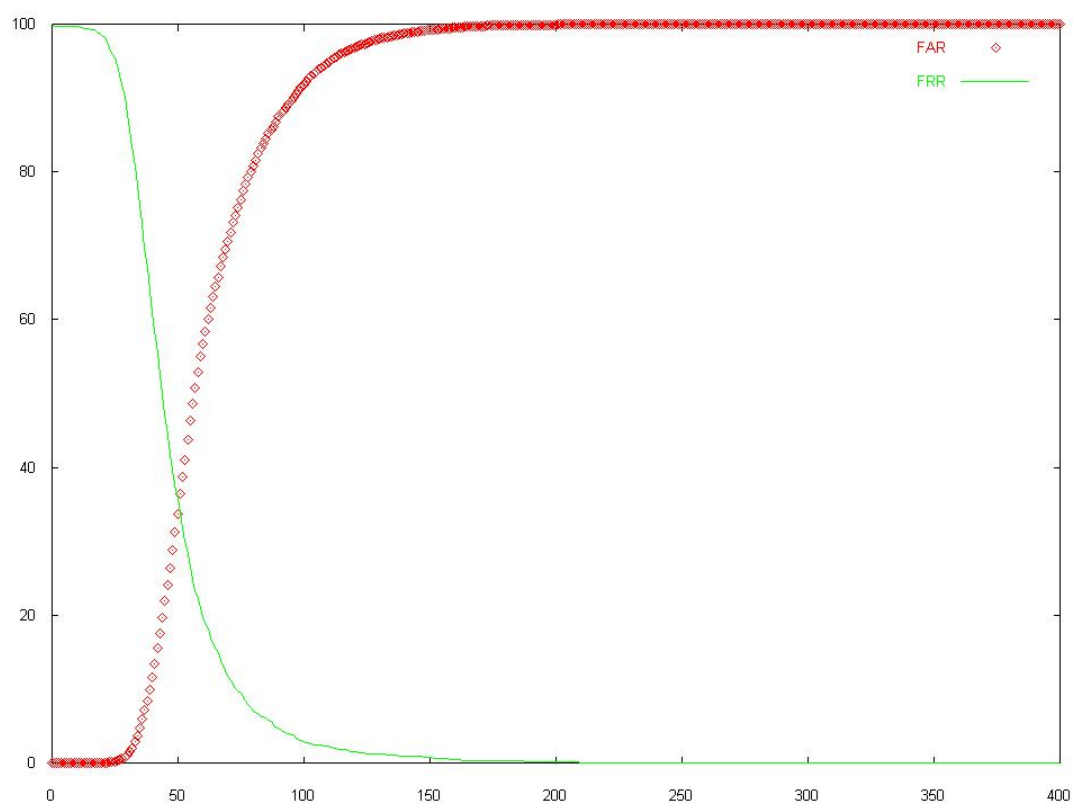


Figura 6.7
 Grafo FAR-FRR ottenuto settando a 1000 il peso di ed e a 0 tutti gli altri.
 EER a 34,5% al valore soglia di 50.5.

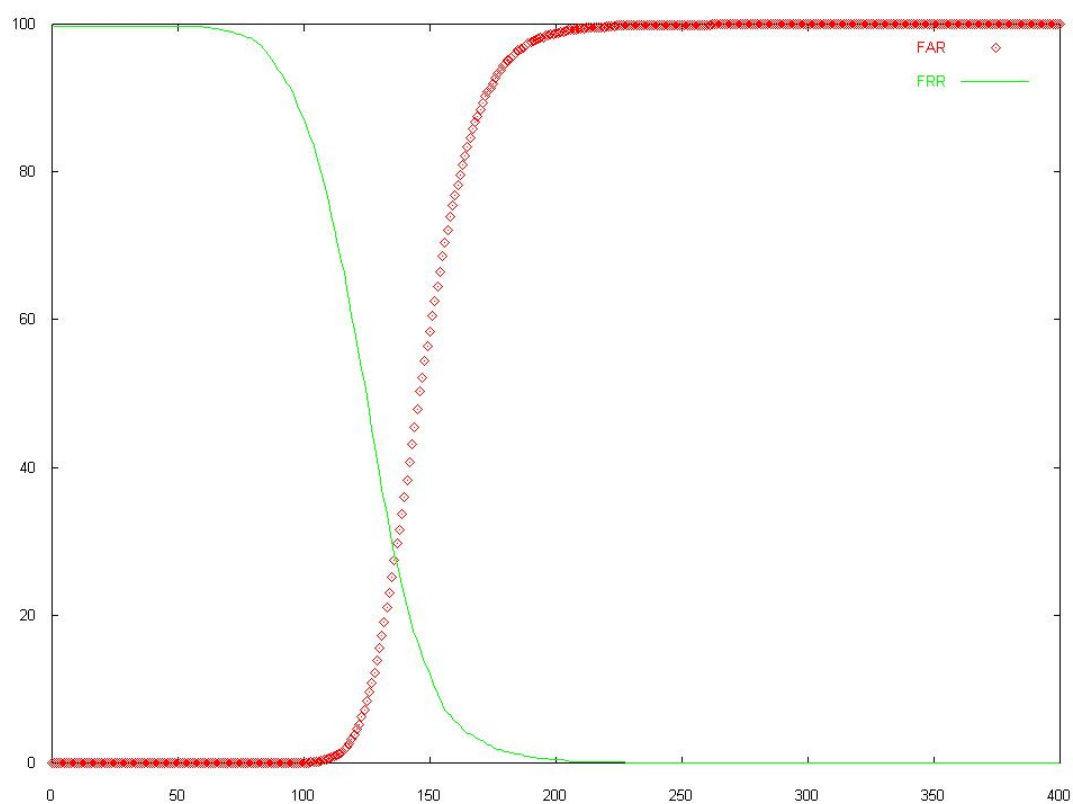


Figura 6.8
 Grafo FAR-FRR ottenuto settando a 1000 il peso di *dra* e a 0 tutti gli altri.
 EER a 27,8 al valore soglia di 136.5.

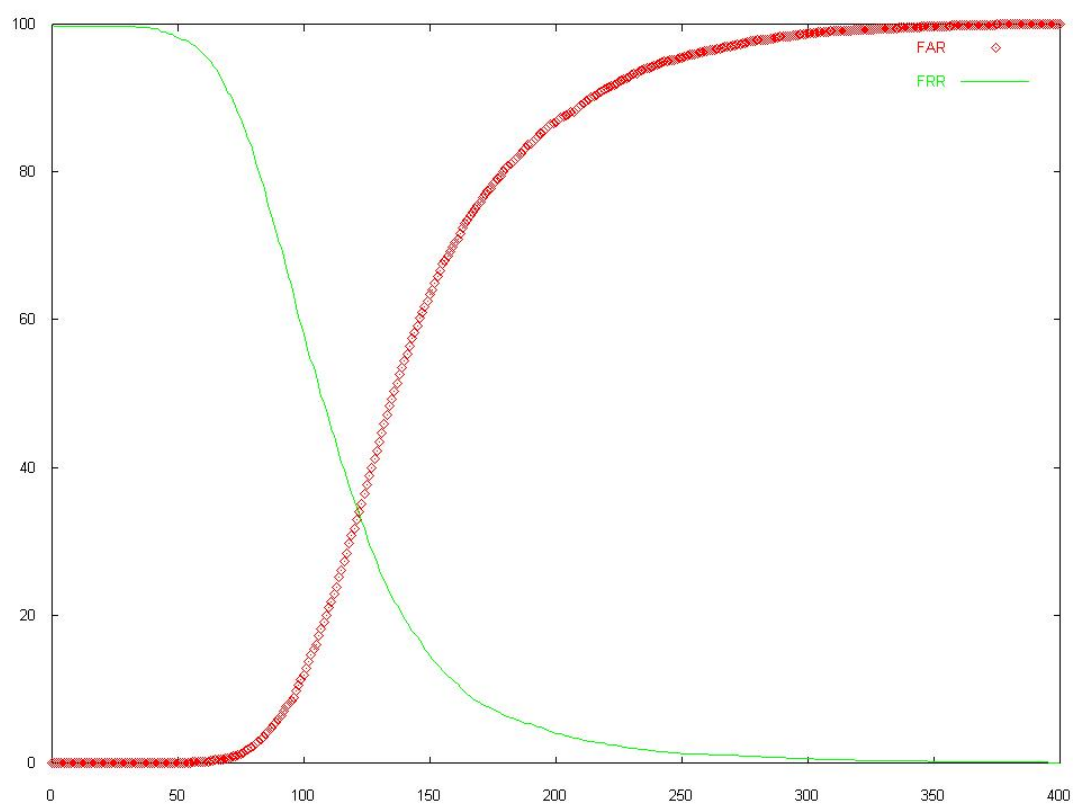


Figura 6.9
 Grafo FAR-FRR ottenuto settando a 1000 il peso di ra e a 0 tutti gli altri.
 EER a 33,7% al valore soglia di 121.5.

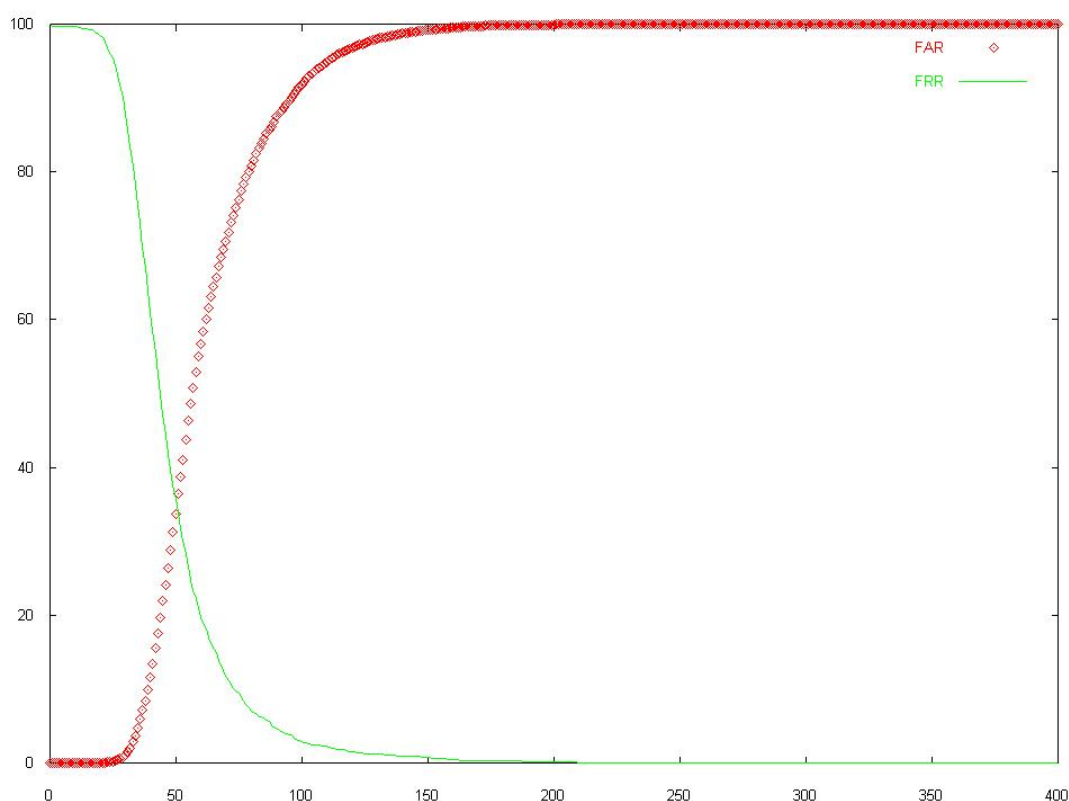


Figura 6.10
Grafo FAR-FRR ottenuto settando a 1000 il peso di rc e a 0 tutti gli altri.
EER a 39% al valore soglia di 55.5.

Per trovare una buona combinazione dei pesi delle differenti feature per poterle sommare insieme e migliorare così l'affidabilità del match, abbiamo in un primo momento osservato che i valori soglia per cui le prestazioni sulle singole feature erano migliori (quelli con i quali si otteneva il punto di EER), erano abbastanza distanti tra di loro. Abbiamo quindi cercato di riportare tutti i punti di EER calcolati per le varie feature attorno al valore soglia di 500: per ciascuna feature questo è stato ottenuto dividendo il valore 1000 iniziale del peso per il valore soglia ottenuto per il punto di EER ed infine è stato moltiplicato per 500. Per esempio per la feature ra il nuovo peso è :

$$1000 / 50.5 \times 500 = \text{nuovo peso } ra$$

L'ultimo dei passi che abbiamo eseguito per trovare dei pesi per ottimizzare le prestazioni è stato affrontato sperimentalmente: grazie ai test eseguiti variando il valore dei pesi siamo riusciti a stabilire dei buoni miglioramenti dando più importanza alle feature che garantiscono maggiore affidabilità nel riconoscimento, in primis *dra* e poi, in ordine decrescente, *ra*, *ed* e *rc*.

Nella figura 6.14 viene mostrato anche il grafo Receiving Operating Curve (ROC) del DB1; questo grafo si calcola con i risultati di $FAR(t)$ e $FRR(t)$ per lo stesso t : in pratica le coordinate sul piano sono date dal valore di FAR e FRR calcolati alla stessa soglia di accettazione t (vedi paragrafo 3.3).

Il tempo necessario per verificare due impronte partendo dai due template già costruiti è risultato essere mediamente di poco superiore a 0,12 secondi e quindi l'intero test di 9,075 match su un database ha impiegato 18 minuti e 35 secondi su di un processore Athlon a 1.4Ghz e 512Mbyte di RAM sotto Windows[®]XP.

Il tempo necessario invece per l'enrollment è pari mediamente a 0,44 secondi per ottenere il file con la lista delle minuzie tramite MINDTCT, per un totale di 6 minuti e 30 secondi su tutto uno dei database utilizzati da 880 impronte. A questo tempo va aggiunta la conversione in un nuovo file di testo contenente le informazioni in un formato compatibile con l'algoritmo: questa fase impiega mediamente 0,057 secondi (50 secondi circa per tutto un database da 880 impronte).

Riassumendo quindi le prestazioni fondamentali del nostro algoritmo trovate sul DB1:

◇ *Tempo medio di verifica*: 0,12 secondi circa.

- ◇ *Tempo medio di registrazione*: 0,45 secondi circa.
- ◇ *Equal Error Rate*: 18,07% al valore soglia di 899.
- ◇ *FAR₁₀₀*: 50,97% al valore soglia di 765.
- ◇ *FAR₁₀₀₀*: 74,93% al valore soglia di 663.
- ◇ *ZeroFMR*: 83,83% al valore soglia di 605.
- ◇ *ZeroFNMR*: 98,68% al valore soglia di 1157.

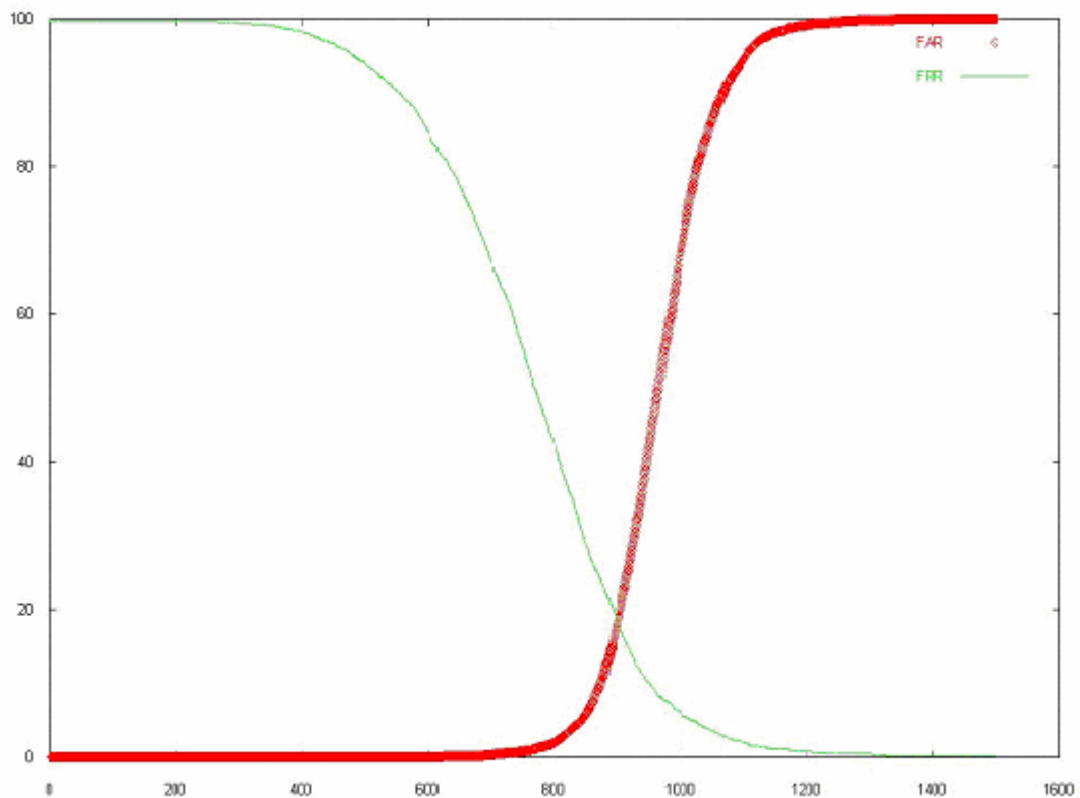


Figura 6.11
Grafo FAR-FRR ottenuto sul DB1 con l'aiuto di tutte e quattro le feature.

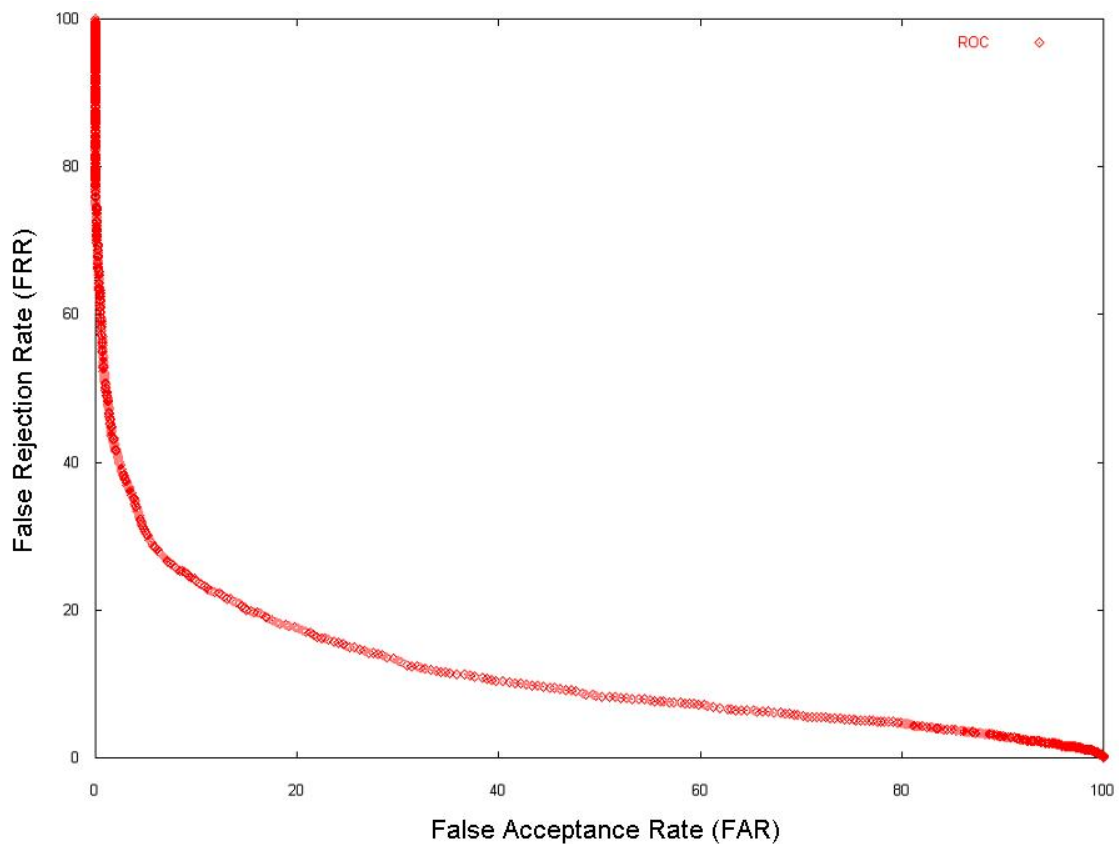


Figura 6.12
Receiving Operating Curve delle prestazioni dell'algoritmo.

Analizziamo ora i risultati delle prestazioni partendo dall'aspetto non certo secondario dei tempi: per quanto riguarda la velocità di poco superiore agli 0,12 secondi attualmente necessari per effettuare un match, possiamo dire che in realtà questa può migliorare notevolmente passando da una realizzazione JAVA ad una in linguaggio C, non interpretato e quindi sicuramente più veloce. Il porting verso il secondo linguaggio è già stato effettuato per poter partecipare alla gara di algoritmi FVC2004 [68] ed è risultato di facile realizzazione.

Per quanto riguarda il tempo medio di registrazione (enrollment), anche questo può essere abbassato eliminando il passaggio intermedio di costruzione di un nuovo file

di testo a partire da quello ottenuto dal programma MINDTCT e contenente le informazioni di input che l'algoritmo di matching può utilizzare direttamente: creando queste informazioni all'interno dell'algoritmo di matching stesso, questo potrebbe usare direttamente come file di template quello ottenuto da MINDTCT, aumentando di poco il tempo di verifica, ma evitando così la creazione di un file aggiuntivo.

Capitolo 7

Implementazione e prestazioni in ambiente Java CardTM.

7.1 Programmazione in Java CardTM.

Per la programmazione in ambiente Java CardTM è stato necessario approfondire alcuni concetti come la struttura dei messaggi scambiati tra il lettore e la carta, l'identificatore unico dell'applicazione da scaricare sulla carta, il toolkit utilizzato per la creazione e i test del nostro algoritmo di matching ed infine alcune informazioni riguardanti strettamente la programmazione sotto questo ambiente. In questo paragrafo verranno approfondite tutte queste tematiche in questo ordine.

7.1.1 Struttura della Application Protocol Data Unit (APDU).

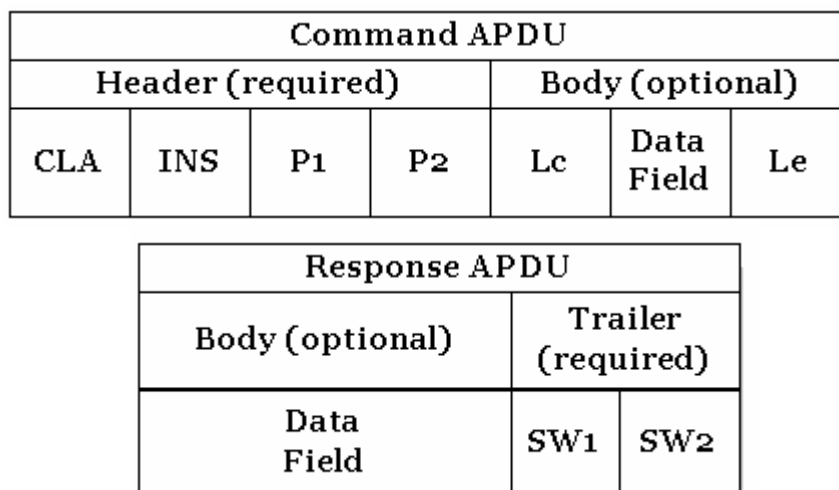


Figura 7.1
Formato delle Application Protocol Data Unit di comando e risposta [124].

Con il termine Application Protocol Data Unit (APDU, descritto negli standard ISO/IEC 7816-3 e 7816-4) viene chiamato il pacchetto di dati che viene scambiato tra il lettore/scrittore e il framework Java Card™ che la consegna poi all'applet destinatario. La APDU di comando viene spedita dal lettore alla carta, mentre quella di risposta ha un formato diverso e viene inviata dalla carta al lettore.

Prendendo in esame prima la APDU di comando (figura 7.1 [124]), questa si suddivide in un header obbligatorio che comprende i campi CLA, INS, P1 e P2 e un corpo del messaggio opzionale formato da Lc, i dati inviati e Le; il *conditional body* è presente solo nel caso nel quale, oltre alla necessità di inviare un comando alla carta, si debba fornire anche dei dati per la sua elaborazione. Il CLA (*Class of the Command*) specifica la classe del comando spedito nella APDU ed è contenuto in un byte; un valore di ooh (in esadecimale) segnala che il comando è conforme agli standard ISO. INS (*Instruction Identifier of the Command*) identifica insieme al CLA il comando ed occupa anch'esso un

byte; il comando *selectApplication* viene passato al JCRE (*Java Card™ Runtime Enviroment*), mentre tutti gli altri vengono rivolti all'applicazione al momento selezionata. P1 e P2 rappresentano invece dei parametri di input del comando ed il loro contenuto (1 byte), dipende strettamente da esso. Passando alla parte opzionale, il campo Lc indica la lunghezza in byte dei dati spediti dal lettore alla carta: questo valore è compreso tra 01h e FFh e quindi se invio 15 byte di informazioni conterrà 0Fh. Il campo Le (sempre lungo 1 byte) rappresenta invece il numero di byte della lunghezza della risposta che il lettore si aspetta di ricevere dalla carta: i byte della risposta che superano tale valore non vengono ritornati (vedi figura 7.1 [124]).

Nella APDU di ritorno invece, oltre ai dati della risposta veri e propri, vengono inclusi 2 byte SW1 e SW2, che indicano se il comando ha avuto successo oppure è fallito: esempi di ritorno per questo campo conformi agli standard ISO possono essere:

- ◇ 90 00: il comando ha avuto successo.
- ◇ 6F 00: errore della Java Virtual Machine che non ha una precisa diagnosi. Può essere ritornato anche nel caso in cui vengono superate le capacità dello stack.
- ◇ 6A 84: memoria EEPROM insufficiente per aggiungere un nuovo oggetto.
- ◇ 6E 00: valore del codice CLA non supportato.
- ◇ 6D 00: valore del codice INS non supportato.
- ◇ 65 81: fallimento sulla memoria.
- ◇ 69 99: errore nella selezione dell'applicazione.
- ◇ 6A 82: valore dell'AID non presente nel registro per la selezione dell'applicazione (vedi paragrafo 4.3 per quanto riguarda la definizione di AID).
- ◇ 6A 86: valori specificati nei campi P1 o P1 o entrambi incorretti o non supportati.

Le modalità di interazione tra carta e lettore sono 4 e vengono classificate in base alla presenza del campo dei dati, come specificato in ISO 7816-4: nella modalità S (send) nella l'APDU di comando sono presenti dati, ma non lo sono in quella di risposta, in R (receive) accade esattamente il contrario, mentre nella S/R (send/receive) i dati sono presenti in tutte e due le APDU scambiate. Nell'ultima configurazione sono presenti solo i campi obbligatori in tutte e due: CLA, INS, P1 e P2 per la prima e SW1 e SW2 per la seconda. In figura 7.2 ([124]) sono mostrate tutte e quattro le possibili configurazioni di interazione dal punto di vista della APDU di comando: la APDU di risposta invece presenterà il campo Data Field solo nel caso in cui la risposta deva contenere dei dati, mentre le due status word saranno sempre e comunque presenti.

I comandi specificati nelle APDU possono essere inviati sia al Card Manager (che agisce da sistema operativo predefinito) per stabilire per esempio una sessione di comunicazione sicura o selezionare un'altra applicazione, oppure possono essere specifici per applicazioni aggiunte sulla carta per offrire nuovi servizi. Per avere informazioni più precise riguardanti questi comandi ed il loro formato in un ambiente Java Card™ basta consultare la guida di programmazione [105].

Caso 1:
Nessun dato in comando
Nessuna risposta richiesta

CLA	INS	P1	P2

Caso 2:
Nessun dato in comando
Risposta richiesta

CLA	INS	P1	P2	Le

Caso 3:
Dato nel comando
Nessuna risposta richiesta

CLA	INS	P1	P2	Lc	Data Field

Caso 4:
Dato nel comando
Risposta richiesta

CLA	INS	P1	P2	Lc	Data Field	Le

Figura 7.2
 Varie possibilità di costruzione della APDU di comando in base
 alla comunicazione del lettore/scrittore con la carta [124].

7.1.2 Descrizione dell'Application Identifier (AID).

Ciascun applet su una carta è univocamente determinato da un *Application ID* (AID). Un AID, così come viene definito all'interno dello standard ISO 7816-5, consiste in una sequenza tra 5 e 16 byte ed è diviso in due parti dal punto di vista della gestione degli assegnamenti: la prima parte di 5 byte prende il nome di *National Registered Application Provider* (o RID) e viene assegnata da ISO direttamente alle compagnie che ne fanno richiesta (ciascuna di esse avrà il suo RID che la identifica); la seconda, chiamata *Proprietary Application Identifier Extension* (o PIX), può essere lunga da 0 a 11 byte e viene gestita dalla compagnia al fine di identificare univocamente i propri applet.

Application Identifier (AID)	
National Registered application provider (RID)	Proprietary application identifier extension (PIX)
5 bytes	0 to 11 bytes

Figura 7.3
Struttura di un Application Identifier (AID) [122].

Oltre all'applet, anche il package che lo contiene possiede un proprio AID che deve differire da quello dell'applet per lo meno di un bit nella parte del PIX, dato che una nota tecnica contenuta nel manuale [107] indica che nell'ultima versione (sia la 1.2, sia la 1.3) del *Java Converter* di Sun, i primi 5 byte dell'AID (RID, vedi figura 7.3 tratta da [122]) del package e dell'applet devono essere identici.

Quando l'applet viene istanziata all'interno della carta essa si registra all'interno del JCRE con il suo AID che la identifica univocamente ed è tramite questo stesso identificatore che il JCRE (applet in esecuzione di default) chiama in esecuzione uno specifico programma che continua ad occupare la JCVM fino al momento in cui non ne viene selezionato un altro.

7.1.3 Toolkit Cyberflex di Schlumberger.

Per la realizzazione del nostro algoritmo in linguaggio Java Card™, abbiamo utilizzato la piattaforma di sviluppo Cyberflex [89] di Schlumberger. Questo prodotto include vari programmi per controllare la coerenza del bytecode agli standard Java Card™, per costruire il file da scaricare sulla carta, per caricarlo sulla carta ed infine istanziare l'applet contenuta; inoltre, nel kit è contenuto anche una utility chiamata *APDU Manager*, che serve per assemblare e spedire messaggi (APDU) all'applet; i tool della piattaforma sono descritti di seguito:

- ◇ *Card Manager*: mostra la carta inserita nel lettore/scrittore e anche il contenuto della carta, come i file caricati, le librerie e gli applet istanziati.
- ◇ *Program file generator*: questo tool serve per generare il file .ijc da scaricare poi sulla carta: necessita di parametri quali il nome e l'AID del package, l'applet (file .class) da scaricare e la directory contenente i file .exp che contengono le informazioni sugli altri package riferiti.
- ◇ *APDU Manager*: offre un aiuto nella costruzione dei campi che formano il messaggio da spedire al lettore (nel seguito del capitolo vengono trattati più esaurientemente); dopo l'invio della APDU è possibile osservare l'esito del comando tramite i dati di ritorno e la status word. Con l'*APDU scripter* facente parte dell'*APDU Manager*, consente inoltre di creare alcuni script di comandi (in pratica una sequenza) da inviare alla carta, al fine di testare un'intera sessione di enrollment e di matching dell'algoritmo, velocizzando così il controllo della correttezza.
- ◇ *Key Manager*: consente di eseguire alcune operazioni basate sull'uso delle chiavi come aggiungere, cancellare o modificare un insieme di chiavi della carta. Per l'accesso a questa infatti è necessario affrontare una sessione iniziale che permette poi di lavorare in sicurezza su di essa. Sbagliare questa fase iniziale per tre volte consecutive significa bloccare permanentemente la carta e perdere così l'accesso permanentemente.

Oltre ai più importanti strumenti citati precedentemente, nel programma di Cyberflex sono presenti altre finestre di log che registrano l'intera comunicazione tra il

lettore/scrittore e la carta mostrando i dati in esadecimale e altre di supporto all'istanziamento dell'applet che prendono come parametro anche l'AID.

Come altre realizzazioni proprietarie, anche la piattaforma Cyberflex da noi usata include dei tool di sviluppo e installazione molto più “amichevoli” rispetto a quelli di Sun, che d'altronde sono gratuitamente scaricabili e usabili. Ad esempio, invece che da riga di comando è possibile editare le opzioni nei campi delle finestre ed il processo di scelta risulta notevolmente semplificato nei suoi parametri.

Differentemente dal Java Card™ realizzato da Sun però, il file CAP è stato sostituito dal file .ijc (*Interoperable Javacard CAP*), una versione più semplice della prima e adottata da molti costruttori: risultano esserci infatti alcuni problemi di compatibilità rispetto ai file CAP accentuati forse dal fatto che Sun non rese immediatamente una versione di *converter* di file .class.

I passi per sviluppare applet Java Card™ tramite piattaforma Cyberflex sono descritti in figura 7.4.

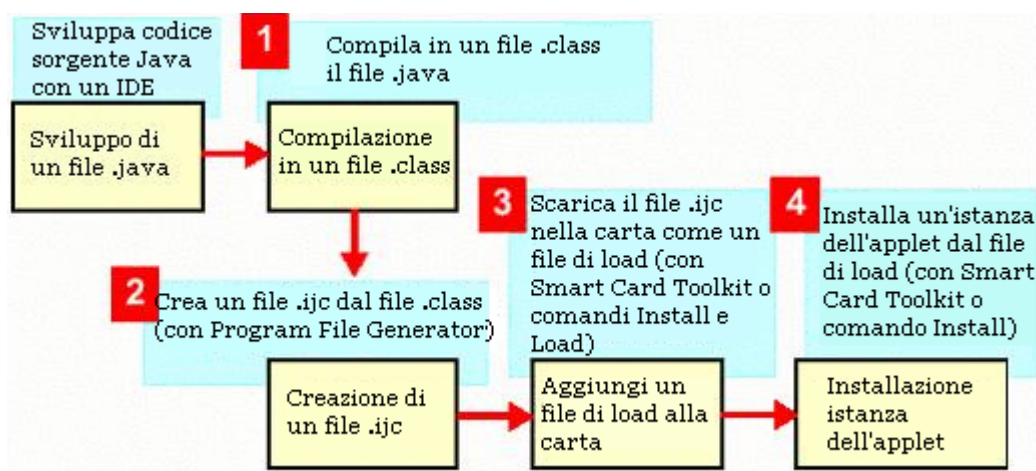


Figura 7.4
Passi necessari all'installazione di un programma su smart card con la Piattaforma Cyberflex [89].

Per quanto riguarda il processo di creazione di un applet e la sua successiva installazione sulla carta, facendo sempre riferimento alla figura 7.4, come primo passo basta usare un normale compilatore Java importando le API Java Card™ dove necessario, per esempio per estendere la classe *Applet*; sulla piattaforma Cyberflex [89], si sconsiglia di compilare con l'opzione di ottimizzazione del codice, dato che aumenta l'occupazione di memoria del risultato finale e di usare invece l'opzione che aggiunge informazioni di debugging di cui la VM ha bisogno. Per questa prima fase di lavoro è stato utilizzato *Netbeans IDE 3.4* [138], un ambiente integrato per lo sviluppo di applicazioni Java, in cui è presente anche un valido debugger.

Dopo aver ottenuto il bytecode nel file .class, a questo punto è necessario convertire grazie al tool *Program File Generator* i file .class ottenuti in un unico file .ijc, che corrisponde logicamente all'intero package. Successivamente il file potrà essere caricato sulla carta o su un simulatore per testarne l'effettivo comportamento. Brevemente, i principali motivi della conversione in un file CAP sono la necessità di un pre-load delle classi dato che la tecnologia non supporta il caricamento dinamico, quella di effettuare operazioni costose come il linking sul lato host, quella di assegnare in questa fase un AID all'applicazione ed in generale il bisogno di controllare che l'applet sia stata scritta seguendo i costrutti e i tipi dato a disposizione del linguaggio Java Card™ (per esempio, che non ci siano stringhe). Per quanto riguarda la verifica del bytecode effettuata dal tool invece, essa serve per controllare sia la correttezza della sua struttura, sia per esaminare la corretta interazione dei propri metodi con altri package (per esempio nel numero dei parametri e nel loro tipo). Se la conversione ha successo, *Program File Generator* salva il file .ijc in una directory di output, altrimenti segnala in una finestra di debug l'errore rilevato.

Nel terzo passo non resta che scaricare il file .ijc nella carta grazie al *Card Manager*, che non fa altro che ricevere in input in una finestra il percorso del file .ijc e l'AID del package; questo tool non fa altro che nascondere la sequenza di comandi *Load* e *Install* per facilitare lo sviluppatore nello sviluppo: questi comandi potrebbero essere spediti direttamente alla carta tramite APDU.

Una volta scaricato tutto il materiale sulla carta, rimane solo da creare un'istanza dell'applet tramite il comando *install*, selezionarla tramite il comando *select* per farla entrare in esecuzione sulla carta ed infine spedire comandi specifici per l'applicazione per testarne il funzionamento; entrambi i sopracitati comandi sono inviati al Card Manager (l'applicazione di default) e sono descritti nel loro formato insieme agli altri nel manuale di programmazione [105]. Sempre tramite il *Card Manager* è possibile effettuare istanziazione e selezione dell'applet in modo più amichevole (tramite finestra grafica).

Anche la carta *Cyberflex Access Developer 32K card*, da noi utilizzata per realizzare l'algoritmo di verifica di impronte digitali, è distribuita da Schlumberger [130] ed alcune delle sue caratteristiche più interessanti possono essere riassunte nei seguenti punti:

- ◇ 32Kbyte di memoria EEPROM.
- ◇ CPU a 8 bit.
- ◇ Supporto per un ambiente multi applicazione.
- ◇ Frequenza del clock esterna da 1 a 7.5 Mhz.
- ◇ Supporto per entrambi i protocolli di comunicazione T=0 e T=1 (vedi 4.2.4)

- ◇ Conformità verso gli standard ISO 7816, Java Card™ 2.1.1, Open Platform 2.0.1, PKCS#11 e Crypto API.

Durante la realizzazione del codice ed i test indirizzati a controllare il funzionamento del programma, sono stati risolti con successo tutti i problemi di configurazione e funzionamento nell'uso, del sopra citato toolkit. Ci siamo infatti accorti che anche se la card ritornava un errore alla fine del download dell'applet, oppure il programma si bloccava al termine di quella fase, l'applet era stata effettivamente in molti casi scaricata comunque nella memoria EEPROM.

7.1.4 Note di programmazione in ambiente Java Card™.

Nella realizzazione del programma su smart card, è stato seguito in primis l'esempio fornito nel manuale di programmazione fornito con i tool di sviluppo *Cyberflex Access Programmer's Guide* in dotazione alla versione 4.3 [105]: esso contiene infatti alcune pagine di listato commentato del classico (per l'ambiente smart card) applet per la gestione di un conto bancario (*Wallet*) direttamente sulla carta; in pratica le operazioni consentite sono quelle di deposito, prelievo e controllo della somma presente sul conto dopo aver sbloccato la carta tramite un normale PIN. Tale esempio si è dimostrato molto "popolare" specialmente su internet.

Durante la scrittura sono state seguite inoltre alcune linee guida fornite anch'esse nel precedente manuale di programmazione, al fine di rendere il programma più efficiente possibile e di minimizzare i requisiti su stack, heap e riducendola memoria

EEPROM occupata dall'applet. Questo promemoria può essere riassunto nei seguenti punti:

- ◇ Visibilità locale: non creare oggetti o array con visibilità locale, visto che ciascuna istanziazione di questi tipi dato alloca memoria; la carta non fornisce le funzionalità di garbage collection e quindi la memoria occupata non verrà mai liberata.
- ◇ Variabili *final* e *static*: usare quando possibile queste parole chiave nella dichiarazione delle variabili, visto che questa accortezza diminuisce la dimensione dell'istanza ed aumenta le prestazioni. Questo perché con *static* esiste una sola incarnazione del campo della classe anche se esistono istanze (oggetti) della classe. Anche l'uso del metodo *final*, che serve per definire campi in sola lettura (costanti), rappresenta un aiuto per l'ottimizzatore del codice: sapendo che il suo valore non potrà cambiare nel corso dell'esecuzione potranno essere prese decisioni più precise ed efficienti.
- ◇ Riutilizzo delle variabili: anche se poco elegante dal punto di vista dello stile di programmazione, in un ambiente dalle caratteristiche ristrette può risultare utile riusare le stesse variabili (chiaramente quando possibile) per memorizzare valori dello stesso tipo; in questo modo viene risparmiato per esempio spazio sullo stack. Anche definire i campi delle classi come *public* può servire: un campo dichiarato *private* ha infatti bisogno di due metodi (tipicamente chiamati *set* e *get*) per poterci leggere e scrivere sopra, mentre un campo dichiarato *public* può essere acceduto dall'esterno anche tramite il suo solo nome; questo fa risparmiare

spazio sullo stack dato che viene chiamato un metodo di meno, anche se tutto ciò va a scapito della chiarezza del codice.

- ◇ Tipi primitivi: invece di creare oggetti dai tipi primitivi, è meglio usare loro direttamente; per esempio, se viene creato un oggetto che incapsula un tipo dato primitivo, il carico sulle risorse può superare le funzionalità guadagnate.
- ◇ Gerarchia delle classi: è importante mantenere inoltre la gerarchia delle classi molto semplice, dato che la sua complessità porta a ulteriori problemi di spazio e prestazioni.
- ◇ Semplicità nei metodi: non utilizzare più di 3 o 4 parametri nella definizione dei metodi delle classi e innestare poche loro chiamate.
- ◇ *switch-case*: usare con parsimonia questo costrutto, dato che la sua compilazione genera un bytecode molto verboso.

Oltre a queste note informative prettamente di ottimizzazione, dobbiamo ricordare che comunque esistono delle limitazioni nel linguaggio e nella Virtual Machine rispetto all'ambiente genitore Java; per quanto riguarda il primo punto non sono supportati il Dynamic class loading, il security manager (`java.lang.SecurityManager`), i thread, l'object clonino e alcuni aspetti del controllo dell'accesso al package, le parole chiave *native*, *synchronized*, *transient*, *volatile*, *strictfp* sono escluse dal linguaggio Java Card™; delle classi API basilari sono supportate solo *Object* e *Throwable* anche se molti dei loro metodi non sono presenti ed allo stesso modo non sono supportati i tipi *char*, *double*, *float*, e *long*, oppure gli array multidimensionali (il tipo *int* è opzionale). Le limitazioni della JVM consistono invece nel fatto che un package può riferire al massimo altri 128 package, un package può avere fino a 255 classi e interfacce ed il proprio nome

non può occupare più di 255 byte; una classe può implementare fino a 15 interfacce e a sua volta una interfaccia può ereditare al massimo da 14 altre interfacce; un package può avere fino a 256 metodi statici se contiene un applet (*applet package*) o 255 se non ne contiene (*library package*); un'istanza di classe può contenere al massimo 255 campi e può implementare fino a 128 metodi di istanza pubblici o protetti; il massimo numero di variabili locali ad un metodo non può superare 255 (una variabile intera conta per due) e un comando switch è limitato a non più di 65536 casi [124]. Una cosa molto importante da ricordare inoltre, è che essendo mancanti le funzionalità di garbage collection, i nuovi oggetti istanziati non potranno essere cancellati rimanendo così per sempre all'interno della carta dato che il ciclo di vita della Java Virtual Machine termina in pratica con la sua distruzione. La memoria EEPROM nella quale vengono allocati gli oggetti verrebbe quindi velocemente e irrimediabilmente consumata interamente nel caso fossero previste nuove allocazioni da parte di ripetute invocazioni dei metodi dell'applet; a quel punto si renderebbe necessaria la cancellazione dell'intera applicazione che ha allocato tali oggetti.

7.2 Java CardTM Biometric API.

In questo paragrafo verranno presentate più approfonditamente le Java CardTM Biometric API [95], che sono state già introdotte nel paragrafo 4.5 e che sono state utilizzate dal nostro programma realizzato su Java CardTM per la gestione delle funzioni di enrollment e matching dell'algoritmo di verifica.

Le JC Biometric API sono costituite da tre interfacce (figura 7.5 da [95]) e due classi che rientrano tutte all'interno del package *org.javacardforum.javacard.biometry*:

l'interfaccia *BioTemplate* consiste nell'interfaccia base da cui estendono le altre due, che si chiamano

OwnerBioTemplate e *SharedBioTemplate*. La prima aggiunge le funzionalità per la registrazione (enrollment) dei template biometrici, mentre la seconda, che estende anche *javacard.framework.Shareable*, fornisce i mezzi per accedere alle funzionalità non ristrette, come ad esempio le funzioni di matching, di controllo del risultato della verifica, del recupero del numero di tentativi ancora consentiti e del tipo di template biometrico implementato dall'algoritmo (fingerprint, iride, ecc...). Un manager/server può implementare questa interfaccia con un proxy a tali funzioni, consentendo la verifica da parte delle applicazioni clienti, ma “schermando” allo stesso tempo i metodi relativi alla registrazione del template; senza questa caratteristica, un'applicazione client avrebbe potuto effettuare un cast del riferimento a disposizione e avere così accesso all'enrollment potendo aggirare in questo modo le misure di sicurezza; nella nostra implementazione abbiamo realizzato questo proxy sicuro realizzando la classe *ProxyBioTemplate* che estende appunto *SharedBioTemplate*. La relazione tra le tre interfacce precedentemente citate può essere riassunta dalla seguente figura:

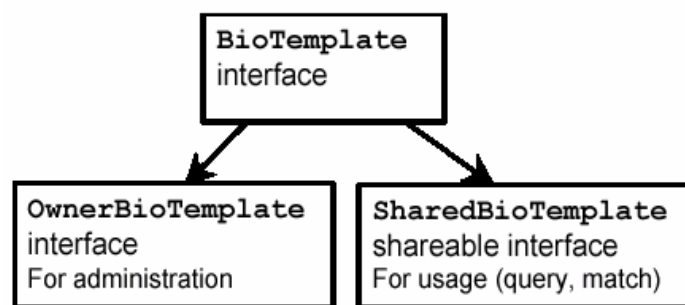


Figura 7.5
Interfacce nelle Java Card™ Biometric API con relazione di ereditarietà [95].

Abbiamo quindi realizzato la classe *JMOCOwnerBioTemplate* che estende l'interfaccia *OwnerBioTemplate* e ne abbiamo implementato tutti i metodi per offrire i mezzi per accedere alle funzionalità biometriche del template.

Le due classi delle API sono invece costituite dall'eccezione *BioException* e da *BioBuilder*. La prima estende *javacard.framework.CardRuntimeException* e serve per gestire gli errori che possono essere rilevati dai metodi introdotti in questo package. La seconda, con il suo unico metodo, serve per costruire nuovi oggetti *OwnerBioTemplate*: in pratica è un costruttore di template biometrici; chiaramente, entrambe queste due classi sono entrate a far parte della nostra applicazione scrivendone i metodi.

Solitamente, come viene ripreso anche in queste API, un'applicazione biometrica consiste concettualmente in un applet servente/gestore ed uno cliente. Il primo ha il compito di gestire l'enrollment del template biometrico e di fornire attraverso un oggetto proxy (che implementa l'interfaccia *SharedBioTemplate*) l'accesso protetto al secondo. In questo modo possono essere aggiunti nel corso del tempo anche altri applet client senza il pericolo che possano modificare il template biometrico. Le API forniscono solo le funzionalità di base, ma altre possono essere aggiunte nei package del cliente o del servente; a proposito di package, per molte applicazioni, server e client possono essere inclusi in uno solo di questi, come viene mostrato anche nella figura 7.6 (tratta da [95]) che riassume ancora l'architettura dell'ambiente, questa volta dal punto di vista delle unità di programmazione: il diagramma mostra come i vari applet clienti che rappresentano le varie applicazioni sulla carta che hanno necessità di usufruire del riconoscimento biometrico, si interfaccino con l'applet manager dei dati biometrici utilizzando le Java Card™ Biometric API (anche per le funzioni di registrazione del template). L'applet server a sua volta, tramite questa interfaccia standard, si rivolge alla vera e propria implementazione dell'algoritmo di riconoscimento (che possono essere

anche più di uno e basati su tecnologie biometriche diverse) per ottenere il risultato del confronto. Per molte applicazioni inoltre, server e client possono essere combinati in un solo package come viene mostrato dalla relazione rappresentata dalla freccia tratteggiata.

Nella nostra applicazione abbiamo realizzato solo l'applet servente delle funzionalità biometriche che risponde ai messaggi provenienti dall'esterno (APDU dal lettore/scrittore di carte) che comandano principalmente la registrazione di un template o la verifica trasportando anche i dati consistenti nei template: nel primo caso il template da registrare all'interno della carta (al massimo uno nella nostra implementazione), mentre nel secondo caso il template con cui comparare quello già precedentemente registrato. Con un altro messaggio proveniente dal lettore/scrittore l'applicazione su PC richiederà il risultato del match per sapere se l'impronta è stata verificata positivamente. Non è stato necessario quindi realizzare anche applet clienti dato che non interessavano direttamente le finalità del nostro progetto; attraverso la classe *ProxyBioTemplate* (che è stata comunque realizzata) è possibile sfruttare le funzionalità biometriche da parte di un eventuale futuro applet cliente, senza modificare o aggiungere codice sul servente.

Come accennato anche precedentemente, questo schema di suddivisione delle classi/interfacce che compongono le API, viene ripreso dalla struttura delle classi/interfacce per la gestione del PIN all'interno del package *javacard.framework* facente parte di *Javacard Platform specification* (fino alla corrente versione 2.2). Oltre alla classe *PINException* per la gestione degli errori infatti, sono presenti l'interfaccia *PIN*, che assomiglia fortemente a *SharedBioTemplate* e la classe *OwnerPIN* che implementa *PIN* per le funzioni di gestione del codice segreto allo stesso modo di come *OwnerBioTemplate* implementa *SharedBioTemplate* per la gestione del template; chiaramente i metodi introdotti per l'uso del PIN sono in numero minore e anche la struttura globale è meno complessa, vista la semplicità dell'informazione rappresentata

(un codice numerico di poche cifre) e la facilità nella verifica, costituita da un controllo cifra per cifra tra le due stringhe di numeri.

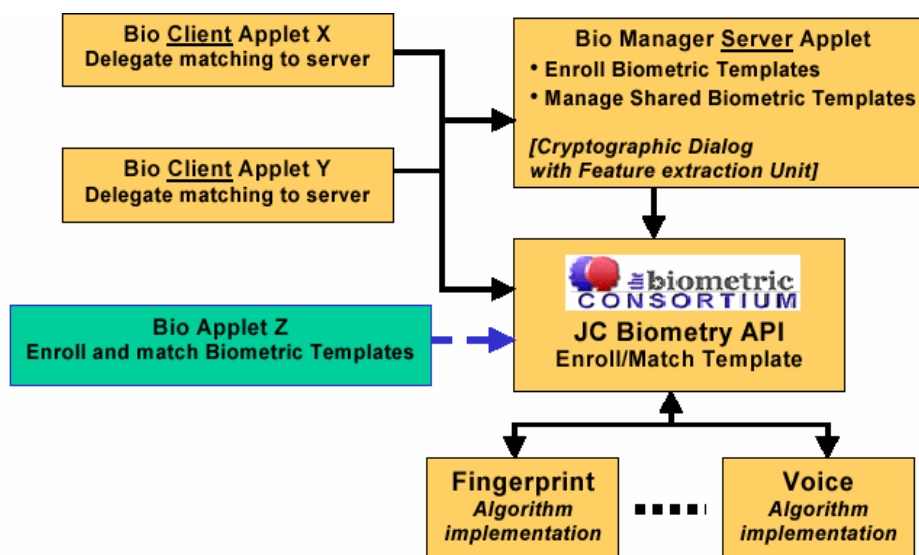


Figura 7.6
Possibili schemi di implementazione di applet biometrici [95].

In pratica i passi da seguire per lo sviluppatore (o meglio più sviluppatori che lavorano indipendentemente) sono quelli di scrivere il codice dell'algoritmo di matching e rendere disponibili le funzioni per mezzo dei metodi delle API. Al tempo stesso, sempre grazie alla standardizzazione di queste, è possibile sviluppare contemporaneamente il codice del server e del client. Queste API possono essere usate come qualsiasi altra Java Card™ API [95] ed i sorgenti di stub per la compilazione e i file di export per la conversione possono essere attualmente scaricati senza contributi o registrazioni di alcuna sorta dal sito di JCF [98].

Per rendere più chiara la descrizione della API, segue ora la lista di tutti i metodi suddivisi per unità di programmazione che li contiene, con una breve descrizione per il loro utilizzo tratta dalla documentazione ufficiale scaricabile sempre dal sito di JCF [96].

Nel seguito si farà riferimento al template di *reference* come quello acquisito durante la registrazione al rilascio della carta all'utente autorizzato e quello *candidate*, come il template acquisito ogni volta per il riconoscimento.

Classe BioException:

Estende *javacard.framework.CardRuntimeException* e le sue istanze vengono lanciate dai metodi del package in caso di errore. La classe prevede 5 codici di errori predefiniti contenuti nei suoi campi:

- ◇ *ILLEGAL_USE*: indica che uno o più parametri passati al metodo rappresentano valori non consentiti.
- ◇ *ILLEGAL_VALUE*: indica genericamente che i dati incontrati non sono compatibili; viene usato per rilevare che i template di *reference* o *candidate* non hanno un formato conosciuto.
- ◇ *INVALID_DATA*: rilevato se il costruttore di template (metodo *bioBuilder*) non supporta il tipo di template indicato.
- ◇ *NO_SUCH_BIO_TEMPLATE*: questo codice di errore serve per indicare che nessun template è stato registrato sulla carta, o che il template di riferimento non è stato ancora inizializzato (qualora preveda un'inizializzazione).
- ◇ *NO_TEMPLATES_ENROLLED*: indica che il metodo non dovrebbe essere stato invocato, basandosi sullo stato attuale della carta.

Oltre al costruttore, l'altro metodo di questo oggetto è *throwIt*, che prende come parametro il codice dell'errore rilevato e lancia l'istanza del JCRE per non consumare risorse.

Classe BioBuilder:

L'unico metodo di questa classe (dichiarata *final*) è costituito da *buildBioTemplate*, che prende come parametri il numero di tentativi falliti consecutivi che possono essere effettuati ed il tipo di template biometrico da costruire; quelli supportati sono *FINGERPRINT* che si basa sull'identificazione basata sull'impronta digitale di un qualsiasi dito (e che interessa direttamente il nostro algoritmo), o *FINGER_GEOMETRY*, che si basa invece sulla forma generale del dito e non si deve quindi scambiare erroneamente per l'impronta digitale ed altri ancora come *BODY_ODOR*, *VOICE_PRINT*, *IRIS_SCAN*, ecc.

Molti di questi template, come la scansione dell'iride, richiedono prestazioni di calcolo e memoria che attualmente la tecnologia smart card difficilmente può garantire per offrire allo stesso tempo una affidabilità accettabile nel riconoscimento. Ciò nonostante sono stati inclusi per una futura evoluzione dei chip.

Il metodo *bioBuilder* restituisce in caso di successo il nuovo template biometrico creato. Nella nostra implementazione restituisce invece errore nel caso i due parametri forniti non siano corretti: per esempio se il tipo di template è diverso da *FINGERPRINT* il metodo solleva un'eccezione con codice *BioException.NO_SUCH_BIO_TEMPLATE*, oppure se il numero di tentativi falliti viene impostato per un valore minore di 1 o maggiore di 5 viene sollevata un'eccezione *BioException.ILLEGAL_VALUE*.

Interfaccia BioTemplate:

Rappresenta l'interfaccia base per tutti i template biometrici e fornisce a chi la utilizza i mezzi per accedere alle funzionalità biometriche ed ha come sotto-interfacce *OwnerBioTemplate* e *SharedBioTemplate*. Nella nostra applicazione sono state implementate le sue due sotto-interfacce e non direttamente essa.

Per quanto riguarda la fase di match, vero cuore dei servizi offerti, una corretta sessione prevede nell'ordine le chiamate *initMatch-[match]*; l'invocazione di *initMatch* è quindi obbligatoria, mentre quella di *match* è opzionale e può essere ripetuta più volte consecutive all'interno della stessa sessione, tipicamente per fornire tutti i dati biometrici del candidate template all'algoritmo di matching, dato che difficilmente potranno essere tutti inclusi nel solo array di byte di *initMatch*.

Per quanto riguarda il valore restituito dai metodi di match, se è maggiore o uguale al valore definito da *MINIMUM_SUCCESFUL_MATCH_SCORE*, allora il match è ritenuto positivo; se è compreso tra 0 e *MINIMUM_SUCCESFUL_MATCH_SCORE*, i due template non sono stati riconosciuti come appartenenti allo stesso dito; se il valore di ritorno è invece uguale a *MATCH_NEEDS_MORE_DATA* significa che la sessione di matching deve continuare (con un'altra invocazione del metodo *match*), dato che le informazioni fornite non sono ancora sufficienti per terminare la sessione.

MINIMUM_SUCCESFUL_MATCH_SCORE e *MATCH_NEEDS_MORE_DATA* sono costanti di tipo short contenute in questa interfaccia ed hanno valore rispettivamente 16384 (metà del valore massimo di uno short) e -1; altri valori negativi sono riservati per un futuro uso. La definizione del *matching score* non viene chiaramente inclusa nelle API perché deve dipendere dall'algoritmo. I metodi dell'interfaccia sono:

- ◇ *getBioType*: metodo che non ha parametri in ingresso e restituisce il tipo del template biometrico; i tipi sono quelli elencati nella descrizione della classe *bioBuilder*.
- ◇ *getPublicTemplateData*: serve per acquisire, se presente, la parte pubblica del template di reference, o più verosimilmente una sua parte. Prende come parametri l'array di byte di ritorno che conterrà i dati biometrici, altri parametri utili per la scrittura su tale array (come l'offset), e anche l'offset sulla parte pubblica del template che si vuole ricevere. Ritorna il numero di byte scritti sull'array-risultato o il valore 0, se i dati pubblici non sono disponibili. Può sollevare un'eccezione con codice *BioException.NO_TEMPLATES_ENROLLED* se il template di reference non è stato ancora inizializzato. Nella nostra realizzazione questo metodo ritorna sempre l'ultima minuzia che costituisce il template e può essere utilizzato in fase di enrollment per controllare che l'ultima minuzia inviata alla carta sia stata registrata correttamente.
- ◇ *getTriesRemaining*: non prende parametri, ma ritorna il numero di tentativi ancora disponibili per l'operazione di match prima che il template di reference venga bloccato.
- ◇ *getVersion*: ritorna la versione e l'ID dell'algoritmo di matching implementato sulla carta: dipende chiaramente dal realizzatore dell'applet e nel nostro caso ritorna il valore 1.
- ◇ *initMatch*: metodo principale insieme a *match*. Ha lo scopo di inizializzare o reinizializzare una sessione di verifica (o match): se il template non è bloccato, prima di ogni altra operazione deve essere resettato il flag di validazione del template, il contatore di tentativi deve essere decrementato e se raggiunge il

valore 0, deve essere bloccato. Come parametri, viene passato un array di byte contenente il template candidate (o una sua parte); il metodo può sollevare un'eccezione con codice *BioException.INVALID_DATA* se questi dati non si uniformano al formato supportato, o *BioException.NO_TEMPLATES_ENROLLED* se il reference template non ha superato la fase di inizializzazione. Il ritorno di questo metodo rappresenterà il valore la necessità di ulteriori dati (*MATCH_NEEDS_MORE_DATA*) o direttamente il valore che rappresenta il successo o l'insuccesso dell'algoritmo di match: maggiore di *MINIMUM_SUCCESSFUL_MATCH_SCORE* equivale ad un riconoscimento positivo, minore uno negativo. Per quanto riguarda la nostra realizzazione, il metodo non trasporta dati biometrici riguardanti il template, ma serve solo ad iniziare la fase di match settando alcuni flag: ritorna sempre e comunque il valore *BioTemplate.MATCH_NEEDS_MORE_DATA*. Oltre all'eccezione *BioException.NO_TEMPLATES_ENROLLED*, può sollevare *BioException.ILLEGAL_USE* se sono terminati i tentativi consecutivi di match messi a disposizione.

- ◇ *match*: è identico al metodo *initMatch* per parametri e valori di ritorno; differentemente da questo può sollevare un'eccezione con codice *BioException.ILLEGAL_USE* se viene invocato al di fuori di una sessione di match (se non è stato chiamato precedentemente *initMatch*). Questo metodo deve essere invocati ripetutamente fino a che i dati necessari all'algoritmo non sono stati tutti passati: a tal punto, il valore di ritorno non sarà più *MATCH_NEEDS_MORE_DATA*, ma il valore del risultato della verifica. Nella nostra implementazione, dopo un invocazione del metodo *initMatch* per segnalare l'inizio della sessione di match, il metodo *initMatch* viene chiamato più

volte per inviare alla carta tutte le minuzie del template candidato necessarie per la verifica dell'algoritmo. Il metodo potrà sollevare quindi le eccezioni di *BioException.NO_TEMPLATES_ENROLLED* se nessun template di riferimento è già stato registrato, *BioException.ILLEGAL_USE* se non è stato segnalato l'inizio di una fase di match col metodo *initMatch*, *BioException.INVALID_DATA* se il formato delle minuzie ricevute è irregolare. Quando con questo metodo sono state inviate un numero prestabilito di minuzie (le precedenti invocazioni restituivano il valore *BioTemplate.MATCH_NEEDS_MORE_DATA*), l'ultima scatena il processo di match vero e proprio e ritorna il valore che rappresenta la somiglianza tra le due minuzie.

- ◇ *isInitialized*: metodo senza parametri in ingresso; ritorna un booleano che indica se il template di reference è stato registrato sulla carta ed è pronto per le operazioni di matching.
- ◇ *isValidated*: metodo senza parametri in ingresso; ritorna un booleano che indica se l'ultima verifica biometrica sul template avvenuta dopo l'ultimo reset della carta o dopo l'ultima invocazione del metodo *reset*, ha avuto esito positivo o negativo: in pratica, se l'ultimo match effettuato ha riconosciuto che i due template di reference e candidate appartenevano allo stesso dito.
- ◇ *reset*: metodo che non prende parametri in ingresso e che non ritorna valori; serve per resettare il flag che indica la validazione dell'ultima operazione di match. Se viene chiamato all'interno di una fase di spedizione dati nel corso del match (più chiamate di *match*) annulla completamente il match in corso.

Interfaccia *OwnerBioTemplate*:

Estende l'interfaccia *BioTemplate* e i suoi metodi realizzano le funzioni necessarie all'applicazione che gestisce il template durante la registrazione del template di reference e quindi non devono essere poter utilizzate dalle applicazioni cliente.

Nella classe *JMOCOwnerBioTemplate* sono stati implementati sia i metodi esclusivi di *OwnerBioTemplate*, sia quelli ereditati da *BioTemplate* descritti precedentemente.

Una corretta sequenza di registrazione prevede la chiamata in ordine dei metodi *init*-[*update*]-*doFinal*: mentre il primo e l'ultimo sono obbligatori, *update* è opzionale.

I metodi dell'interfaccia sono:

- ◇ *init*: è il primo metodo che deve essere chiamato per la registrazione del template, ma può essere utilizzato anche per inizializzare l'aggiornamento di un template già precedentemente acquisito sulla carta; come conseguenza provoca l'invalidazione del template e quindi impedisce operazioni di verifica su di esso. I suoi parametri comprendono il nuovo template o parte del nuovo template. Solleva un eccezione con codice *INVALID_DATA* se i parametri passati contengono errori.
- ◇ *update*: continua la registrazione del template di reference iniziata con il metodo *init* e dovrebbe essere usato solo nel caso il template occupasse più di un array di byte (quello passato appunto col metodo *init*); può essere dunque chiamato diverse volte. I parametri sono esattamente uguali a quelli del metodo *init* in numero, tipo e utilizzo. Solleva un eccezione con codice *INVALID_DATA* se i

parametri passati contengono errori e *ILLEGAL_USE* se il sistema non si aspetta la chiamata di questo metodo (per esempio se non è stato invocato prima *init*)

- ◇ *doFinal*: Termina la registrazione del template di reference avviata con il metodo *init* e continuata con varie chiamate di *update* (in realtà non obbligatorio). Le operazioni finali da compiere sono, tra le altre, marcare il template come inizializzato e pronto per operazioni di matching e resettare il contatore dei tentativi effettuabili; un eventuale controllo del template passato può essere effettuato a questo punto. Il metodo non ha parametri e ritorna errore di *INVALID_DATA* se il formato del template non è corretto o *ILLEGAL_USE* se viene chiamato fuori sequenza.
- ◇ *resetUnblockAndSetTryLimit*: ristabilisce lo stato interno. Sblocca infatti il template, aggiorna il valore massimo di tentativi falliti consecutivi (che è rappresentato dall'unico parametro preso in ingresso) e porta a questo valore il contatore di tali tentativi. Il metodo solleva un'eccezione con codice *ILLEGAL_VALUE* se il nuovo valore massimo di tentativi è minore di 1.

Interfaccia *SharedBioTemplate*:

Estende contemporaneamente sia *BioTemplate* sia *javacard.framework.Shareable*. Fornisce i mezzi per accedere alle funzionalità biometriche non ristrette, come per esempio il servizio di matching, da parte delle applicazioni cliente. Il manager/servente dei dati biometrici implementerà questa interfaccia con un oggetto proxy e renderà così utilizzabili solo le funzioni che non modificano i dati da proteggere, come potrebbero fare invece le funzioni di registrazione del template.

L'interfaccia (che abbiamo implementato nella classe *ProxyBioTemplate*) non comprende metodi aggiuntivi e quindi la sua scrittura consisterà nello specificare i corpi

di quelli presenti in *BioTemplate*; il corpo consisterà nel richiamare lo stesso metodo con gli stessi parametri, ma questa volta sull'oggetto che implementa *OwnerBioTemplate* come nell'esempio sottostante:

```
public class ProxyBioTemplate implements SharedBioTemplate {

// ----- Data -----
OwnerBioTemplate fullBio;           // Package accessible and not directly shared
                                     outside the package

// ----- Methods -----
public boolean isValidated() { return fullBio.isValidated(); }

public boolean isInitialized() { return fullBio.isInitialized(); }

public short initMatch(byte[] applicantTemplate, short offset, short length) {
    return fullBio.initMatch(applicantTemplate, offset, length); }

. . . . . (e tutti gli altri metodi)
```

In conclusione, le JCF Biometric API sono anche compatibili con un'eventuale quantizzazione del punteggio di matching (fino alla restituzione di un risultato concettualmente booleano (matching avvenuto positivamente o negativamente) che previene in questo modo attacchi di tipo *hill climbing* [99], nei quali, avendo la disponibilità del risultato, si corregge gradualmente il template (se si è chiaramente in grado di farlo un numero sufficiente di volte) per farlo verificare positivamente con quello presente sulla carta. L'argomento della quantizzazione non è però affrontato dalle specifiche diffuse, ma deve essere risolto all'interno dell'algoritmo di matching, poiché le API fanno solo da vettore di informazioni tra algoritmo e applicazione cliente. Molti altri dettagli implementativi della JC Biometric API sono descritti direttamente nei commenti

del codice da cui è possibile generare la documentazione Javadoc [137] (vedi Appendice D).

7.3 Scelte implementative dell'algoritmo su Java Card™.

Il porting del codice dall'ambiente Java all'ambiente Java Card™ ci ha obbligato chiaramente ad alcune semplificazioni dovute alla scarsità di risorse della smart card e conseguentemente alla minore potenza del secondo linguaggio di programmazione.

Prima di tutto è stato necessario abbandonare le strutture dati offerte dalle API Java, come *HashMap* usate per implementare le stelle uscenti dei nodi del grafo, oppure anche *Vector* per implementare la lista delle minuzie di un template; queste strutture sono state sostituite da liste realizzate con vettori di dimensione fissa e puntatori ad inserzione ed estrazione., rendendo così più complicata la loro manipolazione. L'algoritmo di assegnamento di costo minimo che costituisce la base del nostro algoritmo di verifica di impronte digitali è stato comunque perfettamente realizzato in linguaggio Java Card, anche se è stato sicuramente più complicato da programmare ed è stata necessaria un'attenzione maggiore a causa delle già esposte difficoltà dovute alle strutture dati e ad altri fattori sperimentati in fase di sviluppo e descritti di seguito.

Dato che, come specificato in 7.1.4, un oggetto istanziato col comando *new* occupa memoria EEPROM della carta per tutto il ciclo di vita della Java Virtual Machine, ci siamo dovuti imporre di allocare in fase di installazione dell'applet (dal metodo *install* dell'applet) tutti gli oggetti che sono necessari al funzionamento dell'algoritmo di verifica ed in generale tutti gli oggetti che sono utilizzati dall'esecuzione dell'applicazione.

La penalizzazione maggiore dovuta alla mancanza di un *Garbage Collector* consiste dunque in un riuso forzato degli stessi oggetti che a sua volta porta a dover introdurre codice aggiuntivo per la gestione degli stessi, visto che il programmatore deve essere attento a non lasciare informazione “vecchia” servita nel corso di un precedente uso: questo si traduce in generale ad una reinizializzazione di alcune variabili utilizzate ad un valore iniziale necessario ad un corretto funzionamento dell’algoritmo.

L’esempio più importante è rappresentato dai grafi (quello delle minuzia-minuzia e quello vicino-vicino) su cui deve essere trovato l’assegnamento di costo minimo oppure anche dalle stesse strutture dati introdotte dall’assegnamento, come i vettori delle etichette dei nodi parente o delle etichette dei costi utilizzati dalla procedura *Shortest Path Tree* adottata per trovare i cammini minimi sui precedenti grafi.

Anche per quanto riguarda i semplici tipi dato primitivi, il linguaggio Java Card™ ci ha costretto a sostituire tutti quelli non previsti cercando di sostituirli con quelli offerti: in pratica il tipo con più informazione risulta essere lo *short*. Nella pratica abbiamo dovuto convertire i reali *float* utilizzati nell’implementazione PC per rappresentare le feature che caratterizzano i vicini delle minuzie (vedi 5.2.1): questa necessaria perdita di informazione si può ripercuotere sulle prestazioni dell’algoritmo che deve confrontarsi quindi con visione meno precisa dei vicini da parte delle minuzie. Anche per quanto riguarda i risultati dell’algoritmo di assegnamento (e quindi del valore finale di *match*), abbiamo dovuto far “entrare” tali valori nello spazio di rappresentazione di uno *short*. Inoltre, per diminuire l’occupazione di memoria volatile, in realtà lo heap dell’applicazione, abbiamo anche sostituito dove possibile *byte* al posto degli *int* usati in ambiente Java su PC: per esempio questo è stato eseguito per tutti i casi di cicli *for* con sicuramente meno iterazioni di 127, valore massimo per un *byte*.

Sempre per diminuire l'occupazione di memoria abbiamo cercato di riusare le stesse variabili per memorizzare in momenti diversi dell'esecuzione valori anche molto diversi tra loro per significato (ma chiaramente, sempre memorizzabili nello stesso tipo): abbiamo per esempio utilizzato la stessa variabile indice per tutti i cicli *for* dello stesso metodo, oppure per memorizzare prima il valore del match ottenuto dall'algoritmo e poi quello da restituire all'esterno della carta con la APDU di risposta. Tutto questo per cercare chiaramente di limitare il numero di variabili da allocare in memoria.

Per quanto riguarda invece il risparmio in termini di tempo di esecuzione, abbiamo dovuto spostare il più possibile da EEPROM a RAM le strutture dati da modificare nel corso dell'esecuzione dell'algoritmo di verifica: tra la scrittura sulla prima memoria rispetto alla seconda c'è infatti un fattore 1000 di differenza, che nel caso di numerose scritture può gravare notevolmente sulle prestazioni in termini anche di decine di minuti. Nella prima verifica di test effettuata avevamo impiegato infatti ben 45 minuti circa, visto che avevamo dichiarato anche gli indici dei *for* come campi della classe in cui erano presenti i metodi che li utilizzavano (sempre per risparmiare memoria RAM, dato che i campi dell'oggetto vengono memorizzati su EEPROM). Abbiamo quindi dovuto mantenerci in equilibrio tra queste due necessità, spostando più strutture dati possibile da EEPROM a RAM, ma cercando allo stesso tempo di non superare le capacità dello heap: ciò provocherebbe un'eccezione a tempo di esecuzione riportato nella APDU di risposta con codice SW 60 00. Gran parte del tempo lo abbiamo recuperato dichiarando gli *array* di *byte*, di *short* o di *boolean* come transienti con dei metodi appropriati, per esempio per gli *short* con *makeTransientShortArray*; questi metodi servono per creare array di tipi dato primitivi in memoria volatile invece che su EEPROM, ma non ne esiste il corrispondente per la definizione di array di oggetti. In questo modo siamo riusciti ad arrivare ad un'esecuzione in 7 minuti che risulta essere

ancora un tempo troppo elevato per una tecnologia di *match on card* in cui un utente deve avere una risposta entro pochi secondi: d'altra parte la memoria RAM presente nella smart card a nostra disposizione risulta essere abbastanza e quindi il problema può essere superato con l'adozione di altre carte in commercio.

Un'altra importante semplificazione effettuata per risparmiare spazio sulla carta è consistita nel condensare la descrizione del vicino, così come viene osservato dalla minuzia centrale (vedi paragrafo 5.2.1), con un unico valore invece che con i quattro valori rappresentanti le quattro feature: in pratica abbiamo utilizzato un unico *short* riassunto delle quattro feature e che viene calcolato normalizzandole, pesandole e sommandole nell'applicazione host (quella su PC) prima di inviare il template alla carta. Poi nell'algoritmo di assegnamento di costo minimo sulla carta, il costo di un arco tra vicino e vicino viene calcolato trovando la differenza tra i due riassunti delle feature; nell'implementazione in Java normalizzavamo invece le differenze tra le corrispondenti varie feature, per esempio distanza euclidea con distanza euclidea, e sommavamo poi tutto insieme con l'aiuto di pesi per trovare il costo dello stesso arco.

Di seguito sono riportati i codici di comando e i parametri aggiuntivi delle APDU che l'applicazione supporta, con una breve descrizione di quello che viene effettuato sulla carta; per tutte le APDU il CLA (la classe del comando, vedi paragrafo 7.1) è il valore esadecimale 0x03, mentre il valore INS (l'identificatore del comando) viene riportato di seguito per ciascun comando:

- ◇ *BIO_VERIFY_INIT_MATCH* (INS=0x01): con questo comando viene avvertito l'applet gestore della carta che sta iniziando una sessione di verifica tra il template già presente sulla carta ed un altro template che verrà spedito tramite

successive esecuzioni del comando *BIO_VERIFY_MATCH*; deve essere obbligatoriamente chiamato all'inizio di ogni nuova sessione. Una APDU contenente questo comando spedita all'interno di una sessione di verifica, ne provoca la terminazione e ne avvia un'altra totalmente nuova. La APDU di risposta non contiene mai nessun campo dati. Il comando invoca il metodo *initMatch* delle Biometric API (paragrafo 7.2).

- ◇ *BIO_VERIFY_MATCH* (INS=0x02): serve per continuare la sessione di matching e viene chiamato più volte per spedire tutte le informazioni sulle minuzie del template candidate alla carta. Quando il numero di minuzie spedite raggiunge un certo valore stabilito (nella corrente implementazione 25), l'ultima invocazione provoca automaticamente anche l'esecuzione dell'algoritmo di matching e restituisce con la APDU di risposta il valore del finale del match. Nelle APDU di risposta delle precedenti invocazioni del comando nel corso della sessione, il campo dati contiene sempre il valore specificato nelle API *MATCH_NEEDS_MORE_DATA*, in quanto il template candidato deve essere ancora in parte spedito. Il comando invoca il metodo *match* delle Biometric API.
- ◇ *BIO_INS_TEMPLATE_INIT* (INS=0x03): con questo comando viene iniziata la sessione di registrazione del template di riferimento, e cioè quello appartenente al possessore della carta e registrato su di essa la momento del rilascio all'utente, . Il comando invoca il metodo *init* delle Biometric API.
- ◇ *BIO_INS_TEMPLATE_UPDATE* (INS=0x04): serve per spedire tutte le minuzie del template di riferimento e salvarle all'interno dello stato della carta: può quindi essere chiamato più volte per inviare tutta l'informazione, dal momento in cui è già stato eseguito *BIO_INS_TEMPLATE_INIT*. Il comando invoca il metodo *update* delle Biometric API.

- ◇ *BIO_INS_TEMPLATE_DOFINAL* (INS=0x05): dopo *BIO_INS_TEMPLATE_INIT* e *BIO_INS_TEMPLATE_UPDATE*, questo comando serve per chiudere la fase di registrazione del template di riferimento; questa fase può essere eseguita una volta sola in tutta la vita dell'applet e cioè quando la carta viene rilasciata all'utente. Questi ultimi tre comandi ritornano una APDU di risposta senza campo dati (solo SW). Il comando invoca il metodo *doFinal* delle Biometric API.
- ◇ *BIO_CHECK_VERIFY_RESULT* (INS=0x06): serve per controllare il risultato del precedente match eseguito sulla carta: la SW nella APDU di risposta indica con un appropriato codice se il match è stato positivo, oppure con un altro codice ha avuto esito negativo o se il match non è ancora avvenuto da quando la carta è stata nuovamente inserita nel lettore/scrittore. Il comando invoca il metodo *isValidated* delle Biometric API.
- ◇ *BIO_CHECK_ENROLLED* (INS=0x07): serve per controllare se il template di riferimento sia presente e pronto per il matching. Nella APDU di risposta viene settata una SW di avviso per avvertire se il controllo ha avuto esito negativo. Il comando invoca il metodo *isInitialized* delle Biometric API.
- ◇ *BIO_GET_PUBLIC_TEMPLATE_DATA* (INS=0x08): serve per controllare la parte resa pubblica del template e quindi visibile a tutti. La APDU di risposta contiene nel suo campo dati questa informazione accessibile anche all'esterno, solo nel caso sia stato già stato registrato sulla carta il template di riferimento. Il comando invoca il metodo *getPublicTemplateData* delle Biometric API.
- ◇ *BIO_UNBLOCK_REFERENCE* (INS=0x09): questo comando serve per sbloccare le funzionalità biometriche sul template registrato nel caso la carta sia stata bloccata a causa di un numero prefissato di match consecutivi falliti;

chiaramente si rende necessario bloccare la carta nel caso un eventuale impostore cerchi di spacciarsi per il reale possessore della carta. Questo comando è stato realizzato solo per eseguire i test funzionamento e sarebbe naturalmente da disabilitare in una reale applicazione sicura. La APDU di risposta contiene solo la SW. Il comando invoca il metodo *resetUnblockAndSetTryLimit* delle Biometric API.

Come si può notare abbiamo scelto di far corrispondere a ciascun comando l'invocazione di un metodo delle API biometriche per offrire un diretto e più potente controllo sulle funzioni offerte: non tutti i metodi delle API hanno un corrispondente comando esterno che gli invoca, dato che queste ci sono sembrati principali servizi da offrire al di fuori della carta.

Per la realizzazione dell'applet abbiamo implementato le Java Card™ Biometric API [95] di Java Card™ Forum descritte nel paragrafo 7.2 ed in particolare l'interfaccia *SharedBioTemplate*. In particolare abbiamo realizzato anche la classe proxy per consentire un accesso sicuro alle funzionalità biometriche da parte di eventuali altri applet presenti nella carta, anche se non era strettamente necessario visto che nei nostri esempi di funzionamento non c'era nessun'altra applicazione, ma i comandi per eseguire match e enrollment (e tutti gli altri descritti prima, come) venivano forniti dall'esterno (per esempio un applicazione su PC) tramite lettore/scrittore.

Anche in questo caso, come nella versione su PC, il risultato finale di match viene restituito già mediato rispetto al numero di minuzie realmente accoppiate dall'algoritmo, che può essere in realtà minore del massimo consentito (25). Un'altra cosa da dire a

proposito del risultato finale è che abbiamo scelto di ritornare sempre il valore *MINIMUM_SUCCESSFUL_MATCH_SCORE* specificato nelle Biometric API in caso di verifica positiva: il precedente valore, come specificato nella documentazione delle API, corrisponde al minimo valore per il quale il match viene ritenuto positivo; al contrario, se il match è negativo e le due impronte sono ritenute non appartenere allo stesso dito della stessa persona, il risultato ritornato al di fuori dell'applet è zero.

In appendice D viene riportato il codice della realizzazione dell'algoritmo su Java Card™ da noi effettuata. Il codice è stato scritto, compilato e testato in ambiente Java con *Netbeans IDE 3.4* [138] ed in seguito abbiamo utilizzato la piattaforma di sviluppo Cyberflex [89] di Schlumberger descritta nel paragrafo 7.1.3 per costruire il file .ijc dell'applicazione, scaricarlo sulla carta (l'occupazione risulta essere di 6407 byte su 32Kbyte di EEPROM), istanziare poi l'applet e testare il corretto funzionamento dei vari comandi messi a disposizione (come quello di enrollment e matching) spedendo le appropriate APDU di comando. Il corretto funzionamento dell'applicazione di verifica biometrica su impronta digitale è quindi stato accertato nella pratica anche in ambiente Java Card™; il lettore di smart card utilizzato nel corso di tutto lo sviluppo è stato ChipDrive 110 di Towitoko [140].

Capitolo 8

Conclusioni.

Nel corso di questo capitolo commenteremo prima le caratteristiche principali dell'algoritmo di riconoscimento di impronte digitali da noi progettato e descritto nel paragrafo 5.2 e che, per quanto ci è dato di conoscere, lo distinguono in parte da tutti gli altri presenti in letteratura; in seguito riepilogheremo le sue prestazioni e suggeriremo alcune possibili modifiche per un futuro lavoro di approfondimento e miglioramento globale; infine commenteremo le implementazioni in Java™ e Java Card™ e le problematiche risolte a tal riguardo.

Come già ricordato nel paragrafo 3.2, i principali problemi presentati dalla verifica di impronte digitali sono dovuti all'elasticità della pelle, che porta a deformazioni sempre differenti durante l'acquisizione dell'immagine del polpastrello, che oltre tutto non viene sempre premuto sullo scanner nella stessa posizione e con la stessa forza; inoltre sono presenti anche disturbi dovuti ad altri fattori come lo sporco, eventuali ferite e alcuni errori dovuti al software di acquisizione delle minuzie, che può non rilevare alcuni punti di singolarità o introdurre altri che in realtà non esistono, per esempio durante la fase in cui viene "ritagliata" la parte più importante dell'immagine. Alle

trasformazioni di traslazione, rotazione e cambio di scala si aggiungono quindi tutta una serie di piccole deformazioni nello schema delle creste che alcune volte, complice l'incertezza nell'apporre il dito sullo scanner, viene riprodotto solo parzialmente nell'immagine. Un algoritmo di fingerprint matching deve essere il più possibile robusto a questi disturbi e errori, senza però allo stesso tempo abbassare la soglia di sicurezza fornendo quindi falsi riconoscimenti.

Ricordiamo innanzi tutto che lo scopo principale di questa tesi era quello di progettare un nuovo algoritmo di fingerprint matching che fosse abbastanza veloce e semplice da poter essere implementato su una architettura Java Card™ caratterizzata da una potenza molto ridotta in termini di velocità di calcolo e di memoria volatile e non volatile a disposizione, rispetto ad un comune Personal Computer.

Partendo dalla “visione del mondo circostante” (“world view”) proposta da Fionn Murtagh in [72], siamo riusciti a costruire un algoritmo sostanzialmente insensibile a rotazioni e traslazioni: questa caratteristica è necessaria per realizzare una procedura di verifica affidabile nel campo del riconoscimento delle impronte digitali, e si è potuta ottenere mediante la descrizione di come una minuzia centrale, quella a cui facciamo riferimento ad un dato istante, “osserva” le altre minuzie sue vicine. Per far questo abbiamo utilizzato quattro caratteristiche (le feature descritte in 5.2.1) ispirateci dall'algoritmo in [69] e riguardanti in sostanza la distanza euclidea tra minuzia centrale e minuzia vicina, il numero di creste presenti tra le due, la differenza tra le angolazioni delle due minuzie ed infine l'angolo tra la retta indicante l'angolazione della minuzia centrale e la retta passante per le due minuzie. La robustezza alle trasformazioni nel piano è dovuta al fatto che se la minuzia centrale ruota o trasla, le altre vicine ruoteranno e trasleranno nella stessa identica misura; il fatto di considerare nel “world view” solo le minuzie vicine rende trascurabili anche le possibili deformazioni dovute all'elasticità

della pelle, poiché tutta la zona in prossimità della minuzia centrale subirà all'incirca le stesse deformazioni. Al contrario, più ci si allontana dalla minuzia centrale, più la deformazione risulterà significativa, distorcendo maggiormente la “vista” della minuzia e vanificando così l'affidabilità del match.

Una volta ottenuta questa descrizione del mondo circostante, il nostro algoritmo cerca in pratica di accoppiare le minuzie che possiedono un panorama simile: se due minuzie vedono i propri vicini nella stessa maniera, probabilmente corrisponderanno nei due template e quindi nelle due impronte.

Per realizzare l'accoppiamento utilizziamo un algoritmo di assegnamento di costo minimo, costruendo un grafo bipartito dove i nodi rappresentano le minuzie ed i costi degli archi rappresentano il valore che riassume la “world view”; i nodi a “sinistra” del grafo bipartito corrispondono alle minuzie del template di riferimento e quelli a “destra” corrispondono alle minuzie del template candidato. L'algoritmo cerca di accoppiare tutti i nodi minimizzando il costo dell'assegnamento, ottenuto sommando tutti i costi degli archi che lo compongono. La soluzione del problema di fingerprint matching usando un algoritmo di Assegnamento di costo minimo è il vero punto di innovazione in questo campo, dato che gli altri sistemi presenti in letteratura adottano procedure basate sulla trasformazione di Hough, sulla distanza tra stringhe o sulla programmazione dinamica in generale (vedi paragrafo 3.5). In pratica, come risultato finale abbiamo ottenuto un algoritmo affidabile perché robusto a trasformazioni e deformazioni ed oltre tutto veloce e facile da programmare, caratteristica necessaria la sua realizzazione su smart card.

Tuttavia, al lavoro già fatto potrebbero essere apportate alcune modifiche, al fine di migliorare l'affidabilità dell'algoritmo, per esempio cercando di sfruttare tecniche utilizzate efficacemente in altri algoritmi (vedi il paragrafo 3.5 sullo stato dell'arte) che prevedono un primo passo iniziale in cui, mediante punti dell'impronta di notevole

importanza come i punti di *core* o *delta*, vengono stimate le trasformazioni, come traslazioni e rotazioni, necessarie a far sovrapporre due immagini una sull'altra; l'algoritmo di verifica verrebbe quindi applicato a due insiemi di minuzie che sono già stati "avvicinati".

Miglioramenti potrebbero essere anche ottenuti studiando nuove feature da sostituire o aggiungere alle quattro che al momento sono usate nel calcolo: alcuni esperimenti fatti hanno mostrato che la feature "distanza tra i vicini", che assume il valore della distanza tra due vicini di due differenti minuzie riportati nello stesso sistema di coordinate, presenta un punteggio della percentuale EER più basso rispetto a quello di tutte le altre feature; il suo utilizzo insieme a quelle già presenti, inoltre, migliora l'EER globale ottenuto con la somma pesata di tutte le feature. Nell'algoritmo sono poi presenti molti parametri che potrebbero essere determinati in maniera più precisa, come il numero massimo delle minuzie presenti nel template (parametro che può anche essere trascurato quando si usa l'intero insieme di minuzie trovato da MINDTCT), il numero dei vicini, adesso fissato a cinque, e che può essere fatto variare in base alla relativa affidabilità. Un altro parametro che può essere modificato riguarda il controllo che viene effettuato sull'arco del grafo congiungente due nodi-minuzie: in questo caso si tratterebbe di scegliere se inserire o meno tale arco e quindi rendere possibile o evitare che le due minuzie vengano accoppiate durante l'assegnamento. Da un lato questo controllo può essere sfruttato per diminuire il numero di archi presenti nel grafo e conseguentemente anche il tempo medio di verifica, dall'altro possiamo cercare di non inserire archi tra due minuzie tra le quali osserviamo una profonda differenza, per esempio in termini di affidabilità rilevata da MINDTCT, oppure in termini di posizione all'interno delle due immagini acquisite; in questo modo l'erroneo accoppiamento di tali minuzie risulterebbe evitabile a priori.

Infine anche il calcolo dei pesi per le differenti feature può essere ulteriormente affinato da altri esperimenti e dall'osservazione dei relativi grafi FAR-FRR con i punti di intersezione tra le due curve (punti di EER). Nel nostro studio inoltre abbiamo deciso di configurare l'algoritmo per migliorarne le prestazioni nel punto di EER scegliendo una soglia opportuna, dato che è quello che solitamente viene fatto per le applicazioni in ambito civile. Dato che il passo finale del nostro studio consisteva nel passaggio ad una tecnologia smart card, ci sembrava naturale calibrare al meglio le prestazioni sul punto di EER.

Un'altra possibilità, invece di sommare le feature normalizzate e pesate in un unico valore, è quella di organizzare un test a votazione eseguendo l'algoritmo con ciascuna feature presa singolarmente ed attribuire un voto a ciascuno dei quattro distinti risultati; aggiungendo per esempio un'altra feature o eseguendo l'algoritmo con tutte le feature sommate, otteniamo il quinto voto per realizzare una votazione a maggioranza. Ancora, è possibile modificare la procedura eseguendo il match sulla prima feature, poi, se questo è risultato positivo, controllare la verifica sulla seconda feature e così via: il match sarà globalmente positivo se, uno dopo l'altro, lo saranno stati tutti quelli sulle singole feature; la verifica si interrompe invece al primo match negativo, naturalmente con esito globale negativo.

Per la realizzazione in ambiente Java™, per prima cosa è stato necessario modificare il codice MINDTCT scritto in linguaggio C per adattarlo alle nostre esigenze: il programma trovava infatti solo i vicini a destra (rispetto alla posizione sull'asse delle ascisse) della minuzia centrale, mentre eravamo interessati alle 5 minuzie più vicine in assoluto alla minuzia centrale. Dopo questa fase abbiamo dovuto costruire un parser di del file generato da MINDTCT ed un convertitore da quel formato al nostro, che contiene altre informazioni; infine abbiamo programmato anche uno scrittore di file di testo

contenete le informazioni del nostro formato. L'unica vera grande difficoltà affrontata durante la fase di debugging del codice è stata causata dalle approssimazioni dovute all'errore di macchina sui reali di tipo *float*, che in alcuni casi portava a trovare un falso ciclo di costo negativo nell'albero dei cammini minimi cercato dalla procedura *Shortest Path Tree* e quindi ad un loop infinito nella procedura stessa. Tenendo conto di questo possibile errore, la procedura terminava invece correttamente, nella restante parte di calcolo, con l'albero dei cammini minimi utile.

Per quanto riguarda la realizzazione finale su Java Card™ l'algoritmo ha subito alcune semplificazioni imposte dall'ambiente, sicuramente più limitato in termini di memoria disponibile e velocità del processore, che però possono essere in parte superate con il rapido miglioramento della tecnologia o comunque con la scelta di un supporto smart card con più prestazioni, fra quelli disponibili in commercio; chiaramente, tanto maggiore è la potenza di cui dispone il supporto, tanto più le prestazioni dell'algoritmo su carta si avvicinano a quelle su PC. L'algoritmo base è stato comunque implementato nella sua interezza, mentre le semplificazioni sono state rivolte alle strutture dati memorizzate sulle quali lavora l'algoritmo. Per esempio il numero delle minuzie è stato limitato ad un massimo di 25, è stato fissato un numero massimo di archi tra le minuzie nel grafo e, soprattutto, invece di mantenere un vettore di quattro posizioni per le quattro feature che descrivono come una minuzia vicina viene vista da parte di quella centrale, abbiamo scelto di avere un solo valore che riassume le quattro feature già pesate e sommate, per cercare di diminuire il più possibile l'occupazione.

Le difficoltà affrontate durante questa implementazione hanno quindi riguardato la gestione di un ambiente molto più povero rispetto al “fratello maggiore” Java, che da una parte consente di dovere conoscere un numero minore di strumenti relativamente più semplici (le API Java Card™ sono molto scarse), ma dall'altra impone una fatica e

una fantasia maggiori dal punto di vista della programmazione per poter compensare il primo fattore.

La padronanza acquisita poi sugli strumenti di un toolkit grafico e quindi di più immediato utilizzo, come quello di Cyberflex [89], ha notevolmente velocizzato la realizzazione ed i test di funzionamento sulla carta: ricordiamo che l'applicazione non è stata provata in un ambiente simulato su PC, ma su una vera e propria smart card inserita in un lettore/scrittore.

Concludendo, e facendo un veloce riassunto di tutte le tappe del nostro lavoro, riteniamo di aver raggiunto i più importanti obiettivi posti all'inizio del lavoro di tesi: in primis ci siamo rivolti ai principali aspetti della biometria in generale, approfondendo poi la tecnologia basata sulle impronte digitali con tutto quello che ne consegue, come le informazioni distintive rappresentate dalle minuzie ed i metodi di acquisizione presenti attualmente. In seguito ci siamo rivolti allo studio di alcuni dei più importanti algoritmi presenti in questo settore esaminando anche i parametri su cui si basa la valutazione delle prestazioni, e necessari quindi per valutare le implementazioni successive del nostro algoritmo. Forti di questo background acquisito abbiamo scelto di ideare un nuovo algoritmo di fingerprint matching ispirandoci alla "visione del mondo circostante" ("world view") proposta da Fionn Murtagh in [72]. L'obiettivo di realizzare una procedura di riconoscimento veloce è stato raggiunto, anche se le prestazioni di affidabilità, sicuramente migliorabili, non sono certo paragonabili con i migliori algoritmi presentati a FVC2002. Infine abbiamo dimostrato la portabilità del codice da un ambiente Java ad un ambiente Java CardTM per poter rendere possibile il match-on-card (MOC) e garantire così una maggiore sicurezza dell'ambiente di verifica biometrica, dal momento che i dati relativi all'utente che vuole riconoscersi non abbandonano mai il supporto personale e quindi non sono passibili di intercettazioni e manomissioni. Per

compiere quest'ultimo passo abbiamo implementato le Java Card™ Biometric API [95] messe a disposizione da Java Card™ Forum [88] ed utilizzato la piattaforma di sviluppo Java Card™ Cyberflex di Schlumberger; in questo caso abbiamo dimostrato la piena possibilità di implementare il nostro algoritmo in questo ambiente anche se le limitazioni in termini di affidabilità e tempi dipendono fortemente soprattutto dalla memoria RAM messa a disposizione dal supporto smart card ed in secondo luogo anche dalla memoria EEPROM necessaria alla memorizzazione dei template.

Infine, l'algoritmo presentato in questa tesi parteciperà all'edizione del 2004 della competizione per algoritmi di verifica di impronte digitali *Fingerprint Verification Competition* (FVC2004) [68] organizzata dal *Biometric Systems Lab* (Università di Bologna), dal *Pattern Recognition and Image Processing Laboratory* (Michigan State University) e dal *Biometric Test Center* (San Jose State University). Ringraziamo il Dottor Stefano Frassi dell'Istituto di Informatica e Telematica del CNR di Pisa, per il porting effettuato nel linguaggio C e per l'adattamento ai parametri imposti dall'organizzazione. La data limite per la sottomissione del codice è fissata per il 30 Novembre 2003, mentre i risultati saranno disponibili il 15 Febbraio 2004 sul sito internet [68].

Bibliografia:

- [1] D. B. Parker, "A New Framework for Information Security to Avoid Information Anarchy," presented at IFIP TC11 eleventh International Conference on Information Security, IFIP/Sec '95, London, 1995.
- [2] Clarke, R. "Human Identification in Information Systems: Management Challenges and Public Policy Issues" *Information & People*, vol.7, no.4 (December 1994) pp 6-37
- [3] Fox-Davies A.C. and Carlyon-Britton P.W.P. " A treatise on the law concerning names and changes of name", Elliot Stock, Londra 1906.
- [4] Ehrlich, T "Passports" *Stanford L. Rev.*, v.19, pp.129--149, 1966-67.
- [5] Wood, H.M. "The use of passwords for controlled access to computer resources" *National Bureau of Standards Special Publication 500-9*, US Dept. of Commerce/NBS, May 1977
- [6] Galton, G. "Personal Identification and Description " *Nature* pp. 173-177, June 21, 1988
- [7] Harmon, L.D., Khan, M.K., Lasch, R. and Ramig, P.F. "Machine Identification of human faces" *Pattern Recognit.*, vol.13, pp 97--110., 1981.
- [8] Samal, A.and Iyengar, P.A. "Automatic recognition analysis of human faces and facial expressions: A survey" *Pattern Recognit.*, vol.25, pp.65--77, 1992.

- [9] R.-L. Hsu, "Face Detection and Modeling for Recognition", Ph.D. Thesis, 2002,
ftp://ftp.cse.msu.edu/pub/prip/vincent/hsu.thesis_phd.pdf
- [10] Fabian Monrose and Aviel D. Rubin. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4):351-- 359, 2000.
- [11] 1999 Glossary of Biometric Terms, Tony Mansfield (Association for Biometrics (AfB)), Gary Roethenbaugh (International Computer Security Association (ICSA)).
- [12] Horse Identification Method Using Eye Patterns Masahiko Suzaki、 Osamu Yamakita、 Yuji Kuno、 Hiroko Aida、 Naoki Sasaki、 Ryo Kusunose, *IPSJ JOURNAL* Vol.41 No.08 – 028, 2000
- [13] Davies, D.W. and Price W.L. "Security for Computer Networks" John Wiley & Sons, 1984.
- [14] George M.H., King R.A. " Robust speaker verification biometric" *IEEE Annual International Carnahan Conference on Security Technology, Proceedings of the 29th Annual 1995 International Carnahan Conference on Security Technology*, pp. 41-46.
- [15] Net Nanny web site: www.netnanny.com/
- [16] Stephen Northcutt and Judy Novak, *Network Intrusion Detection: An Analyst's Handbook*, Second Edition, New Riders, September 2000.
- [17] Elizabeth D. Zwicky, Simon Cooper, and D. Brent Chapman, *Building Internet Firewalls*, 2nd Edition, O'Reilly & Associates, Inc, June 2000.
- [18] Kenneth R. van Wyk and Richard Forno, *Incident Response*, O'Reilly & Associates, Inc., August 2001.
- [19] Biometric Technology Today <http://www.biometrics-today.com/>
- [20] Newman, E. "The Biometric Report" SJB Services, UK, 1995
- [21] Autori vari, *Manuale di Anatomia Comparata*, Delfino Editore, 2000.

- [22] J.G. Daugman and G.O. Williams, A Proposed Standard For Biometric Decidability, Proc. CardTech/SecureTech Conference, Atlanta, GA, pp. 223-234, 1996
- [23] Kevin Mitnick's Christmas Day crack of Tsutomu Shimomura's machine
http://www.totse.com/en/hack/hack_attack/hacker03.html
- [24] Zunkel, R. "Biometrics and Border Control" Security Technology & Design, May 1996, p.22-27
- [25] TASS project: http://secure.cihi.ca/cihiweb/en/downloads/partner_docs_e_clientid.pdf
- [26] Centrobiometrika srl web site: <http://www.centrobiometrika.it/>
- [27] Campbell, J., Alyea, L., Dunn J. "Biometric Security: Government Applications and Operations": <http://www.biometrics.org/REPORTS/CTSTG96/>
- [28] User's Guide to NIST Fingerprint Image Software (NFIS) NISTIR 6813.
- [29] McCabe, "Data Format for the Interchange of Fingerprint, Facial, Scar Mark & Tattoo (SMT) Information," American National Standard ANSI/NIST-ITL 1-2000, July 2000.
Available from R.M. McCabe at NIST, 100 Bureau Drive, Stop 8940, Gaithersburg, MD 20899-8940.
- [30] Giorgio Gallo, Stefano Pallottino, Maria Grazia Scutellà, Antonio Frangioni e Giancarlo Bigi (autori principali), *Appunti di Ricerca Operativa*, Anno Accademico 2002/2003.
Documento scaricabile dal sito web del gruppo di ricerca operativa (ORG), Dipartimento di Informatica dell'Università di Pisa. <http://www.di.unipi.it/di/groups/optimize/ORGGroup.html>
- [31] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, "Network flows. Theory, algorithms, and applications", Prentice Hall, Englewood Cliffs, NJ (1993).
- [32] M. S. Bazaraa, J. J. Jarvis, H. D. Sherali, "Linear programming and network flows", Wiley, New York, NY (1990).
- [33] The MacTutor History of Mathematics archive, University of St Andrews Scotland
<http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Galton.html>

- [34] H. C. Lee and R. E. Gaensslen (editors), *Advances in Fingerprint Technology*, Elsevier, New York, 1991.
- [35] A. K. Jain, L. Hong, S. Pankanti, and Ruud Bolle, "An Identity Authentication System Using Fingerprints," *Proceedings of the IEEE*, Vol. 85, No. 9, pp. 1365-1388, 1997.
- [36] Federal Bureau of Investigation. *The Science of Fingerprints: Classification and Uses*, U.S. Government Printing Office, Washington D.C., 1984.
- [37] Javacard.org web site: <http://www.javacard.org>
- [38] FBI *Advanced Latent Fingerprint School*. U.S. Department of Justice, Federal Bureau of Investigation, 1983.
- [39] Federal Bureau of Investigation. <http://www.fbi.gov>
- [40] Scientific Working Group on Friction Ridge Analysis, Study and Technology
<http://www.swgfast.org/>.
- [41] A. K. Jain, S. Prabhakar, L. Hong, and S. Pankanti, "Filterbank-based Fingerprint Matching," *IEEE Transactions on Image Processing*, Vol. 9, No.5, pp. 846-859, May 2000,
<http://biometrics.cse.msu.edu/publications.html>..
- [42] Unpublished 1995 report by Frank Torpay of Mitre Corporation using data extracted from the FBI's Identification Division Automated Services database of 22; 000; 000 human-classified fingerprints.
- [43] Unofficial NSW police web site for the state of New South Wales, Australia.
<http://www.policensw.com/index.html>
- [44] NIST web site: <http://www.nist.gov> and
<http://www.itl.nist.gov/iad/894.03/databases/defs/#finglist>
- [45] User's guide to NIST FingerPrint Image Software (NFIS), NISTIR 6813, Michael D. Garris, Craig I. Watson, R. Michael McCabe, Charles L. Wilson, National Institute of Standards and Technology.

- [46] R. Bahuguna, Fingerprint Verification Using Hologram Matched Filterings, Proc. Biometric Consortium Eight Meeting, San Jose, California, June 1996.
- [47] H.C. Lee and R.E. Gaensslen, editors, Advances in Fingerprint Technology, Elsevier, New York 1991
- [48] Grew Nehemiah: <http://es.rice.edu/ES/humsoc/Galileo/Catalog/Files/grew.html>
- [49] Mastercard Smart Card Technologies web site:
http://www.mastercardintl.com/newtechnology/smartcards/onesmart_home.html
- [50] The FBI Fingerprint Image Compression Standard:
<http://www.c3.lanl.gov/~brislawn/FBI/FBI.html>
- [51] Anil K Jain, Salil Prabhakar, Arun Ross, "Fingerprint matching: data acquisition and performance evaluation", Department of Computer Science and Engineering, Michigan State University East Lansing.
- [52] A.K. Jain, L. Hong, and R. Bolle, On-line Fingerprint Verification, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19 num. 4 (1997) 302-314.
- [53] H. Cummins and C. Midlo, Finger Prints, Palms and Soles, Dover Publications Inc., New York 1961
- [54] S. Pankanti, S. Prabhakar and A.K. Jain, "On the Individuality of Fingerprints", Proc. IEEE CVPR, pp. 805-812, Hawaii, Dec 2001
- [55] E. P. Richards, "Phenotype vs. Genotype: Why Identical Twins Have Different Fingerprints?", http://www.forensic-evidence.com/site/ID/ID_Twins.html
- [56] Fingerprint Evidence In The U.K
http://www.forensicevidence.com/site/ID/Buckley_UK.html
- [57] Andre Moenssens, The Reliability of Fingerprint Identification.
<http://www.forensic-evidence.com/site/ID/pollak2002.html>
- [58] Daubert v. Merrell Dow Pharmaceuticals, 113 S. Ct. 2786 (1993).

- [59] U.S. v. Byron Mitchell, Criminal Action No. 96-407, U.S. District Court for the Eastern District of Pennsylvania.
- [60] Legal challenges to fingerprint
http://www.onin.com/fp/daubert_links.html#whatisadauberthearing
- [61] Atmel Products http://www.tcs.thomson-csf.com/fingerchip/FC_home.htm
- [62] Ultra-Scan <http://www.ultra-scan.com/>
- [63] Veridicom Products <http://www.veridicom.com>
- [64] N.Ratha, K.Karu, S.Chen and A.K. Jain, A Real-time Matching System for large Fingerprint Database, *IEEE Trans. On PAMI* , Vol .18, No 8, pp 799-813, 1996.
- [65] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, “FVC2000: Fingerprint Verification Competition”, Proc. 15th International Conference Pattern Recognition, Barcelona, September 3-8, 2000,
- [66] FVC2000 (Fingerprint Verification Competition) Web Site:
<http://bias.csr.unibo.it/fvc2000/>
- [67] FVC2002 (Fingerprint Verification Competition) Web Site:
<http://bias.csr.unibo.it/fvc2002/>
- [68] FVC2004 (Fingerprint Verification Competition) Web Site:
<http://bias.csr.unibo.it/fvc2004/>
- [69] Wee Peng Tay Boon Thye Yeo Ying Yu Tai, EE368A Project: Fingerprint Verification, Image Systems Engineering Program, University of Stanford,
http://ise0.stanford.edu/class/ee368a_proj01/dropbox/project22/finger/
- [70] Point Pattern Matching, A tutorial by Vincent Paquin,
<http://www3.sympatico.ca/vpaquin/tutorial/>

- [71] Steven Gold, Anand Rangarajan, Chien-Ping Lu, Suguna Pappu and Eric Mjolsness, New Algorithms for 2D and 3D Point Matching: Pose Estimation and Correspondence , in press, Pattern Recognition, 1997.
- [72] Murtagh, F., "A new approach to point pattern matching", Publications of the Astronomical Society of the Pacific, 104, 301-307, 1992.
- [73] Dinesh P. Mital, Eam Khwang Teoh, "An Automated Matching Technique for Fingerprint Identification," 1997 Fist International Conference on Knowledge-based Intelligence Electronic System, 21-23 May 1997.
- [74] Zsolt Miklos Kovacs-Vajna, "A Fingerprint Verification System Based on Triangular Matching and Dynamic Time Warping," IEEE Transaction on Pattern Analysis and Machine Intelligence, VOL. 22, No. 11, pp 1266 - 1276, November 2000.
- [75] Gian Luca Marcialis, Fabio Roli, Paolo Loddo, "Fusion of Multiple Matchers for Fingerprint Verification", Department of Eletrical and Electronic Engineering, University of Cagliari, Italy.
- [76] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas, On Combining Classifiers, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20 num. 3 (1998) 226-239.
- [77] S.Chen, A.K.Jain, *A fingerprint matching algorithm using dynamic programming*, Technical Report, Department of Computer Science and Engineering, Michigan State University.
- [78] Biometric Identification: <http://www.biometricid.com/>
- [79] Cogent: <http://www.cogentsystems.com/>
- [80] East Shore Technology: <http://www.east-shore.com/>
- [81] FingerSec: <http://www.fingersec.com/>
- [82] Mentalix: <http://www.mentalix.com/>
- [83] NEC: <http://www.nectech.com/afis/>

- [84] NeuroDynamics: <http://www.neurodynamics.com/>
- [85] Optel: <http://www.optel.com.pl/>
- [86] Sagem Morpho: <http://www.morpho.com/>
- [87] Biometrics at Michigan State University, Abstracts of Current Projects:
<http://biometrics.cse.msu.edu/abstracts.html>
- [88] Java Card™ Forum web site: <http://www.javacardforum.org/>
- [89] Cyberflex Schlumberger web site <http://www.cyberflex.com/>
- [90] Gemplus web site <http://www.gemplus.com/>
- [91] Globalplatform web site <http://www.globalplatform.org/>
- [92] BioAPI Consortium web site <http://www.bioapi.org/>
- [93] Common Biometric Exchange File Format (*CBEFF*) <http://www.nist.gov/cbeff>
- [94] NIST Biometric Consortium Working Group, <http://www.nist.gov/bcwg>
- [95] Java Card™ Biometric API White Paper (Working Document) Prepared by: Biometric Consortium Interoperability, Assurance, and Performance Working Group, download from
http://www.javacardforum.org/Documents/Biometry/JCBiometricsAPI_WhitePaper.pdf
- [96] Java Card™ Forum Biometric API specification.
<http://www.javacardforum.org/Documents/JCFBioAPIV1A.pdf>
- [97] International Biometric Group web site: <http://www.biometricgroup.com/>
- [98] Java Card™ Forum Biometric API export file and stub files
<http://www.javacardforum.org/Documents/Biometry/ExportAndStubsJCBiometricAPI.zip>
- [99] Colin Soutar, Biometric System Performance and Security (2002)
http://www.bioscrypt.com/assets/bio_paper.pdf
- [100] International Organization for Standardization (ISO) web site: <http://www.iso.ch>
- [101] ISO/IEC JTC1/SC17/WG8 Contactless integrated circuit(s) cards <http://wg8.de/sd1.html>

- [102] ISO/IEC JTC1/SC17 Cards and Personal Identification. Home page
<http://www.sc17.com/>
- [103] alt.technology.smartcards FAQ web site: <http://www.scdk.com/atsfaq.htm>
- [104] STMicroelectronics home page <http://us.st.com/stonline/welcome.htm>
- [105] Cyberflex® Access™ Cards Programmer's Guide, Document Edition C300473_rev1,
June 2002, download version C300546_rev1 at
http://www.cyberflex.com/Support/CyberflexPG_4-1_rev1.pdf
- [106] Java Card™ Technology home page, <http://java.sun.com/products/javacard/>
- [107] Java Card™ Application Programming Notes, Sun MicroSystem Inc., June 2002,
download at <http://java.sun.com/products/javacard/downloads/>
- [108] Java Card™ 2.1 Platform API Specification Final Revision 1.0
<http://java.sun.com/products/javacard/htmldoc/>
- [109] MULTOS web site, <http://www.multos.com/>
- [110] Java Developer web site <http://java.sun.com/>
- [111] Java Card™ 2.2 Development Kit User's Guide, Sun MicroSystem Inc., June 2002,
download at <http://java.sun.com/products/javacard/specs.html>
- [112] Gemplus web site <http://www.gemplus.com/>
- [113] Giesecke & Devrient web site <http://www.gdm.de/>
- [114] Oberthur Card Systems web site <http://www.oberthurcs.com/>
- [115] Orga web site <http://www.orga.com/>
- [116] Schlumberger web site <http://www.slb.com/>
- [117] Sony FIU-710 "Puppy" <http://www.securecomputing.com/>
- [118] How to write a Java Card™ applet: A developer's guide (javaworld site).
<http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard.html>

- [119] U.Chirico, *“Riconoscimento biometrico e impronta digital”*, Computer Programmino n.123, ISSN 1123-85-26, Gruppo Editoriale Infomedia.
- [120] BioAPI Specification, Version 1.01, <http://www.bioapi.org/BIOAPI1.1.pdf>
- [121] Smart card FAQ at Mobile-Mind: <http://www.mobile-mind.com/htm/scstd.htm>
- [122] An Introduction to Java Card™ Technology - Part 2, The Java Card™ Applet by C. Enrique Ortiz September 2003, <http://wireless.java.sun.com/javacard/articles/javacard2/>
- [123] Java Card™ 2.2 Development Kit: http://java.sun.com/products/javacard/dev_kit.html
- [124] An Introduction to Java Card™ Technology - Part 1 by C. Enrique Ortiz May 29, 2003 <http://wireless.java.sun.com/javacard/articles/javacard1/>
- [125] Release Notes Java Card™ Specifications, Version 2.2.1, Public Review Draft ("Beta Release") September, 2003 http://java.sun.com/products/javacard/RELEASENOTES_public_online.html#documentation_roadmap
- [126] NIST Fingerprint Image Software (NFIS): http://www.itl.nist.gov/iad/vip/databases/defs/nist_nfis.html
- [127] Universal Latent Workstation: <http://www.fbi.gov/hq/cjisd/iafis.htm>
- [128] ANSI/NIST-ITL 1-2000 “Data Format for the Interchange of Fingerprint, Facial, Scar Mark & Tattoo (SMT) Information”, download PDF document at <http://www.itl.nist.gov/iad/894.03/fing/fing.html>
- [129] S.G. Davies, Touching Big Brother: How Biometric Technology Will Fuse Flesh and Machine, Information Technology & People, Vol. 7, No. 4, pp.60-69, 1994.
- [130] Cyberflex Access Developer 32K card http://www.cyberflex.com/Products/Cyberflex_Access/cyberflex_access.html
- [131] NIST Special Databases and Software from the Image Group. Web site: http://www.itl.nist.gov/iaui/894.03/databases/defs/vip_dbases.html

- [132] NIST Special Database 9: <http://www.nist.gov/srd/nistsd9.htm>
- [133] NIST Special Database 9 sample, file prints9.tar.z:
<ftp://sequoyah.ncsl.nist.gov/pub/databases/data>
- [134] SFINGE (Synthetic FINGERprint GEnerator):
<http://bias.csr.unibo.it/research/biolab/Sfinge.html>
- [135] S. Prabhakar, *Fingerprint classification and matching using a filterbank*, PhD 2001.
- [136] Biometric Systems Team, laboratorio di biometria, dipartimento di scienze dell'informazione dell'università di Bologna (Cesena),
<http://bias.csr.unibo.it/research/biolab/team.html>
- [137] Javadoc Tool Home Page: <http://java.sun.com/j2se/javadoc/>
- [138] Netbeans web site: <http://www.netbeans.org/>
- [139] S. Prabhakar and A.K. Jain, Decision-level Fusion in Fingerprint Verification, *Pattern Recognition*, vol. 35 num. 4 (2002) 861-874.
- [140] Towitoko home page: <http://www.towitoko.com>
- [141] Simson Garfinkel, *Database Nation: The Death of Privacy in the 21st Century*, O'Reilly and Associates, 2001.

Appendice A: biometria e applicazioni.

A.1 Introduzione.

Uno delle più importanti minacce alla sicurezza intesa come concetto generale, è rappresentata dalla “impersonificazione”. Con la impersonificazione qualcuno cerca di assumere un’identità differente; l’identificazione e l’autenticazione servono per ridurre le possibilità di questo pericolo.

In [1], “l’identificazione” viene definita come “l’associazione di dati con una particolare entità rappresentata, a cui sono associati uno o più identificatori che la distinguono, come per esempio esseri umani o entità fisico-logiche”.

Il motivo originale che ha portato al bisogno di un sistema di identificazione è stato sicuramente sociale ed in seguito, quando le transazioni cominciavano a farsi sempre più complicate, si è tramutato anche in economico.

I nomi sono stati i primi mezzi per l’identificazione, ed in particolare i cognomi, che furono usati in Britannia dal 1066 [2, 3]. Nel 1538 sotto il regno di Enrico VIII, i sacerdoti scrivevano nei registri le nascite, le morti e i matrimoni per scopi identificativi. Anche i passaporti comparvero nella legislazione inglese già dal 1300 [4].

Esiste una ampia varietà di mezzi per identificare l'identità di una persona:

- ◇ L'aspetto (come appare una persona, per esempio dall'altezza, il sesso od il peso);
- ◇ Il comportamento sociale (come una persona interagisce con le altre);
- ◇ Il nome (come è chiamata una persona);
- ◇ La conoscenza (cosa conosce);
- ◇ Il codice (come viene chiamata da una organizzazione);
- ◇ Il possesso (qualcosa che la persona possiede);
- ◇ La fisiologia naturale (per esempio i tratti facciali);
- ◇ Le caratteristiche imposte da altri (per esempio collari o braccialetti);
- ◇ Le caratteristiche bio-dinamiche (come agisce una persona);

Il principale fine dell'autenticazione o verifica, è invece quello di proteggere un sistema dall'uso non autorizzato.

L'autorizzazione ad entrare in un edificio, ad aprire un armadietto, a varcare un confine, a prelevare denaro da una banca etc. è sempre collegata all'identità di una persona. È perciò necessario dimostrare tale identità in un modo o nell'altro. Definiamo questa procedura *verifica*. Una persona sostiene di avere una determinata identità e ciò deve essere verificato; è un problema noto alla polizia, per esempio con persone che presentano una carta d'identità sospetta. Tuttavia, la polizia si trova spesso di fronte ad un altro tipo di problema: chi è la persona che ha lasciato una determinata traccia, per esempio un'impronta digitale. In questo caso si ricerca l'identità di una persona

sconosciuta, facciamo quindi una *identificazione*. Le procedure di identificazione sono basate sui seguenti approcci [5]:

- ◇ *Verifica basata sulla conoscenza*: ciò che si sa. L'entità verificatrice conosce informazioni riguardanti l'identità reclamata che possono essere conosciute o prodotte (per esempio password, personal identification number (PIN), questionari).
- ◇ *Verifica basata sul possesso*: ciò che si ha. Il richiedente verrà riconosciuto attraverso la dimostrazione del possesso di un preciso oggetto (per esempio una carta a banda magnetica, una smart card, una carta ottica).
- ◇ *Verifica basata su caratteristiche personali*: ciò che si è. Al richiedente vengono misurate alcune caratteristiche personali scelte. Queste sono definite caratteristiche biometriche e la tecnologia alla base di tale identificazione è chiamata Biometria: in altre parole, il riconoscimento biometrico consiste nell'identificazione automatica dell'individuo sulla base di caratteristiche fisiologiche considerate uniche.

Gli specialisti della Biometria utilizzano l'espressione uno a uno quando si tratta di verifica, oppure uno a molti quando si tratta di identificazione. Nella trattazione seguente si parlerà essenzialmente di verifica che individua il caso più comunemente utilizzato in ambienti non legati all'applicazione della legge.

Nel corso del testo i termini “verifica” o “identificazione” saranno usati alternativamente per individuare tutte e due queste metodologie nel loro insieme, se non specificato diversamente.

Nulla vieta che le tre tipologie di identificazione presentate precedentemente possano essere usate contemporaneamente all'interno dello stesso sistema, aumentandone in tal modo la sicurezza (vedi figura A.1).

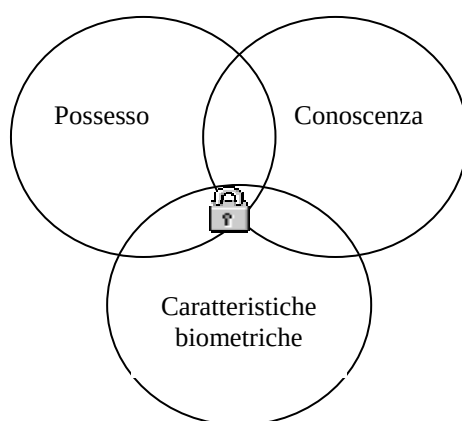


Figura A.1
Come aumentare la sicurezza di un sistema di identificazione usando tutte e tre le tipologie di autenticazione.

Probabilmente la più antica attestazione di identità e di autorizzazione basata su mezzi tecnici, e non sul riconoscimento personale, è la chiave meccanica. In questo caso l'attestazione di identità si basa sul possesso. Tutte le card in plastica leggibili (con sistemi di memorizzazione dati magnetici, elettrici o ottici) sono esempi della stessa categoria. Tali metodi di attestazione di autorizzazione hanno raggiunto un livello tecnico elevato ed alcuni di loro sono molto difficili da copiare o da falsificare. Presentano tuttavia uno svantaggio intrinseco: il sistema tecnico è in grado di verificare l'identità e quindi l'autorizzazione della carta o della chiave ma non l'identità del

portatore. In altre parole: la card o la chiave in proprio possesso possono essere rubate, perdute o cedute a persone non autorizzate.

Sistemi basati sulla *conoscenza* anziché sul possesso hanno il fine di evitare questo problema. La password rappresenta la prima forma di questo tipo di identificazione. Di recente tali metodi sono stati automatizzati mediante l'utilizzo di password di accesso a computer o di codici identificativi (ID). Non è quindi possibile un uso improprio in seguito a furto ma è pur sempre possibile che qualcuno acquisisca in qualche modo tali informazioni e ne faccia poi un cattivo uso. Malgrado tutte le avvertenze, un certo numero di utenti annota, per fare un esempio, il codice ID delle proprie carte di credito, e questo ne riduce il valore di sicurezza a zero.

La combinazione di possesso e sistemi di conoscenza riduce ulteriormente la possibilità di un utilizzo improprio, ma non elimina il problema principale ovvero che il portatore non sia inconfutabilmente identificato.

L'unico mezzo per poter identificare una persona inconfutabilmente è quello di riconoscerne automaticamente le caratteristiche personali attraverso la biometria.

La biometria ha radici che affondano molto lontano nel tempo: Alphonse Bertillon (Francia 1870) inventò un sistema, chiamato appunto sistema Bertillon, basato sull'analisi dell'impronta digitale per identificare i criminali. Francis Galton [6], sempre nella seconda metà del 1800, cercò di migliorare il sistema Bertillon introducendo anche sistemi di misurazioni del profilo facciale e creò addirittura un laboratorio chiamato *Anthropometric Laboratory*, per misurare e collezionare dati antropometrici (vedi anche figura A.2 [33]): i soggetti erano felici di pagare l'esperimento, in cambio di una copia delle misurazioni effettuate e per questo riuscì a collezionare una grande quantità di informazioni, impossibile da analizzare propriamente dato l'allora livello tecnologico.

Da allora sono stati inventati molti altri sistemi di misurazioni dei tratti del volto [7], [8], [9].

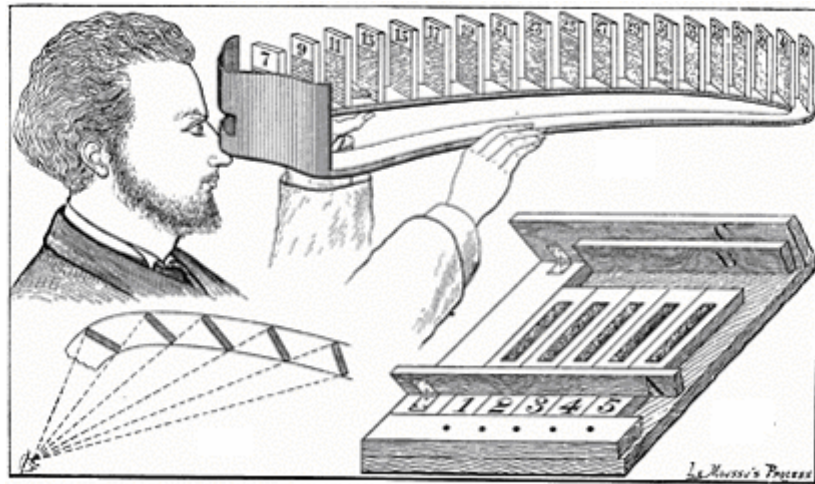


Figura A.2
Alcuni strumenti inventati da Galton per misurare la vista.

Altre tecniche più sofisticate sono state usate negli ultimi anni: schema del patrimonio genetico (DNA), analisi della forma dell'orecchio, analisi dell'odore, analisi dei pori sudoripari, analisi della forma della testa o addirittura dalla frequenza di battitura dei tasti su una tastiera ("keystroke analysis" [10]), utile per l'accesso da una console. Le tecniche biometriche possono essere raggruppate in due classi:

- ◇ *Tecniche basate su aspetti fisiologici:* questa classe raggruppa la verifica della firma, l'analisi della retina, l'analisi facciale, la verifica dell'impronta digitale, la geometria della mano, la verifica dell'impronta vocale, l'analisi facciale, l'analisi della retina, del DNA e in generale si basa sulle misure di caratteristiche fisiologiche di una persona.

- ◇ *Tecniche basate su aspetti caratteriali*: include la verifica della firma, il ritmo di battitura su tastiera, l'analisi dell'impronta vocale e quindi misura in generale aspetti comportamentali. Queste tecniche sono chiamate anche *dinamiche*, dato che il soggetto deve compiere una qualche azione di cui vengono misurati alcuni aspetti, in contrapposizione ai metodi della prima classe, che vengono chiamati invece *statici*. L'analisi vocale appartiene in realtà a tutte e due le classi biometriche contemporaneamente [11].

I sistemi di riconoscimento biometrico presentati precedentemente possono funzionare in due modi: per quanto riguarda l'identificazione, il sistema identifica una persona ricercando in un database dei dati registrati all'atto dell'*iscrizione* e confrontandoli con quelli sottoposti; per riguarda l'autenticazione, vale solo la seconda parte e cioè il sistema cerca di stabilire quanto sono "lontani" i dati registrati all'atto dell'*iscrizione*, con quelli sottoposti durante la verifica, per stabilire se l'identità reclamata corrisponde nei dati biometrici.

Per quanto riguarda gli aspetti salienti al fine della misurazione delle prestazioni di questi sistemi, cinque valori sono importanti nella valutazione dei sistemi biometrici:

- ◇ tempo necessario per la registrazione.
- ◇ tempo necessario per la verifica.
- ◇ Falsa Accettazione (False Acceptance Rate, *FAR*, valore di falsa accettazione, errore di tipo I), la verifica di una persona che non è registrata.

- ◇ Falso Rifiuto (False, Reject Rate, *FRR*, valore di falso rifiuto, o errore di tipo I), non verifica di una persona registrata.
- ◇ Valore di Errore Uguale (Equal Error Rate, *EER*, il punto in cui il valore di Falsa Accettazione e Falsi Rifiuti si equivalgono).

Benché questi valori siano considerevolmente migliorati nel corso dell'evoluzione di tutti i sistemi conosciuti, esistono ancora differenze importanti da un sistema all'altro, in parte dovute al metodo di identificazione prescelto. Fino ad ora non esistono standard per i test. Molto è stato fatto per arrivare alla standardizzazione ma sono state riscontrate varie difficoltà per via della differente natura dei sistemi.

Il valore più difficile da giudicare è il falso rifiuto: i falsi rifiuti infatti dipendono in gran parte dal comportamento dell'utente, per esempio da come pone il dito sullo scanner, e per questo motivo una standardizzazione sarebbe particolarmente utile.

La nascita ed il primo sviluppo della biometria si deve sicuramente a necessità nate in ambito forense e quindi ad applicazioni in grado di identificare criminali o, in un secondo momento, aumentare la sicurezza delle prigioni, diminuendo i tempi di attesa rispetto, per esempio, ad una ricerca manuale tra tutte le impronte digitali contenute in archivio. La tecnologia biometrica sta però evolvendo in ambito civile, anche se ha tuttora ancora un grosso potenziale da poter essere espresso: in qualsiasi circostanza nella quale si deve avere la certezza di avere davanti una ben precisa persona, oppure di consentire l'accesso a informazioni o siti ad accesso ristretto, la biometria può venire in aiuto per garantire la sicurezza richiesta.

I principali esempi possono essere costituiti da documenti di identità elettronici, tessere elettorali, accesso ad informazioni bancarie e possibilità di operazioni sul proprio

conto corrente, commercio elettronico e accesso consentito solo a personale autorizzato in aeroporti, dipartimenti universitari, laboratori medici o centrali nucleari.

L'interesse per la tecnologia biometrica si è spostata addirittura anche sul mondo animale: un sistema di riconoscimento dei cavalli da corsa basato sull'analisi dell'iride è allo studio in Giappone [12] e già delle compagnie hanno utilizzato rilevamenti biometrici per assicurarsi del pedigree del cane di razza.

A.2 Verifica basata sulla conoscenza.

L'approccio decisamente più comune per l'autenticazione di un utente è la verifica attraverso la conoscenza di una password. Si tratta di una tecnologia semplice e di facile implementazione. L'uso delle *password* ha ampia diffusione in ambito militare, bancario, delle telecomunicazioni ed anche in protocolli per l'accesso a sistemi informativi. D'altro canto, ci sono molte ragioni per le quali questa soluzione si dimostra essere insicura: gli utenti solitamente scelgono password predicibili, come nomi propri, date di nascita o comunque stringhe di caratteri rappresentanti parole della propria lingua che possono essere facilmente scoperte attraverso attacchi a "forza bruta" utilizzando come base un vocabolario; il fatto che le password vengano spesso scritte su un promemoria e che vengano modificate dall'utente molto raramente, aumenta notevolmente la debolezza del sistema. Esistono quattro tipi di password [13]:

- ◇ *Password di gruppo*: sono conosciute da più utenti all'interno di un sistema e quindi rappresentano una minaccia alla sicurezza più elevata

- ◇ *Password “uniche”*: le classiche password che vengono distribuite a ciascun utente
- ◇ *Password “non uniche”*: sono usate per confermare un'identità reclamata: una corta password viene distribuita agli utenti dove l'identificazione dipende da un numero più lungo conservato su una carta (per esempio il PIN di una carta magnetica)
- ◇ *Password che cambiano ad ogni accesso*: i problemi correlati con questa tipologia, sono rappresentati dalla gestione della lista di tutte le password che deve essere dunque aggiornata frequentemente e la distribuzione delle medesime all'utente attraverso un canale sicuro

Altro problema legato alle password usate ed in generale alla “conoscenza”, è che possono essere facilmente dimenticate se non adoperate per un lungo periodo, comportando l'avvio di procedure amministrative per la richiesta della vecchia password (o di una nuova) a chi gestisce il sistema.

A.3 Verifica basata sul possesso.

La verifica basata sul possesso si può riassumere facilmente nell'uso di vari tipi di “carte” che possono venir scelte in base all'applicazione nella quale devono servire. Alcuni esempi sono:

- 1) Le carte con banda magnetica.
- 2) Le Radio Frequency Identification Card (o più brevemente, RF-ID).

- 3) Le carte a memoria ottica.
- 4) Le Smart Card.

Le prime sono le più largamente accettate e diffuse dato che vengono usate da molto tempo in molte applicazioni riguardanti per esempio il controllo degli accessi, per prelevare o compiere altre operazioni come la ricarica del credito sul cellulare presso gli *Automatic Teller Machine* o ATM (ovvero i bancomat) o per acquistare prodotti in vari punti vendita sparsi sul territorio. Queste carte sono anche conformi agli standard ISO (*International Organization Standard*) [14] : per esempio *ISO 7811* è lo standard per la memorizzazione su banda magnetica delle informazioni per le carte di credito.

La carta magnetica è usata solitamente in combinazione con un *Personal Identification Number* (PIN), utilizzando così due tecniche contemporaneamente per rafforzare la sicurezza: i pericoli possono derivare dalla possibilità di clonare le carte e di sottrarre il PIN.

Le carte RF-ID contengono invece una piccola trasmittente radio attivata con la ricezione di una particolare frequenza. La verifica può avvenire quindi in modo completamente trasparente rispetto all'utente, che non ha bisogno nemmeno di estrarre la carta dal portafoglio.

Le *Optical Memory Card* sono invece delle carte delle dimensioni di una normale carta di credito, capaci di memorizzare una quantità relativamente grande di dati. I dati sono copiati attraverso un processo di scrittura tramite raggio laser, simile quindi alla normale masterizzazione di un CD, e non possono essere così né modificati, né cancellati. In commercio si possono attualmente trovare modelli con capacità di più di 4Mbyte, oppure di circa 1Mbyte più un chip tipo smart card (*ISO 11693* e *ISO 11694*).

Le smart card ospitano al loro interno un chip che può contenere solo memoria o anche un semplice ma abbastanza completo microprocessore ed un proprio sistema operativo. Anche se non capace di prestazioni paragonabili ai moderni personal computer è possibile usarla come piattaforma per algoritmi abbastanza complessi.

Naturalmente, in tutte le carte presentate in questo paragrafo possono essere salvati dei dati biometrici e quindi possono ricadere nella verifica tramite biometria, utilizzando un lettore per il tipo di carta usata ed un sensore per la misurazione del dato biometrico richiesto per controllare se la persona esaminata è il reale possessore della carta.

A.4 Verifica basata su dati biometrici.

Come già accennato precedentemente, le caratteristiche misurabili di un individuo sono numerose e di conseguenza anche i sistemi di riconoscimento biometrici possono distinguersi in diverse tipologie che sostanzialmente possono essere elencate in analisi dell'iride, dei tratti del volto, della voce, della forma dell'orecchio, dello schema del DNA, nella comparazione di impronte digitali, della forma di tutta la mano, dei pori del sudore, della propria impronta olfattiva, nello schema delle vene della mano o del braccio, della battitura su tasti (*keystroke analysis*) oppure della semplice firma personale comunemente usata per stipulare contratti e scrivere assegni. Nel secondo capitolo ci occuperemo più dettagliatamente di ciò che riguarda le impronte digitali dato che questa tesi tratta di un algoritmo basato su questa tecnologia.

Per quanto riguarda i criteri per l'affidabilità e l'accettabilità degli apparati biometrici, essi possono essere suddivisi in operazionali, tecnici e finanziari. Nel seguito forniremo una breve descrizione per ciascuno di questi punti.

I criteri tecnici affrontano tutti gli ostacoli che si presentano durante la realizzazione del sistema vero e proprio, come per esempio la capienza della memoria del database dei template, la tolleranza a cambiamenti ambientali (luce, rumore, temperatura,) e naturalmente i problemi riguardanti l'affinamento delle prestazioni (affrontate nel paragrafo A.5).

Nei criteri operazionali rientra per esempio la convenienza dell'uso, la dove è richiesto per esempio un basso tempo di registrazione o verifica, come per esempio nel controllo degli accessi a punti di ingresso di edifici o zone molto frequentate (ad esempio un supermercato). Un altro punto molto importante riguarda l'*accettabilità* (*acceptability*) pubblica della tecnica, che risulta posizionarsi tra i primi posti nelle necessità di banche o comunque attività commerciali: i clienti si possono infatti rivolgere alla concorrenza nel caso venga adottata una tecnica troppo invasiva; citando un caso reale, ai cittadini Giapponesi non piace porre i palmi delle loro mani dove altri l'hanno già fatto [20]. È quindi chiaro che questo fattore dipenda fortemente dagli usi e dai costumi locali, dalle credenze religiose ed ingenerale dalla cultura della popolazione.

Inoltre anche fattori come l'*unicità* (*uniqueness*) e l'*universalità* (*universality*) devono essere opportunatamente valutati per poter scegliere la tecnica più efficace per il caso: la misurazione biometrica deve infatti riguardare una caratteristica distintiva dell'individuo e che al tempo stesso sia posseduta dalla totalità delle persone a meno di casi eccezionali (gravi incidenti, menomazioni, ecc...). Per la *permanenza* (*permanence*) la stessa caratteristica deve essere il più possibile invariante nel tempo e naturalmente *facilmente acquisibile* (*collectability*), mentre la *circonvenzione* (*circumvention*) tratta

quanto sia facile ingannare il sistema mediante tecniche fraudolente. Infine *prestazioni* (*performance*) si riferisce all'accuratezza raggiungibile nella verifica e alle risorse necessarie per ottenerla.

Nella tabella A.1 vengono comparate alcune tecnologie biometriche in base alle proprietà mostrate precedentemente [35].

Per ultimo, ma non in ordine di importanza, anche il denaro speso riveste una notevole importanza, dato che devono essere presi in considerazione non solo i costi delle apparecchiature di scannerizzazione e memorizzazione dei template, ma anche quelli della loro installazione e manutenzione, compreso il software di riconoscimento e l'addestramento del personale adibito al loro uso; nel totale possono rientrare anche il supporto fornito dagli amministratori e la protezione del sistema nei casi nei quali l'ambiente operativo deva essere controllato nelle sue condizioni di funzionamento (per esempio una certa temperatura da mantenere).

Biometrics	Universality	Uniqueness	Permanence	Collectability	Performance	Acceptability	Circumvention
Face	high	low	medium	high	low	high	low
Fingerprint	medium	high	high	medium	high	medium	high
Hand Geometry	medium	medium	medium	high	medium	medium	medium
Hand Vein	medium	medium	medium	medium	medium	medium	high
Iris	high	high	high	medium	high	low	high
Retinal Scan	high	high	medium	low	high	low	high
Signature	low	low	low	high	low	high	low
Voice Print	medium	low	low	medium	low	high	low
F.Thermograms	high	high	low	high	medium	high	high

Tabella A.1
Comparazione tra le varie tecnologie biometriche.

A.5 Pericoli per la sicurezza.

In generale queste minacce possono essere viste come potenziali violazioni alla sicurezza con risultati anche imprevisti e possibili grazie alle vulnerabilità del sistema; se per esempio un utente non autorizzato “invade” il sistema, può danneggiare informazioni, sistema operativo o programmi provocando interruzioni del servizio più o meno prolungate e quindi causare mancati guadagni o comunque portare ad inadempienze contrattuali).

Nelle tecnologie informatiche le reti di comunicazione possono essere utilizzate spesso per trasferire template tra sistema centrale e sistemi periferici: ecco così che la realizzazione può essere attaccata in tutti i suoi componenti come la rete stessa, i computer, gli algoritmi o i protocolli, i database ed in generale nei loro punti di integrazione. In questo modo i pericoli specifici verso le singole tecnologie dell'infrastruttura sottostante possono minacciare anche la robustezza del sistema biometrico. Come in qualsiasi altro sistema informatico i pericoli riguardano i seguenti campi:

- ◇ *Tecnici*: che interessano gli apparati fisici o il software e che possono provocare l'arresto di una funzione (fallimento) oppure un altro risultato non previsto (malfunzionamento).
- ◇ *Fisici*: in questa classe sono inclusi disastri naturali come incendi, terremoti o alluvioni, ma anche condizioni ambientali di più modesto impeto, ma che col passare del tempo possono arrecare danno.

- ◇ *Umani*:. In esso sono inclusi gli utenti non autorizzati che intendono arrecare danno, oppure utenti autorizzati che mal utilizzano il sistema deliberatamente o accidentalmente. I motivi per tali malevoli propositi possono essere di natura profondamente diversa: dal semplice atto vandalico informatico, allo spionaggio industriale, al tentativo di sottrarre fondi o per pura conoscenza e dimostrazione della propria bravura; in quest’ultimo caso i danni si potrebbero limitare “solo” al danno d’immagine causato dalla violazione delle proprie macchine.
- ◇ *Teorici*: che racchiude in sé le vulnerabilità proprie degli algoritmi, dei protocolli o negli strumenti matematici utilizzati all’interno delle funzioni del sistema.

Risulta evidente che per occuparci della sicurezza di un sistema biometrico dobbiamo saper esaminarla in più campi. Ad esempio ci sarà bisogno anche di un esperto di reti per cercare di blindare l’architettura da classici attacchi che possono diventare specifici nei confronti della tecnologia biometrica: in questo caso il *replay attack* può presentarsi nelle situazioni in cui l’apparecchio di scannerizzazione non sia in grado di distinguere tra dati ricevuti in presa diretta oppure rubati in precedenza ed utilizzati per simulare un altro utente in un momento successivo, utilizzando per esempio un registratore per truffare il riconoscimento vocale.

A questo si possono aggiungere attacchi crittoanalitici per forzare la cifratura del messaggio contenete il template, oppure tecniche di *eavesdropping* che ascoltano passivamente la comunicazione in corso per ottenerne importanti informazioni, attacchi *Denial Of Service* (DOS) per bloccare semplicemente il servizio oppure usati in concomitanza con altri strumenti per penetrare all’interno del sistema come nel caso dell’”attacco del giorno di Natale” effettuato da Kevin Mitnick [23].

L'affidabilità dipende quindi sia da fattori inerenti la progettazione, sia inerenti la realizzazione vera e propria: l'analisi della matematica e dei teoremi alla base delle funzioni utilizzate devono essere attentamente valutata, dato che l'algoritmo potrebbe presentare come minimo delle "falle" nella sicurezza che renderebbero il tutto vulnerabile ed insicuro. Per quanto riguarda la parte criptografica, particolare attenzione deve essere prestata alla scelta delle chiavi utilizzate (dato che l'algoritmo non può costituire un "segreto"), all'algoritmo di codifica che deve garantire le prestazioni di privacy ed efficienza richiesta ed infine l'attenzione si deve rivolgere all'eventuale algoritmo di decriptazione, che, se esistente, deve rappresentare un problema NP-Completo.

A.6 Aree di applicazione.

In questo paragrafo parleremo delle molteplici applicazioni presenti e future nelle aree nelle quali le tecnologie biometriche si adattano con più naturalezza. Queste risultano essere rappresentate da Legge e ordine pubblico in generale, settore bancario, computer e reti, servizi pubblici e controllo degli accessi [19]. Data l'ambito in cui si muove questa tecnologia, molto spesso risulta difficile reperire informazioni riguardo i prodotti.

Servizi pubblici:

In questo campo rientrano per esempio applicazioni utili alla frontiera per poter schedare gli immigrati ed assegnarli così un'identità certa e facilmente rintracciabile, ma anche il riconoscimento automatico del passaporto, facilitare e rendere sicuro il

passaggio di merci e persone anche dove il documento non deve essere necessariamente mostrato come nei paesi dell'Unione Europea: un esempio di applicazione reale realizzata in questo campo è il progetto *CANPASS* basato su impronte digitali e carte ottiche, con lo scopo di facilitare il transito di persone e merci tra Canada e Stati Uniti nell'aeroporto di Vancouver [24]. Un altro campo di realizzazione può interessare il cosiddetto "welfare" (stato sociale) in generale: servizio sanitario, pagamenti di pensioni oppure anche semplici buoni pasto per i meno abbienti. Per questo campo è utile citare il progetto *Tarjeta Seguridad Social* (TASS) realizzato da *Spanish National Social Security Identification Card* e con lo scopo di verificare i benefici del cittadino tramite impronta digitale e smart card [25].

Computer e reti:

In questa parte di mercato il riconoscimento basato sull'impronta vocale risulta farla da padrona insieme a quella digitale, ma si sta facendo largo anche l'analisi della battitura di tasti (come il sistema BioPassword® di Net Nanny Software International Inc. [15]) data la presenza data quasi per scontata la presenza di una tastiera. Le applicazioni in questo settore possono interessare i comuni PC da ufficio, accessi a database, a sistemi di casella di posta vocale, a sistemi di teleconferenza, a linee telefoniche o risorse remote, ecc....

Settore Bancario:

Anche in questo caso la maggior parte dei sistemi adottati utilizzano il confronto di impronte digitali per rendere sicuri dati o transazioni finanziarie nelle carte di credito, in

vari punti vendita convenzionati, nelle cassette di sicurezza o comunque per accertarsi dell'identità di una persona davanti ad un bancomat o ad uno sportello. Per il cosiddetto "home banking" (operazioni bancarie da casa) può invece essere utilizzato il riconoscimento vocale attraverso l'apparecchio telefonico. Per esempio *Centrobiometrika* [26] ha realizzato applicazioni innovative della tecnologia delle impronte digitali in settori quali l'Home Banking, con l'identificazione sicura del cliente tramite Internet.

Legge e ordine pubblico:

I sistemi a sostegno della legge si basano soprattutto su dito, mano, iride, firma e riconoscimento vocale. Le applicazioni in questo settore si rivolgono alle carceri, per controllare entrate ed uscite dal penitenziario, ma anche per controllare gli arresti domiciliari in certe ore della giornata, per identificare i criminali anche trasmettendo i dati dalle macchine di pattuglia ad una centrale dati, per l'identificazione dei bambini anche anni dopo la loro scomparsa e quindi quando i loro tratti sono fortemente cambiati, oppure per rendere sicure le armi da fuoco facendole sparare solo in mano al loro padrone. Infine sono allo studio sistemi di voto da postazione remote che potrebbero facilitare notevolmente l'operazione per quanto riguarda cittadini disabili, malati, residenti all'estero o che semplicemente per esigenze di lavoro sono lontani dalla città di residenza. Un esempio può essere rappresentato da [27] ed ha l'utilità di trasmettere l'impronta digitale dalla macchina di pattuglia delle forze dell'ordine ad una centrale di controllo.

Controllo degli accessi:

Le scelte più frequentemente usate per questo campo includono le impronte digitali e la comparazione della forma della mano ed il controllo può riguardare edifici in generale, appartamenti, uffici governativi, centrali energetiche, farmacie, accessi a laboratori di università, a cartelle cliniche ospedaliere, a cabine di pilotaggio di aeroplani, a zone militari, a villaggi olimpici, etc....

Naturalmente il sistema di riconoscimento dipende fortemente anche dall'ambiente di lavoro in cui si trova ad operare: per esempio in certe zone di industrie chimiche dove è necessario adoprare guanti e maschera per la propria protezione, difficilmente potranno essere adottate tecniche basate su occhi o mani.

Appendice B: Modifiche al programma MINDTCT di NIST.

- *nist/include/lfs.h*

Note: le 4 modifiche che seguono non sono state in realtà mantenute nella versione finale, ma comunque servono per ottenere la direzione delle minuzie trovate in gradi invece che in un valore di campionamento tra 0 e 31. Questo cambiamento comporta però anche una differente lista di minuzie rilevate sulla stessa impronta.

Linea 86: modificare valore *define*

```
#define NUM_DIRECTIONS      180 /* ERA 16 */
```

Linea 317: modificare valore *define*

```
#define HIGHCURV_VORTICITY_MIN  20 /* ERA 5 */
```

Linea 322: modificare valore *define*

```
#define HIGHCURV_CURVATURE_MIN 20 /* ERA 5 */
```

Linea 368: modificare valore *define*

```
#define FORK_INTERVAL 22 /* ERA 2 */
```

Note: per cambiare il numero massimo di vicini per minuzia, modificare linea 564: modificare valore della macro *define* (questa non è una vera e propria modifica, ma più precisamente un promemoria per cambiamenti futuri)

Linea 564: modificare valore *define*

```
#define MAX_NBR5 5
```

- */nist/src/bin/mindtct/mindtct.c*

Linea 312: aggiungere codice

```
/* INSERITO */  
if (ret = count_minutiae_ridges (minutiae, bdata, iw, ih, lfsparms) ) {  
  
    free_minutiae(minutiae);  
    free(direction_map);
```

```

        free(low_contrast_map);

        free(low_flow_map);

        free(high_curve_map);

        free(quality_map);

        free(bdata);

        return(ret );

    }

```

- */nist/src/lib/lfs/detect.c*

Linea 643: commentare linea

```

/*gray2bin(1, 255, 0, bdata, iw, ih);*/

```

- */nist/src/lib/lfs/ridges.c*

Linea 48: inserire in *count_minutiae_ridges()*:

```

static int mioFlag= 0;

```

Inserire tutto il corpo della funzione (fino al return) in un costrutto *if*

```

if (mioFlag != 0) {

```

```
// CORPO count_minutiae_ridges()

}

mioFlag++;
```

NOTE: serve per non eseguire il calcolo dei vicini la prima volta.

Inserire definizione in *find_neighbors()* che serve controllare l'affidabilità della minuzia

```
double miaRel;
```

Linea 216 in *find_neighbors()*:

```
second = 0; /* era first + 1*/
```

Nel ciclo che inizia alla linea 226 in *find_neighbors()* inserire subito all'inizio per non inserire come vicino la minuzia stessa

```
if (first == second) {
    second++;
    continue;
}
```

Controllare l'affidabilità della minuzia dopo aver estratto le due minuzia dalla lista in *find_neighbors()*:

```
miaRel= minutia2->reliability;
if (miaRel < 0.2) {
    second++;
    continue;
}
```

Calcolare la distanza euclidea tra le due minuzie dopo aver controllato l'affidabilità (vedi sopra); dist2 diventa questo valore e quello che c'era prima va cancellato:

```
/* Compute squared euclidean distance between minutia pair. */
dist2 = squared_distance(minutia1->x, minutia1->y,
    minutia2->x, minutia2->y);
```

Cancellare (commentandolo) il break che esce dal ciclo se ho trovato MAX_NBRS vicini, dato che li devo scorrere tutti;

```
/* Otherwise, if the neighbor lists is full AND the x-distance */
/* to current secondary is larger than maximum neighbor distance */
/* stored ... */
/*else*/
    /* So, stop searching for more neighbors. */
    /*break;*/
```


Appendice C: listato del codice sorgente dell'applicazione su PC.

C.1 Package algMinInformation.

```
/*
 * AlgMinutia.java
 *
 * Created on 12 ottobre 2002, 11.27
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package algMinInformation;

/***** IMPORTS *****/
import minutiaeInformation.MinType;
import minutiaeSuperClasses.MinutiaSuperClass;
import minutiaeSuperClasses.MinutiaException;
import minutiaeSuperClasses.NeighborListException;

/**
 * This class inherits from <code>minutiaeSuperClasses.MinutiaSuperClass
 * </code>.<p>
```

```

* Represents minutia obtained after converting from <i>MINDTCT</i> text file.
*
* @author Francesco Santini
* @version 1.0
*/
public class AlgMinutia extends MinutiaSuperClass {

    /**
     * Minutia type (bifurcation, ridge-ending...). <p>
     * See <code>algMinInformation.AlgMinutiaType</code> class.
     */
    private AlgMinutiaType type= null;

    /**
     * Minutia neighbors list.
     */
    private AlgNeighborList fts= null;

    /**
     * Creates a new instance of AlgMinutia.
     *
     * @throws MinutiaException Thrown after failing in minutia neighbor list
     *                           creation.
     */
    public AlgMinutia() throws MinutiaException {

        try {
            this.fts= new AlgNeighborList();
        }
        catch (NeighborListException nle) {
            throw new MinutiaException("Error creating neighbor list: \n\t" +
                                      nle.getMessage());
        }

    }

    /**
     * Controls and sets direction angle for this minutia.
     *
     * @param newXPos Minutia direction angle to be set.
     *
     * @throws MinutiaException If it is a not valid direction angle (e.g. less
     *                           than zero)
     */
    public void setDirection(float newDirection) throws MinutiaException {

        // Converted from MINDTCT angle format
        if ((newDirection < -360.0) || (newDirection > 360.0)) {
            throw new MinutiaException("Direction angle out of range [-360..360]
                                      (" + newDirection + ")");
        }
        else {
            super.direction= newDirection;
        }
        return;
    }

    /**
     * Sets minutia new type. (e.g. "BIF", "RIG").

```



```

    *
    * @param newType Minutia type to be set.
    */
    public void setMinType(AlgMinutiaType newType) {
        this.type= newType;
        return;
    }

    /**
     * Inserts neighbor passed in minutia neighbor list.
     *
     * @param ft Neighbor to be inserted.
     */
    public void addAlgNeighbor(AlgNeighbor ft) {
        fts.add(ft);
    }


    /**
     * Inserts neighbor passed in minutia neighbor list. Neighbor list is
     * ordered.
     * See method <code>addOrdered</code> in <code>
     * algMinInformation.AlgNeighborList </code>.
     *
     * @param ft Neighbor to be inserted.
     *
     * @throws MinutiaException If this method fails to add a minutia to minutia
     * list ordered).
     */
    public void addOrderedAlgNeighbor(AlgNeighbor ft) throws MinutiaException {
        try {
            fts.addOrdered(ft);
        }
        catch (NeighborListException nle) {
            throw new MinutiaException("Error adding ordered to list: \n\t" +
                nle.getMessage());
        }
    }

    /**
     * Returns <code>minutiaeInformation.AlgMinutiaType</code> minutia type.
     *
     * @return Minutia type.
     */
    public AlgMinutiaType getType() {
        return(this.type);
    }

    /**
     * Returns neighbor from passed position.
     *

```

```

* @param index Index of neighbor requested (from zero to <i>size - 1</i>).
* @return Returns neighbor <code>index</code>.
*
* @throws MinutiaException If this method fail to get minutia at that
*                           index.
*/
public AlgNeighbor get(int index) throws MinutiaException {

    try {
        return (fts.get(index));
    }
    catch (NeighborListException nle) {
        throw new MinutiaException("Error getting element from list: \n\t" +
                                   nle.getMessage());
    }

}

/**
 * Returns minutia neighbors.
 *
 * @return This minutia neighbor list.
 */
public AlgNeighborList getAlgNeighborList() {

    return (this.fts);

}

/**
 * Prints on <code>System.out</code> minutia and its neighbors.
 *
 * @throws MinutiaException If fails to scan the list.
 */
public void print() throws MinutiaException {

    System.out.println("Matching Algorithm minutia identifier = " +
                       this.id);
    System.out.println("\t- x position = " + this.xPos);
    System.out.println("\t- y position = " + this.yPos);
    System.out.println("\t- angle = " + this.direction);
    System.out.println("\t- type = " + this.type.getMinTypeString());
    System.out.println("\t- reliability = " + this.reliability);
    int s= this.fts.size();
    System.out.println("\tMinutia has " + s + " neighbors.");
    AlgNeighbor rifAlgNeighbor= null;
    for (int i= 0; i < s; i++) {
        try {
            rifAlgNeighbor= (AlgNeighbor) (this.fts.get(i));
        }
        catch (NeighborListException nle) {
            throw new MinutiaException(nle.getMessage());
        }

        System.out.println("\t\t- " +
                           rifAlgNeighbor.stringRappresentation());
    }

}

}

```

```

/*
 * AlgMinutiaType.java
 *
 * Created on 12 ottobre 2002, 19.36
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package algMinInformation;

```

```

/***** IMPORTS *****/

```

```

import minutiaeSuperClasses.MinTypeSuperClass;
import minutiaeSuperClasses.MinTypeException;

```

```

/**
 * Class extended from <code>minutiaeSuperClasses.MinTypeSuperClass</code>.
 * Represents minutia type for algorithm compatible minutiae.
 */

```

```

 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class AlgMinutiaType extends MinTypeSuperClass {

```

```

    /**
     * Creates a new instance of AlgMinType.
     *
     * @param typeString String representing minutia type: can be <i>"BIF"
     *                  "RIG"</i> or <i>"NO TYPE"</i>.
     *
     * @throws MinTypeException If passed string is not one of those above.
     */

```

```

    public AlgMinutiaType(String typeString) throws MinTypeException {

        if (typeString.equalsIgnoreCase(super.bifString))
            super.type= 0;
        else
            if (typeString.equalsIgnoreCase(super.ridgeEndingString))
                super.type= 1;
            else
                throw new MinTypeException("Minutia Type not allowed (" +
                    typeString + ")");
    }

```

```

/**
 * Returns a boolean true if type is bifurcation type
 *
 * @param anotherMinType Other minutia for the comparison.
 * @return <i>True</i> if is a "BIF" type, <i>false</i> otherwise.
 */
public boolean equalType(AlgMinutiaType anotherMinType) {
    return (super.type == anotherMinType.getMinType());
}
}

```

```

/*
 * AlgMinutiaeList.java
 *
 * Created on 12 ottobre 2002, 11.58
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package algMinInformation;

```

```

/***** IMPORTS *****/

```

```

import java.util.Vector;
import minutiaeSuperClasses.MinutiaListSuperClass;
import minutiaeSuperClasses.MinutiaException;
import minutiaeSuperClasses.MinutiaeListException;

```

```

/**
 * Class representing minutiae list converted from <i>MINDTCT</i> text file in a
 * new format.<p>
 * Extended from <code>minutiaeSuperClasses.MinutiaeListSuperClass</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class AlgMinutiaeList extends MinutiaListSuperClass {

```

```

    /**
     * Max distance between central minutia and its neighbors in this minutiae
     * list
     */
    private float maxNeighDistance= 0;

```

```

    /**
     * Maximum ridge count value in this list between a minutia and its
     * neighbors.
     */
    private short maxRidgeCount= 0;

```

```

    /**
     * Creates a new instance of AlgMinutiaeList.
     *
     * @throws MinutiaeListException If fails to create a new list (e.g error in
     * new java.util.Vector parameters).
     */

```

```

    */
    public AlgMinutiaeList() throws MinutiaeListException {
        try {
            super.list= new Vector(super.initialCapacity,
                                   super.capacityIncrement);
        }
        catch (IllegalArgumentException iae) {
            throw new MinutiaeListException("Error creating vector: \n\t" +
                                             iae.getMessage());
        }
    }

    /**
     * Add a new minutia to this minutiae list.
     *
     * @param min Minutia to insert.
     */
    public void add(AlgMinutia am) {
        super.list.add(am);
        super.elementNumber++;
    }

    /**
     * Returns a list with the first <code>listSize</code> minutiae ordered by
     * reliability. Parameter n is now set to 20.
     *
     * @param listSize Number of minutiae in new list.
     * @return A new list containing the first n minutiae ordered before by
     *         reliability.
     *
     * @throws MinutiaeListException Thrown after failing to cut the list (e.g.
     *         error manipulating input list).
     */
    public AlgMinutiaeList cutByReliability(int listSize) throws
        MinutiaeListException {

        // Controlling input parameters
        if (listSize <= 0)
            throw new MinutiaeListException("Error on input parameter: must be >
                                             0.");

        // Size of returned list
        int rMinNumber= listSize;

        int size= super.list.size();

        // Returning this list if contains less minutiae than parameter
        // "listSize"
        if (size <= listSize)
            return (this);

        // Return list
        AlgMinutiaeList returnList= new AlgMinutiaeList();

        try {
            returnList.setMaxNeighDistance(this.getMaxNeighDistance());

```

```

    }
    catch (MinutiaeListException mle) {
        throw mle;
    }

    try {
        returnList.setMaxRidgeCount(this.getMaxRidgeCount());
    }
    catch (MinutiaeListException mle) {
        throw mle;
    }

    // Scanning all this list to order by reliability
    for (int i= 0; i < size; i++) {

        AlgMinutia tempMin= null;
        try {
            tempMin= (AlgMinutia) (this.get(i));
        }
        catch(ArrayIndexOutOfBoundsException aioobe) {
            throw new MinutiaeListException("Error ordering minutiae list
                                             using reliability: \n\t" +
                                             aioobe.getMessage());
        }

        // Inserting extracted minutiae in new list ordering by reliability
        try {
            returnList.insertOrdered(tempMin);
        }
        catch (MinutiaeListException mle) {
            throw new MinutiaeListException("Error inserting minutia in list
                                             ordered by reliability: \n\t" +
                                             mle.getMessage());
        }
    }

    // Changing new list minutiae Identifiers
    for (short i= 0; i < rMinNumber; i++) {
        try {
            returnList.get(i).setId(i);
        }
        catch(MinutiaException me) {

        }
    }

    // Taking first rMinNumber minutiae of the list removing other minutiae.
    try {
        while (returnList.size() > rMinNumber)
            returnList.removeElementAt(rMinNumber);
    }
    catch (MinutiaeListException mle) {
        throw new MinutiaeListException("Error removing element in position
                                         " + rMinNumber + " : \n\t");
    }

    return (returnList);
}

```

```

/**
 * Removes minutia in position <i>i</i> from this minutiae list.

```

```

*
* @param i Index of minutia to be removed.
*
* @throws MinutiaeListException If fails to remove minutia from the list:
*                               error on list index.
*/
private void removeElementAt(int i) throws MinutiaeListException {
    try {
        super.list.removeElementAt(i);
    }
    catch(ArrayIndexOutOfBoundsException aioobe) {
        throw new MinutiaeListException();
    }
}

/**
* Inserts new minutia in the list, keeping the list ordered by reliability
* field.
*
* @param insMin Minutia to be inserted in this list.
*
* @throws MinutiaeListException If fails to scan minutiae list (error with
*                               array indexes).
*/
private void insertOrdered(AlgMinutia insMin) throws MinutiaeListException {
    int i= 0;
    int size= super.list.size();

    // Scanning the list to find correct position
    while (i < size) {
        AlgMinutia tempMin= null;
        try {
            tempMin= (AlgMinutia) (super.list.get(i));
        }
        catch (ArrayIndexOutOfBoundsException aioobe) {
            throw new MinutiaeListException(aioobe.getMessage());
        }

        if (tempMin.getReliability() < insMin.getReliability())
            break;
        else
            i++;
    }

    // Adding list in correct position
    try {
        super.list.add(i, insMin);
    }
    catch (ArrayIndexOutOfBoundsException aioobe) {
        throw new MinutiaeListException(aioobe.getMessage());
    }
}

/**
* Gets minutia in position index from this minutia list.

```



```

*
* @param index Index of minutia requested.
* @return Minutia in position index from list.
*
* @throws MinutiaeListException Error getting minutia from list (e.g. index
*                                out of bounds).
*/
public AlgMinutia get(int index) throws MinutiaeListException {
    AlgMinutia ret= null;
    try {
        ret= (AlgMinutia) (super.list.get(index));
    }
    catch (ArrayIndexOutOfBoundsException aioobe) {
        throw new MinutiaeListException("Error getting " + index + "
                                         element: \n\t" +
                                         aioobe.getMessage());
    }
    return (ret);
}

/**
 * Prints this minutiae list.
 *
 * @throws MinutiaeListException If fails to print list (e.g. error scanning
 *                                list).
 */
public void print() throws MinutiaeListException {
    int size= super.size();
    System.out.println("In the list there are " + size + " minutiae.\n");
    AlgMinutia rifMin= null;
    for (int i= 0; i < size; i++) {
        try {
            rifMin= (AlgMinutia) (super.list.get(i));
        }
        catch (ArrayIndexOutOfBoundsException aioobe) {
            throw new MinutiaeListException("Error getting " + i + "
                                             element: \n\t" +
                                             aioobe.getMessage());
        }

        try {
            rifMin.print();
        }
        catch (MinutiaException me) {
            throw new MinutiaeListException("Error printing minutia " +
                                             rifMin.getId() + ": \n\t" +
                                             me.getMessage());
        }

        System.out.println("");
    }
}

/**
 * Returns max distance between central minutia and its neighbor in this
 * list.
 *
 * @return Maximum distance between central minutia and its neighbor in this

```

```

        *          list.
        */
public float getMaxNeighDistance() {
    return (this.maxNeighDistance);
}

/**
 * Returns maximum ridge count distance between a minutia and its neighbors
 * in this list.
 *
 * @return Maximum ridge count.
 */
public short getMaxRidgeCount() {
    return (this.maxRidgeCount);
}

/**
 * Sets max distance between central minutia and its neighbor in this list.
 *
 * @param newMaxDistance Maximum distance between central minutia and its
 *                        neighbor in this list.
 *
 * @throws MinutiaeListException If new max distance is invalid (less than
 *                                zero).
 */
public void setMaxNeighDistance(float newMaxDistance) throws
    MinutiaeListException {
    if (newMaxDistance < 0)
        throw new MinutiaeListException("Max distance between central
                                         minutia and its neighbor can't be
                                         less than zero");

    this.maxNeighDistance= newMaxDistance;
}

/**
 * Sets maximum ridge count distance between a minutia and its neighbors
 * in this list.
 *
 * @param maxRC Maximum ridge count.
 *
 * @throws MinutiaeListException If new maximum ridge count is invalid (less
 *                                than zero).
 */
public void setMaxRidgeCount(short maxRC) throws MinutiaeListException {
    if (maxRC < 0)
        throw new MinutiaeListException("Max ridge count between minutiae
                                         can't be less than zero");

    this.maxRidgeCount= maxRC;
}
}

```

```

/*
 * AlgNeighbor.java
 *
 * Created on 9 ottobre 2002, 16.56
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package algMinInformation;
```

```
/****** IMPORTS *****/
```

```
import minutiaeSuperClasses.NeighborSuperClass;
import minutiaeSuperClasses.NeighborException;
```

```
/**
```

```
 * Class representing neighbor converted from <i>MINDTCT</i> text file in a new
 * format.<p>
 * Extended from abstract class in <code>
 * minutiaeSuperClasses.NeighborSuperClass </code>.
 *
```

```
 * @author Francesco Santini
 * @version 1.0
 */
```

```
public class AlgNeighbor extends NeighborSuperClass {
```

```
    /**
     * Neighbor (minutia) identifier.
     */
```

```
    private short id= 0;
```

```
    /**
     * Euclidean distance between neighbor and its central minutia.
     */
```

```
    private float distance= 0;
```

```
    /**
     * Angle between line connecting neighbor and its central minutia and
     * neighbor direction angle.
     */
```

```
    private float distanceRelativeAngle= 0;
```

```
    /**
     * Difference between neighbor direction angle and its central minutia
     * direction angle.
     */
```

```
    private float relativeRidgeAngle= 0;
```

```

/**
 * Additional feature
 */
public float fifthFeature= 0;

/**
 * Creates a new instance of AlgNeighbor.
 */
public AlgNeighbor() {

}

/**
 * Creates a new instance of AlgNeighbor with passed parameters.
 *
 * @param newId Neighbor identifier of new neighbor.
 * @param newDistance Euclidean distance of new neighbor.
 * @param newDistanceRelativeAngle Distance relative angle of new neighbor.
 * @param newRelativeAngle Relative angle of new neighbor.
 * @param newRidgeCount Ridge count of new neighbor.
 */
public AlgNeighbor(short newId, float newDistance, float
                    newDistanceRelativeAngle, short newRelativeAngle, short
                    newRidgeCount) {

    this.id= newId;
    this.distance= newDistance;
    this.distanceRelativeAngle= newDistanceRelativeAngle;
    this.relativeRidgeAngle= newRelativeAngle;
    this.ridgeCount= newRidgeCount;

}

// Methods to get fields

/**
 * Controls and sets identifier for this neighbor.
 *
 * @param newId Neighbor identifier to be set.
 * @throws NeighborException If it is a not valid identifier (e.g. less than
 *                             zero)
 */
public void setId(short newId) throws NeighborException {

    if (newId < 0) {
        throw new NeighborException("Negative minutia identification number
                                    (" + newId + ")");
    }
    else {
        this.id= newId;
    }
    return;

}

/**
 * Controls and sets euclidean distance between this neighbor and its
 * central minutia.

```

```

*
* @param newDistance New euclidean distance to be set.
*
* @throws NeighborException If it is a not valid euclidean distance (e.g.
*                             less than zero).
*/
public void setDistance(float newDistance) throws NeighborException {
    if (newDistance <= 0.0)
        throw new NeighborException("Distance between minutia not valid " +
                                    newDistance + "");
    this.distance= newDistance;
    return;
}

/**
 * Controls and sets angle between line connecting neighbor and its central
 * minutia and neighbor direction angle.
 *
 * @param newDistanceRelativeAngle New distance-relative angle.
 *
 * @throws NeighborException If it is a not valid distance relative angle
 *                             (e.g. less than -360 or above 360).
 */
public void setDistanceRelativeAngle(float newDistanceRelativeAngle) throws
    NeighborException {
    if ((newDistanceRelativeAngle < -360.0) || (newDistanceRelativeAngle >
        360.0))
        throw new NeighborException("Distance-relative angle not valid " +
                                    newDistanceRelativeAngle + "");
    this.distanceRelativeAngle= newDistanceRelativeAngle;
    return;
}

/**
 * Controls and sets difference between neighbor direction angle and its
 * central minutia direction angle.
 *
 * @param newRelativeAngle New distance relative angle to be set..
 *
 * @throws NeighborException If it is a not valid relative angle
 *                             (e.g. less than -360 or above 360).
 */
public void setRelativeAngle(float newRelativeAngle) throws
    NeighborException {
    if ((newRelativeAngle < -360.0) || (newRelativeAngle > 360.0))
        throw new NeighborException("Relative angle not valid " +
                                    newRelativeAngle + "");
    this.relativeRidgeAngle= newRelativeAngle;
    return;
}

// Methods to get fields
/**

```

```

    * Returns neighbor identifier.
    *
    * @return Neighbor identifier.
    */
public short getId() {
    return(this.id);
}

/**
 * Returns neighbor euclidean distance from central minutia.
 *
 * @return Neighbor euclidean distance.
 */
public float getDistance() {
    return(this.distance);
}

/**
 * Returns angle between line connecting neighbor and its central minutia
 * and neighbor direction angle.
 *
 * @return Neighbor distance relative angle.
 */
public float getDistanceRelativeAngle() {
    return(this.distanceRelativeAngle);
}

/**
 * Difference between neighbor direction angle and its central minutia
 * direction angle.
 *
 * @return Neighbor relative angle.
 */
public float getRelativeAngle() {
    return(this.relativeRidgeAngle);
}

/**
 * Returns a text representation of this neighbor.
 *
 * @return String representation of this neighbor.
 */
public String stringRappresentation() {
    String retString= new String("id = " + this.id + " , distance = " +
        this.distance + " , distance-relative angle
        = " + this.distanceRelativeAngle + " ,
        relative angle = "+this.relativeRidgeAngle
        + " , ridge count = " + this.ridgeCount);
    return (retString);
}
}

```

```

/*
 * AlgNeighborList.java
 *
 * Created on 12 ottobre 2002, 11.09
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package algMinInformation;
```

```
/****** IMPORTS *****/
```

```
import java.util.Vector;
import minutiaeSuperClasses.NeighborListSuperClass;
import minutiaeSuperClasses.NeighborListException;
```

```

/**
 * Class representing minutia neighbor list converted from <i>MINDTCT</i> text
 * file in a new format.<p>
 * Extended from <code>minutiaeSuperClasses.MinutiaeListSuperClass</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class AlgNeighborList extends NeighborListSuperClass {
```

```

    /**
     * Creates a new instance of AlgNeighborList.
     *
     * @throws NeighborListException If fails to create a new list (e.g error in
     *                               new java.util.Vector parameters).
     */
    public AlgNeighborList() throws NeighborListException {

        try {
            super.list= new Vector(super.initialCapacity,
                                   super.capacityIncrement);
        }
        catch (IllegalArgumentException iae) {
            throw new NeighborListException("Error creating vector: \n\t" +
                                             iae.getMessage());
        }

    }
}

```

```

/**
 * Add a new neighbor to this minutia neighbor list.
 *
 * @param ft Neighbor to insert.
 */
public void add(AlgNeighbor ft) {

    super.list.add(ft);
    super.elementNumber++;

}

/**
 *
 * Inserts neighbor in neighborlist keeping distance-relative angle
 * order.<p>
 * Neighbor with lower distance-relative angle are in first positions:
 * increasing order.
 *
 * @param ft Neighbor to be inserted.
 *
 * @throws NeighborListException If fails to insert neighbor (e.g. error
 * scanning the list.)
 */
public void addOrdered(AlgNeighbor ft) throws NeighborListException {

    int size= super.elementNumber;
    for (int i= 0; i < size; i++) {
        AlgNeighbor fTemp= null;
        try {
            fTemp= (AlgNeighbor) (super.list.get(i));
        }
        catch (ArrayIndexOutOfBoundsException aioobe) {
            throw new NeighborListException("Error getting " + i + "
                element: \n\t" +
                aioobe.getMessage());
        }

        if ((ft.getDistanceRelativeAngle()) <
            (fTemp.getDistanceRelativeAngle())) {
            super.list.add(i, ft);
            return;
        }
    }

    // Inserting even if list is empty.
    list.add(ft);
}

/**
 * Gets neighbor in position index from this minutia neighbor list.
 *
 * @param index Index of neighbor requested.
 * @return Neighbor in position index from list.
 *
 * @throws NeighborListException Error getting neighbor from list (e.g.
 * index out of bounds).
 */
public AlgNeighbor get(int index) throws NeighborListException {

```



```

        AlgNeighbor ret= null;
        try {
            ret= (AlgNeighbor) (this.list.get(index));
        }
        catch (ArrayIndexOutOfBoundsException aioobe) {
            throw new NeighborListException("Error getting " + index + "
                                             element: \n\t" +
                                             aioobe.getMessage());
        }
        return ret;
    }

    /**
     * Returns neighbor list size
     *
     * @return number of element in the list.
     */
    public int size() {
        return (super.list.size());
    }
}

```

C.2 Package algParsing.

```
/*
 * AlgNeighborParser.java
 *
 * Created on 15 ottobre 2002, 18.56
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
```

```
package algParsing;
```

```
/****** IMPORTS *****/
import java.util.StringTokenizer;
import java.util.NoSuchElementException;
import algMinInformation.AlgMinutia;
import algMinInformation.AlgNeighbor;
import minutiaeSuperClasses.NeighborException;

/**
 * Helps <code>AlgParser.parse</code> parsing a string containing minutiae
 * information. Exactly, method <code>parse</code> of this class, parses a
 * minutia neighbor. Text format parsed by this class is created from
 * <i>FinMatchWriter</i> application starting from a <i>MINDTCT</i> file.
 *
 * @author Francesco Santini
 * @version 1.0
 */
```

```
public class AlgNeighborParser {

    /**
     * String used to separate information fields ":".
     */
    private static final String sep= ":";

    /**
     * java.util.StringTokenizer used in parsing.
     */
    private StringTokenizer neighborTokenizer= null;

    /**
     * Minutia to be filled with its neighbors.
     */
```

```

    */
private AlgMinutia min= null;

/**
 * Creates a new instance of AlgNeighborParser.
 *
 * @param newStringTokenizer <code>StringTokenizer</code> created by <code>
 *                               AlgParser.parse</code>: contains tokenized line
 *                               from input stream.
 * @param newMin <code>Minutia</code> the parser is going to insert in the
 *                               list
 *
 * @throws AlgParserException Thrown if it fails to parse a minutia
 *                               neighbor
 */
public AlgNeighborParser(StringTokenizer newStringTokenizer, AlgMinutia
                           newMin) throws AlgParserException {

    if ((newStringTokenizer == null) || (newMin == null))
        throw new AlgParserException("Null input parameter");
    this.neighborTokenizer= newStringTokenizer;
    this.min= newMin;
}

/**
 * Parses one neighbor substring.
 *
 * @throws AlgParserException Thrown after failing to parse text file (e.g.
 *                               invalid data format in text file).
 */
public void parse() throws AlgParserException {

    AlgNeighbor newF= new AlgNeighbor();

    try {
        // Retrieving neighbor id
        short id= 0;
        if (neighborTokenizer.hasMoreTokens()) {
            Short idTemp= null;
            String tokenString= null;
            try {
                tokenString= neighborTokenizer.nextTokent();
                idTemp= new Short(tokenString);
            }
            catch (NumberFormatException nfe) {
                throw new AlgParserException("Invalid number format (" +
                                                tokenString + ")");
            }
            id= idTemp.shortValue();
        }
        else
            throw new AlgParserException("Unexpected line termination.");

        // Inserting id neighbor field
        try {
            newF.setId(id);
        }
        catch (NeighborException ne) {
            throw new AlgParserException("Error setting id at neighbor " +
                                          id + ": \n\t" + ne.getMessage());
        }
    }
}

```

```

// Controlling separator
if (neighborTokenizer.hasMoreTokens()) {
    if (! this.sep.equalsIgnoreCase(neighborTokenizer.nextToken()))
        throw new AlgParserException("Missing separator " + this.sep
                                       + " at neighbor " + id + ".");
}
else
    throw new AlgParserException("Unexpected line termination at
                                neighbor " + id + ".");

// Retrieving euclidean distance between minutia and this neighbor
float distance= 0;
if (neighborTokenizer.hasMoreTokens()) {
    Float distanceTemp= null;
    String tokenString= null;
    try {
        tokenString= neighborTokenizer.nextToken();
        distanceTemp= new Float(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new AlgParserException("Invalid number format (" +
                                     tokenString + ")");
    }
    distance= distanceTemp.floatValue();
}
else
    throw new AlgParserException("Unexpected line termination.");

// Inserting euclidean distance in neighbor field
try {
    newF.setDistance(distance);
}
catch (NeighborException ne) {
    throw new AlgParserException("Error at neighbor " + id + "
                                setting neighbor euclidean
                                distance: " + distance + "\n\t" +
                                ne.getMessage());
}

// Controlling separator
if (neighborTokenizer.hasMoreTokens()) {
    if (! this.sep.equalsIgnoreCase(neighborTokenizer.nextToken()))
        throw new AlgParserException("Missing separator " + this.sep
                                       + " at neighbor " + id + ".");
}
else
    throw new AlgParserException("Unexpected line termination at
                                neighbor " + id + ".");

// Retrieving distance-relative angle between minutia and this
// neighbor
float distanceRelativeAngle= 0;
if (neighborTokenizer.hasMoreTokens()) {
    Float distanceRelativeAngleTemp= null;
    String tokenString= null;
    try {
        tokenString= neighborTokenizer.nextToken();
        distanceRelativeAngleTemp= new Float(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new AlgParserException("Invalid number format (" +

```

```

        tokenString + "));
    }
    distanceRelativeAngle= distanceRelativeAngleTemp.floatValue();
}
else
    throw new AlgParserException("Unexpected line termination.");

// Inserting distance-relative angle in neighbor field
try {
    newF.setDistanceRelativeAngle(distanceRelativeAngle);
}
catch (NeighborException ne) {
    throw new AlgParserException("Error at neighbor " + id + "
                                setting neighbor distance-relative
                                angle: " + distanceRelativeAngle +
                                "\n\t" + ne.getMessage());
}

// Controlling separator
if (neighborTokenizer.hasMoreTokens()) {
    if (! this.sep.equalsIgnoreCase(neighborTokenizer.nextTokn()))
        throw new AlgParserException("Missing separator " + this.sep
                                       + " at neighbor " + id + ".");
}
else
    throw new AlgParserException("Unexpected line termination at
                                neighbor " + id + ".");

// Retrieving relative angle between minutia and this neighbor
float relativeAngle= 0;
if (neighborTokenizer.hasMoreTokens()) {
    Float relativeAngleTemp= null;
    String tokenString= null;
    try {
        tokenString= neighborTokenizer.nextTokn();
        relativeAngleTemp= new Float(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new AlgParserException("Invalid number format (" +
                                       tokenString + ")");
    }
    relativeAngle= relativeAngleTemp.shortValue();
}
else
    throw new AlgParserException("Unexpected line termination.");

// Inserting relative angle in neighbor field
try {
    newF.setRelativeAngle(relativeAngle);
}
catch (NeighborException ne) {
    throw new AlgParserException("Error at neighbor " + id + "
                                setting neighbor relative angle
                                count: " + relativeAngle + "\n\t" +
                                ne.getMessage());
}

// Controlling separator
if (neighborTokenizer.hasMoreTokens()) {
    if (! this.sep.equalsIgnoreCase(neighborTokenizer.nextTokn()))
        throw new AlgParserException("Missing separator " + this.sep
                                       + " at neighbor " + id + ".");
}

```

```

    }
    else
        throw new AlgParserException("Unexpected line termination at
                                     neighbor " + id + ".");

    // Retrieving Ridge count between minuita and this neighbor
    short ridgeCount= 0;
    if (neighborTokenizer.hasMoreTokens()) {
        Short ridgeCountTemp= null;
        String tokenString= null;
        try {
            tokenString= neighborTokenizer.nextToken();
            ridgeCountTemp= new Short(tokenString);
        }
        catch (NumberFormatException nfe) {
            throw new AlgParserException("Invalid number format (" +
                                         tokenString + ")");
        }
        ridgeCount= ridgeCountTemp.shortValue();
    }
    else
        throw new AlgParserException("Unexpected line termination.");

    // Inserting ridge count in neighbor field
    try {
        newF.setRidgeCount(ridgeCount);
    }
    catch (NeighborException ne) {
        throw new AlgParserException("Error at neighbor " + id + "
                                     setting neighbor ridge count: " +
                                     ridgeCount + "\n\t" +
                                     ne.getMessage());
    }

}

}
catch (NoSuchElementException nsee) {
    throw new AlgParserException("Unexpected no more token error: \n\t"
                                + nsee.getMessage());
}

// Inserting new neighbor in minutia
min.addAlgNeighbor(newF);
}
}

```

```

/*
 * AlgParser.java
 *
 * Created on 15 ottobre 2002, 16.13
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package algParsing;

```

```

/***** IMPORTS *****/

```

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.NoSuchElementException;
import algMinInformation.AlgMinutiaeList;
import algMinInformation.AlgMinutia;
import minutiaeSuperClasses.MinutiaeException;
import minutiaeSuperClasses.MinutiaeListException;
import algMinInformation.AlgMinutiaType;
import minutiaeSuperClasses.MinTypeException;

```

```

/**

```

```

 * Method <code>parse</code> of this class parses a file containing text
 * information for matching algorithm. You can write this type of file using
 * <i>FinMatchWriter</i>.

```

```

 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class AlgParser {

```

```

    /**

```

```

     * String used to separate information fields ":".
    */

```

```

    private static final String sep= ":";

```

```

    /**

```

```

     * Creates a new instance of AlgParser.
    */

```

```

    public AlgParser() {
    }

```

```

/**
 * Parses minutia from a file containing information needed by matching
 * algorithm. This type of file is <i>FinMatch</i> input.
 *
 * @param fr <code>FileReader</code> for the file to be parsed
 * @param list minutiae list: method <code>AlgParser.parse</code> puts
 *         here founded minutiae
 *
 * @throws AlgParserException If it fails to parse a line (e.g. invalid
 *         format)
 */
public void parse(FileReader fr, AlgMinutiaeList list) throws
        AlgParserException {

    // Changing input reader
    BufferedReader br= new BufferedReader(fr);

    // Reading line after line
    String lineString= null;
    int countLine= 1;

    // Controlling input parameters
    if ((fr == null) || (list == null))
        throw new AlgParserException("Null input parameter");

    try {
        lineString= br.readLine();
    }
    catch (IOException ioe) {
        throw new AlgParserException("Error reading " + countLine + "
                                     line");
    }

    // Reading global information about minutiae list
    StringTokenizer initialLineTokenizer= new StringTokenizer(lineString,
                                                                ":", true);

    // Reading max distance between central minutia and its neighbors
    float maxNeighDistance= 0;
    if (initialLineTokenizer.hasMoreTokens()) {
        Float maxNeighDistanceTemp= null;
        String tokenString= null;
        try {
            tokenString= initialLineTokenizer.nextToken();
            maxNeighDistanceTemp= new Float(tokenString);
        }
        catch (NumberFormatException nfe) {
            throw new AlgParserException("Invalid number format (" +
                                         tokenString + ") at line " +
                                         countLine);
        }
        maxNeighDistance= maxNeighDistanceTemp.shortValue();
    }
    else
        throw new AlgParserException("Unexpected " + countLine + " line
                                     termination.");

    // Inserting max distance between central minutia and its neighbors
    try {
        list.setMaxNeighDistance(maxNeighDistance);
    }
    catch (MinutiaeListException mle) {

```



```

        throw new AlgParserException("Error setting minutia maximum x
                                   position: \n\t" + mle.getMessage());
    }

    // Controlling separator
    if (initialLineTokenizer.hasMoreTokens()) {
        if (! this.sep.equalsIgnoreCase(initialLineTokenizer.nextToken()))
            throw new AlgParserException("Missing separator " + this.sep + "
                                         after maximum x position at line "
                                         + countLine);
    }
    else
        throw new AlgParserException("Unexpected line termination at line "
                                     + countLine);

    // Reading minutia max ridge count position
    short maxRidgeCount= 0;
    if (initialLineTokenizer.hasMoreTokens()) {
        Short maxRidgeCountTemp= null;
        String tokenString= null;
        try {
            tokenString= initialLineTokenizer.nextToken();
            maxRidgeCountTemp= new Short(tokenString);
        }
        catch (NumberFormatException nfe) {
            throw new AlgParserException("Invalid number format (" +
                                         tokenString + ") at line " +
                                         countLine);
        }
        maxRidgeCount= maxRidgeCountTemp.shortValue();
    }
    else
        throw new AlgParserException("Unexpected " + countLine + " line
                                     termination.");

    // Inserting max ridge count
    try {
        list.setMaxRidgeCount(maxRidgeCount);
    }
    catch (MinutiaeListException mle) {
        throw new AlgParserException("Error setting minutia maximum ridge
                                     count: \n\t" + mle.getMessage());
    }

    // Controlling separator
    if (initialLineTokenizer.hasMoreTokens()) {
        if (! this.sep.equalsIgnoreCase(initialLineTokenizer.nextToken()))
            throw new AlgParserException("Missing separator " + this.sep + "
                                         after maximum y position at line "
                                         + countLine);
    }
    else
        throw new AlgParserException("Unexpected line termination at line "
                                     + countLine);

    countLine++;

    try {
        lineString= br.readLine();
    }
    catch (IOException ioe) {
        throw new AlgParserException("Error reading " + countLine + "

```

```

        line");
    }

    while (lineString != null) {

        StringTokenizer lineTokenizer= new StringTokenizer(lineString, ":",
                                                            true);

        // Creates a new empty minutia
        AlgMinutia min= null;
        try {
            min= new AlgMinutia();
        }
        catch (MinutiaException me) {
            throw new AlgParserException("Error creating new minutia: \n\t"
                                         + me.getMessage());
        }

        try {

            // Retrieving minutia identificator
            short id= 0;
            if (lineTokenizer.hasMoreTokens()) {
                Short idTemp= null;
                String tokenString= null;
                try {
                    tokenString= lineTokenizer.nextToken();
                    idTemp= new Short(tokenString);
                }
                catch (NumberFormatException nfe) {
                    throw new AlgParserException("Invalid number format (" +
                                                  tokenString + ")");
                }
                id= idTemp.shortValue();
                if (id != list.size())
                    throw new AlgParserException("Error on minutia
                                                  identification number: not
                                                  a sequence (" + id + ")");
            }
            else
                throw new AlgParserException("Unexpected line termination at
                                              line " + countLine);

            // Inserting id in minutia field
            try {
                min.setId(id);
            }
            catch (MinutiaException ame) {
                throw new AlgParserException("Error setting id at minutia "
                                             + id + ": \n\t" +
                                             ame.getMessage());
            }

            // Controlling separator
            if (lineTokenizer.hasMoreTokens()) {
                if (! this.sep.equalsIgnoreCase(lineTokenizer.nextToken()))
                    throw new AlgParserException("Missing separator " +
                                                  this.sep + " at minutia " +
                                                  id + ".");
            }
            else
                throw new AlgParserException("Unexpected line termination

```

```

        at minutia " + id + ".");

// Retrieving minutia x position
short xPos= 0;
if (lineTokenizer.hasMoreTokens()) {
    Short xPosTemp= null;
    String tokenString= null;
    try {
        tokenString= lineTokenizer.nextToken();
        xPosTemp= new Short(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new AlgParserException("Invalid number format (" +
            tokenString + ") at minutia " +
            id + ".");
    }
    xPos= xPosTemp.shortValue();
}
else
    throw new AlgParserException("Unexpected line termination at
        minutia " + id + ".");

// Inserting x position in minutia field
try {
    min.setXPos(xPos);
}
catch (MinutiaException me) {
    throw new AlgParserException("Error setting x position at
        minutia " + id + ": " + xPos +
        "\n\t" + me.getMessage());
}

// Controlling separator
if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep.equalsIgnoreCase(lineTokenizer.nextToken()))
        throw new AlgParserException("Missing separator " +
            this.sep + " at minutia " +
            id + ".");
}
else
    throw new AlgParserException("Unexpected line termination at
        minutia " + id + ".");

// Retrieving minutia y position
short yPos= 0;
if (lineTokenizer.hasMoreTokens()) {
    Short yPosTemp= null;
    String tokenString= null;
    try {
        tokenString= lineTokenizer.nextToken();
        yPosTemp= new Short(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new AlgParserException("Invalid number format (" +
            tokenString + ") at minutia
            " + id + ".");
    }
    yPos= yPosTemp.shortValue();
}
else

```

```

        throw new AlgParserException("Unexpected line termination at
                                     minutia " + id + ".");

// Inserting y position in minutia field
try {
    min.setYPos(yPos);
}
catch (MinutiaException me) {
    throw new AlgParserException("Error setting y position at
                                minutia " + id + ": " + yPos +
                                "\n\t" + me.getMessage());
}

// Controlling separator
if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep.equalsIgnoreCase(lineTokenizer.nextTokn()))
        throw new AlgParserException("Missing separator " +
                                     this.sep + " at minutia " +
                                     id + ".");
}
else
    throw new AlgParserException("Unexpected line termination at
                                minutia " + id + ".");

// Retrieving minutia direction angle
float direction= 0;
if (lineTokenizer.hasMoreTokens()) {
    Float directionTemp= null;
    String tokenString= null;
    try {
        tokenString= lineTokenizer.nextTokn();
        directionTemp= new Float(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new AlgParserException("Invalid number format (" +
                                     tokenString + ") at minutia
                                     " + id + ".");
    }
    direction= directionTemp.floatValue();
}
else
    throw new AlgParserException("Unexpected line termination at
                                minutia " + id + ".");

// Inserting direction angle in minutia field
try {
    min.setDirection(direction);
}
catch (MinutiaException me) {
    throw new AlgParserException("Error setting direction at minutia
                                " + id + ": " + direction + "\n\t"
                                + me.getMessage());
}

// Controlling separator
if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep.equalsIgnoreCase(lineTokenizer.nextTokn()))
        throw new AlgParserException("Missing separator " +
                                     this.sep + " at minutia " +
                                     id + ".");
}

```

```

    }
    else
        throw new AlgParserException("Unexpected line termination at
                                     minutia " + id + ".");

    // Retrieving minutia reliability
    float reliability= 0;
    if (lineTokenizer.hasMoreTokens()) {
        Float reliabilityTemp= null;
        String tokenString= null;
        try {
            tokenString= lineTokenizer.nextTokent();
            reliabilityTemp= new Float(tokenString);
        }
        catch (NumberFormatException nfe) {
            throw new AlgParserException("Invalid number format (" +
                                         tokenString + ") at minutia
                                         " + id + ".");
        }
        reliability= reliabilityTemp.floatValue();
    }
    else
        throw new AlgParserException("Unexpected line termination at
                                     minutia " + id + ".");

    // Inserting reliability in minutia field
    try {
        min.setReliability(reliability);
    }
    catch (MinutiaException me) {
        throw new AlgParserException("Error setting reliability at
                                     minutia " + id + ": " +
                                     reliability + "\n\t" +
                                     me.getMessage());
    }

    // Controlling separator
    if (lineTokenizer.hasMoreTokens()) {
        if (! this.sep.equalsIgnoreCase(lineTokenizer.nextTokent()))
            throw new AlgParserException("Missing separator " +
                                         this.sep + " at minutia " +
                                         id + ".");
    }
    else
        throw new AlgParserException("Unexpected line termination at
                                     minutia " + id + ".");

    // Retrieving minutia type
    AlgMinutiaType minTypeTemp= null;
    if (lineTokenizer.hasMoreTokens()) {
        String typeString= lineTokenizer.nextTokent();
        try {
            minTypeTemp= new AlgMinutiaType(typeString);
        }
        catch (MinTypeException mte) {
            throw new AlgParserException("Error on minutia " + id +
                                         " setting minutia type: " +
                                         typeString + "\n\t" +
                                         mte.getMessage());
        }
    }

```

```

    }
    else
        throw new AlgParserException("Unexpected line termination at
                                     minutia " + id + ".");

    // Inserting type in minutia field
    min.setMinType(minTypeTemp);

    // Neighbors parsing: continues until there are no more
    // neighbors to parse
    int nNeigh= 0;
    if (lineTokenizer.hasMoreTokens()) {
        try {
            AlgNeighborParser anp= new AlgNeighborParser
                                   (lineTokenizer, min);

            do {
                nNeigh++;
                if (! this.sep.equalsIgnoreCase
                    (lineTokenizer.nextToken()))
                    throw new AlgParserException("Missing separator
                                                    " + this.sep + " at
                                                    minutia " + id +
                                                    ".");

                anp.parse();
            } while (lineTokenizer.hasMoreTokens());
        }
        catch (AlgParserException ape) {
            throw new AlgParserException("Error parsing " +
                                         countLine + "line at
                                         neighbor in position " +
                                         nNeigh + ": \n\t" +
                                         ape.getMessage());
        }
    }
}
catch (NoSuchElementException nsee) {
    throw new AlgParserException("Unexpected no more token error:
                                  \n\t" + nsee.getMessage());
}

// Inserting created minutia in the list
list.add(min);

// Reading another line
try {
    lineString= br.readLine();
}
catch (IOException ioe) {
    throw new AlgParserException("Error reading " + countLine + "
                                  line");
}

} // end of while
} // end of method

} // end of class

```

```

/*
 * AlgParserException.java
 *
 * Created on 15 ottobre 2002, 16.13
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package algParsing;

```

```

/**
 * Exception thrown by <code>AlgNeighborParser</code> and
 * <code>AlgParser</code>
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class AlgParserException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>AlgParserException</code> without detail
     * message.
     */
    public AlgParserException() {
    }

    /**
     * Constructs an instance of <code>AlgParserException</code> with the
     * specified
     * detail message.
     *
     * @param msg the detail message.
     */
    public AlgParserException(String msg) {
        super(msg);
    }
}

```

C.3 Package matching.

C.3.1 Package graph.

```
/*
 * BipartiteGraph.java
 *
 * Created on 28 ottobre 2002, 17.06
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.graph;

/***** IMPORTS *****/
import java.util.HashMap;

/**
 * This class represents a directed bipartite graph. Each node is intended to
 * represent a minutia in minutia graph case, or a neighbor in neighbor
 * assigning
 * phase.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class BipartiteGraph {

    /**
     * Total number of nodes in this bipartite graph: first set cardinality plus
     * second set cardinality.
     */
    private int nodeNumber= 0;

    /**
     * First node set cardinality.
     */
    private int nNumberFirstSet= 0;

    /**
```



```

    * Second node set cardinality.
    */
private int nNumberSecondSet= 0;

/**
 * Data structure to collect forward star for all node in this bipartite
 * graph;
 */
private HashMap fstars= null;

/**
 * Load factor for HashMap data structure.
 */
private static final float loadFactor= (float) 0.75;

/**
 * Creates a new instance of Graph.
 *
 * @param card1 Carinality of first node set.
 * @param card2 Carinality of second node set.
 *
 * @throws BipartiteGraphException If fail to create the graph (e.g. a
 *                                  cardinality below zero)
 */
public BipartiteGraph(int card1, int card2) throws BipartiteGraphException {
    if ((card1 > 0) && (card2 > 0) /*&& (card1 <= card2)*/) {
        this.nNumberFirstSet= card1;
        this.nNumberSecondSet= card2;
        this.nodeNumber= card1 + card2;
    }
    else
        throw new BipartiteGraphException("Number of node in the bipartite
                                           graph not admitted (" + card1 + ",
                                           " + card2 + ")");

    try {
        this.fstars= new HashMap(this.nodeNumber, loadFactor);
    }
    catch (IllegalArgumentException iae) {
        throw new BipartiteGraphException("Unexpected error creating forward
                                           star Vector: \n\t" +
                                           iae.getMessage());
    }
}

/**
 * Returns first node set cardinality.
 *
 * @return First set cardinality.
 */
public int getFirstSetNodeNumber() {
    return (this.nNumberFirstSet);
}

/**
 * Returns second node set cardinality.

```

```

*
* @return Second set cardinality.
*/
public int getSecondSetNodeNumber() {

    return (this.nNumberSecondSet);

}

/**
 * Add a forward star of a node in this graph.
 *
 * @param sourceNodeKey Source node identifier of the forward star.
 * @param fs Forward star to insert in the graph.
 *
 * @throws BipartiteGraphException If fails to insert forward star (e.g.)
 */
public void addForwardStar(int sourceNodeKey, ForwardStar fs) throws
    BipartiteGraphException {

    if ((sourceNodeKey < 0) || (sourceNodeKey >= this.nodeNumber))
        throw new BipartiteGraphException("Node not in the graph (" +
            sourceNodeKey + ")");

    Integer key= new Integer(sourceNodeKey);
    Object returnObject= this.fstars.put(key, fs);
    if (returnObject != null)
        throw new BipartiteGraphException("Forward star of node " +
            sourceNodeKey + " already in
            this graph");

}

/**
 * Returns a node forward star.
 *
 * @param sourceNodeKey Node id identifying the source of returned forward
 * star.
 * @return ForwardStar of the node.
 *
 * @throws BipartiteGraphException If fails to return fs (e.g. passed node
 * not present in this graph).
 */
public ForwardStar getForwardStar(int sourceNodeKey) throws
    BipartiteGraphException {

    if ((sourceNodeKey < 0) || (sourceNodeKey >= this.nodeNumber))
        throw new BipartiteGraphException("Forward star not in the graph (fs
            " + sourceNodeKey + ")");

    Integer key= new Integer(sourceNodeKey);
    ForwardStar returnForwardStar= (ForwardStar) (this.fstars.get(key));
    if (returnForwardStar == null)
        throw new BipartiteGraphException("Null or no forward star in the
            graph (fs " + sourceNodeKey +
            ")");

    return(returnForwardStar);

}

/**
 * Returns number of node in this bipartite graph.
 *
 * @return Number of node in this bipartite graph.

```

```

    */
    public int getNodeNumber() {
        return (this.nodeNumber);
    }

    /**
     * Add a node with identifier equal to number of node in the graph.
     * Needed to insert super-root to create auxiliary graph.
     *
     * @return Identifier of inserted node.
     */
    public int addNode() {
        this.nodeNumber++;
        return (this.nodeNumber - 1);
    }

    /**
     * Removes the node in last position.
     */
    public void removeNode() {
        this.nodeNumber--;
    }

    /**
     * Prints the graph.
     */
    public void print() {
        int size1= this.fstars.size();
        System.out.println("");
        Integer key= null;
        for (int j= 0; j < size1; j++) {
            key= new Integer(j);
            ForwardStar fs= (ForwardStar) (this.fstars.get(key));
            if (fs == null)
                System.out.println("Null or no forward star in position " + j +
                                   " in the graph");
            int size2= fs.getSize();
            for (int i= 0; i < size2; i++) {
                Edge e= (Edge) (fs.getEdgeIterator().next());
                System.out.print("Source " + j + " - Dest " + e.getDest());
                EdgeCost ec= e.getCost();
                System.out.println(" - Edge cost " + ec.getEuclideanDistance() +
                                   " " + ec.getDistanceRelativeAngle() + " " +
                                   ec.getRelativeAngle());
                System.out.println(" - Edge cost sum = " + ec.getSum());
            }
            if (size2 > 0)
                System.out.println("");
        }
        System.out.println("");
    }
}

/*

```

```

* BipartiteGraphException.java
*
* Created on 28 ottobre 2002, 17.14
*
* Copyright 2003 Francesco Santini
*
* This file is part of FinMatch.
*
* FinMatch is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* FinMatch is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with FinMatch; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

```

```
package matching.graph;
```

```

/**
 * Exception thrown by methods in class <code>graph.Graph</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class BipartiteGraphException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>GraphException</code> without detail
     * message.
     */
    public BipartiteGraphException() {
    }

    /**
     * Constructs an instance of <code>GraphException</code> with the specified
     * detail message.
     *
     * @param msg the detail message.
     */
    public BipartiteGraphException(String msg) {
        super(msg);
    }
}

```

```

/*
 * Edge.java
 *
 * Created on 28 ottobre 2002, 17.22
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.graph;

/**
 * Class representing edge in graph. Destination id is an identifier of
 * destination node and cost is its cost, so edges have direction and weights.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class Edge {

    /**
     * Source node id.
     */
    private int source= -1;

    /**
     * Destination node id.
     */
    private int dest= -1;

    /**
     * Edge cost.
     */
    private EdgeCost cost= null;

    /**
     * Creates a new instance of Edge.
     *
     * @param source Source node id.
     * @param destination Destination node id.
     * @param cost This edge cost.
     *
     * @throws EdgeException Null cost or destination id below zero.
     */
    public Edge(int source, int destination, EdgeCost cost)throws EdgeException{
        if (destination < 0)

```

```

        throw new EdgeException("Impossible edge destination: (" +
                                destination + ")");
    if (source < 0)
        throw new EdgeException("Impossible edge source: (" + source + ")");
    if (cost == null)
        throw new EdgeException("Null edge cost.");

    this.cost= cost;
    this.source= source;
    this.dest= destination;
}

/**
 * Sets destination node of this edge
 *
 * @param destination Destination node id for this edge.
 *
 * @throws EdgeException If fail to set destination node (e.g. below zero.)
 */
public void setDest(int destination) throws EdgeException {

    if (destination < 0)
        throw new EdgeException("Impossible edge destination: (" +
                                destination + ")");

    this.dest= destination;
}

/**
 * Sets source node of this edge
 *
 * @param source Source node id for this edge.
 *
 * @throws EdgeException If fail to set source node (e.g. below zero.)
 */
public void setSource(int source) throws EdgeException {

    if (source < 0)
        throw new EdgeException("Impossible edge source: (" + source + ")");
    this.source= source;
}

/**
 * Returns destination node of this edge.
 *
 * @return Destination node id.
 */
public int getDest() {

    return(this.dest);
}

/**
 * Returns source node of this edge.

```

```

    *
    * @return Source node id.
    */
    public int getSource() {
        return(this.source);
    }

    /**
     * Returns cost of this edge.
     *
     * @return This edge cost.
     */
    public EdgeCost getCost() {
        return(this.cost);
    }
}

```

```

/*
 * EdgeCost.java
 *
 * Created on 29 ottobre 2002, 23.35
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package matching.graph;
```

```

/**
 * Represents cost for edges in graph.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class EdgeCost {

    /**
     * Nedded to correct machine errors avoidin false megative cycles in SPT
     * algorithm.
     */
    private static final double epsilon= 4.94065645841246544e-324;

    /**
     * Euclidean distance of edge cost.
     */
    private float euclideanDistance= 0;

    /**
     * Distance-relative angle component.
     */
    private float distanceRelativeAngle= 0;

    /**
     * Relative angle between minutiae component.
     */
    private float relativeAngle= 0;

    /**
     * Ridge count difference between neighbors.
     */
    private float ridgeCountDistance= 0;

    /**
     * Sum of distances between all minutia neighbors.
     */

```



```

    */
private float neighborDistance= 0;

/**
 * Weight for euclideanDistance.
 */
private static float euclideanDistanceWeight= 2151;

/**
 * Weight for distanceRelativeAngle.
 */
private static float distanceRelativeAngleWeight= 1800;

/**
 * Weight for relativeAngle.
 */
private static float relativeAngleWeight= 922;

/**
 * Weight for ridgeCountDistance.
 */
private static float ridgeCountDistanceWeight= 1743;

/**
 * Weight for neighborDistance
 */
private static float neighborDistanceWeight= 0;

/**
 * Sum of the components.
 */
private double sum= 0;

/**
 * Creates a new instance of EdgeCost.
 *
 * @param ed Euclidean distance component for this edge cost.
 * @param dra Distance-relative angle component for this edge cost.
 * @param ra Relative angle component for this edge cost.
 * @param rc Ridge count difference.
 *
 * @throws EdgeCostException If passed parameters are not valid (e.g. angle
 *                             bigger than 360)
 */
public EdgeCost(float ed, float dra, float ra, float rc, float qf) throws
    EdgeCostException {

    if ((dra < 0) || (dra > 1) || (ra < 0) || (ra > 1) || (ed < 0) || (ed >
        1) || (rc < 0) || (rc > 1) || (qf < 0) || (qf > 1))
        throw new EdgeCostException("Not a valid cost for the edge (" + ed
            + ", " + dra + ", " + ra + ", " + rc +
            ", " + qf + ")");

    this.euclideanDistance= ed;
    this.distanceRelativeAngle= dra;
    this.relativeAngle= ra;
    this.ridgeCountDistance= rc;
    this.neighborDistance= qf;

    this.sum= euclideanDistanceWeight * ed + distanceRelativeAngleWeight *
        dra + relativeAngleWeight * ra + ridgeCountDistanceWeight * rc
        + neighborDistanceWeight * qf;
}

```

```

}

/**
 * Private constructor used by method <code>add</code> of this class.
 *
 * @param ed Euclidean distance component for this edge cost.
 * @param dra Distance-relative angle component for this edge cost.
 * @param ra Relative angle component for this edge cost.
 * @param sum Sum field value.
 */
private EdgeCost(float ed, float dra, float ra, float rc, float qf, double
                sum) {

    this.euclideanDistance= ed;
    this.distanceRelativeAngle= dra;
    this.relativeAngle= ra;
    this.ridgeCountDistance= rc;
    this.neighborDistance= qf;
    this.sum= sum;

}

/**
 * Creates a new instance of EdgeCost copying field values from passed edge
 * cost.
 *
 * @param otherEdgeCost Copys filed values from this edge cost.
 *
 * @throws EdgeCostException If passed parameter is <code>null</code>.
 */
public EdgeCost(EdgeCost otherEdgeCost) throws EdgeCostException {

    if (otherEdgeCost == null)
        throw new EdgeCostException("Null edge cost parameter passed.");
    this.euclideanDistance= otherEdgeCost.getEuclideanDistance();
    this.distanceRelativeAngle= otherEdgeCost.getDistanceRelativeAngle();
    this.relativeAngle= otherEdgeCost.getRelativeAngle();
    this.ridgeCountDistance= otherEdgeCost.getRidgeCountDistance();
    this.neighborDistance= otherEdgeCost.getNeighborDistance();
    this.sum= otherEdgeCost.getSum();

}

/**
 * Creates a new instance of EdgeCost setting its field values to maximum.
 */
public EdgeCost() {

    this.distanceRelativeAngle= 0;
    this.euclideanDistance= 0;
    this.relativeAngle= 0;
    this.ridgeCountDistance= 0;
    this.neighborDistance= 0;
    this.sum= 0;

}

/**
 * Sets weight in the edge for euclidean distance.

```

```

*
* @param weight New euclidean distance weight.
*/
public static void setEuclideanDistanceWeight(float weight) {
    euclideanDistanceWeight= weight;
}

/**
 * Sets weight in the edge for distance relative angle.
 *
 * @param weight New distance relative angle weight.
 */
public static void setDistanceRelativeAngleWeight(float weight) {
    distanceRelativeAngleWeight= weight;
}

/**
 * Sets weight in the edge for relative angle.
 *
 * @param weight New relative angle weight.
 */
public static void setRelativeAngleWeight(float weight) {
    relativeAngleWeight= weight;
}

/**
 * Sets weight in the edge for ridge count.
 *
 * @param weight New ridge count weight.
 */
public static void setRidgeCountDistanceWeight(float weight) {
    ridgeCountDistanceWeight= weight;
}

/**
 * Sets weight in the edge for neighbor distance.
 *
 * @param weight New neighbor distance weight.
 */
public static void setNeighborDistanceWeight(float weight) {
    neighborDistanceWeight= weight;
}

/**
 * Returns euclidean distance field.
 *
 * @return Euclidean distance component of this edge cost.
 */

```

```

public float getEuclideanDistance() {
    return (this.euclideanDistance);
}

/**
 * Returns distance-relative angle field.
 *
 * @return Distance-relative angle component of this edge cost.
 */
public float getDistanceRelativeAngle() {
    return (this.distanceRelativeAngle);
}

/**
 * Returns relative angle field.
 *
 * @return Relative angle component of this edge cost.
 */
public float getRelativeAngle() {
    return (this.relativeAngle);
}

/**
 * Returns ridge count distance field.
 *
 * @return Ridge count difference field.
 */
public float getRidgeCountDistance() {
    return (this.ridgeCountDistance);
}

/**
 * Returns neighbor distance field
 *
 * @return sum of distances between all minutia neighbors.
 */
public float getNeighborDistance() {
    return (this.neighborDistance);
}

/**
 * Returns sum of three component.
 *
 * @return Sum of this edge cost three component.
 */
public double getSum() {
    return (this.sum);
}

```

```

}

/**
 * Sets sum filed for this edge cost.
 *
 * @param newSum New sum filed value.
 */
private void setSum(double newSum) {
    this.sum= newSum;
}

/**
 * Sets three component to a maximum and calulates their sum.
 */
public void costLimit() {
    this.euclideanDistance= 10000;
    this.distanceRelativeAngle= 7200;
    this.relativeAngle= 7200;
    this.ridgeCountDistance= 10000;
    this.neighborDistance= 10000;
    this.sum= 10000 + 7200 + 7200 + 10000 + 10000;
}

/**
 * Adds corrispondent component of two edge cost and creates a new edge cost
 * with these values.
 *
 * @param otherEdgeCost Addend edge cost.
 * @return Edge cost resultin from aadition.
 */
public EdgeCost add(EdgeCost otherEdgeCost) {
    float temp1= this.euclideanDistance +
        otherEdgeCost.getEuclideanDistance();
    float temp2= this.distanceRelativeAngle +
        otherEdgeCost.getDistanceRelativeAngle();
    float temp3= this.relativeAngle + otherEdgeCost.getRelativeAngle();
    float temp4= this.ridgeCountDistance +
        otherEdgeCost.getRidgeCountDistance();
    double temp5= this.sum + otherEdgeCost.getSum();
    float temp6= this.neighborDistance +
        otherEdgeCost.getNeighborDistance();
    EdgeCost returnEdgeCost= new EdgeCost(temp1, temp2, temp3, temp4, temp6,
        temp5);
    return (returnEdgeCost);
}

/**
 * Subtracts corrispondent component of two edge cost and creates a new edge
 * cost with these values.
 *
 * @param otherEdgeCost Subtractor edge cost.
 * @return Edge cost resultin from subtraction.
 */

```

```

public void sub (EdgeCost otherEdgeCost) {

    this.distanceRelativeAngle-= otherEdgeCost.getDistanceRelativeAngle();
    this.euclideanDistance-= otherEdgeCost.getEuclideanDistance();
    this.relativeAngle-= otherEdgeCost.getRelativeAngle();
    this.ridgeCountDistance-= otherEdgeCost.getRidgeCountDistance();
    this.neighborDistance-= otherEdgeCost.getNeighborDistance();
    this.sum-= otherEdgeCost.getSum();

}

/**
 * Returns true if this edge cost is smaller than the passed one.
 *
 * @param otherEdgeCost Other edge cost for comparison.
 * @return <code>True</code> if this edge cost is smaller, false otherwise.
 */
public boolean isSmaller(EdgeCost otherEdgeCost) {

    if (this.sum < (otherEdgeCost.getSum() - epsilon))
        return (true);
    else
        return (false);
}

/**
 * Returns <code>true</code> if this edge cost is greater than the passed
 * one.
 *
 * @param otherEdgeCost Other edge cost for comparison.
 * @return True if this edge cost is greater, false otherwise.
 */
public boolean isGreater(EdgeCost ec) {

    if (this.sum > ec.getSum())
        return (true);
    else
        return (false);
}

/**
 * Inverts this cost: used in minimum cost assignment.
 */
public void invertCost() {

    this.sum= -this.sum;
}

/**
 * This method calculates the division between edge cost and number of
 * edges of assignmnet graph.
 *
 * @param numberOfEdges number of edges in assignmnet.
 *
 * @throws EdgeCostException Raised if <code>numberOfEdges</code> is less
 * than 0.
 */

```

```

public void mean (int numberOfEdges) throws EdgeCostException {
    if (numberOfEdges <= 0)
        throw new EdgeCostException ("Divisor can't be equal or less than
                                      0.");
    this.euclideanDistance= (float)(this.euclideanDistance / numberOfEdges);
    this.distanceRelativeAngle= (float) (this.distanceRelativeAngle /
                                         numberOfEdges);
    this.relativeAngle= (float) (this.relativeAngle / numberOfEdges);
    this.ridgeCountDistance= (float)(this.ridgeCountDistance /
                                     numberOfEdges);
    this.sum= (float) (this.sum / numberOfEdges);
}
}

```

```

/*
 * EdgeCostException.java
 *
 * Created on 29 ottobre 2002, 23.42
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package matching.graph;

```

```

/**
 * Exception throwed by methods in class <code>graph.EdgeCost</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class EdgeCostException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>EdgeCostException</code> without detail
     * message.
     */
    public EdgeCostException() {
    }

    /**
     * Constructs an instance of <code>EdgeCostException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public EdgeCostException(String msg) {
        super(msg);
    }
}

```



```

/*
 * EdgeException.java
 *
 * Created on 28 ottobre 2002, 17.31
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package matching.graph;
```

```

/**
 * Exception thrown by methods in class <code>graph.Edge</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class EdgeException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>EdgeException</code> without detail
     * message.
     */
    public EdgeException() {
    }

    /**
     * Constructs an instance of <code>EdgeException</code> with the specified
     * detail message.
     *
     * @param msg the detail message.
     */
    public EdgeException(String msg) {
        super(msg);
    }
}

```

```

/*
 * ForwardStar.java
 *
 * Created on 28 ottobre 2002, 17.32
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package matching.graph;
```

```

/***** IMPORTS *****/
import java.util.HashMap;
import java.util.Collection;
import java.util.Iterator;

```

```

/**
 * This class represents forward star for a node in oriented graph. So contains
 * all the nodes directed from a node to other nodes.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class ForwardStar {
```

```

    /**
     * Data structure to collect nodes in this forward star.
     */

```

```
    private HashMap hm= null;
```

```

    /**
     * Load factor for HashMap data structure.
     */

```

```
    private static final float loadFactor= (float) 0.75;
```

```

    /**
     * Index for this source node of this forward star.
     */

```

```
    private int sourceNode= -1;
```

```

    /**
     * Creates a new instance of ForwardStar
     */

```

```

    * @param initialCapacity Initial capacity of <code>Hash Map</code> used to
    * collect edges.

```

```

    * @param sourceNode Source node index of this forward star.

```

```

*
* @throws ForwardStarException If fail to create forward star (e.g. invalid
*                               source node index).
*/
public ForwardStar(int initialCapacity, int sourceNode) throws
    ForwardStarException {

    if (sourceNode < 0)
        throw new ForwardStarException("Invalid source node index: " +
                                         sourceNode);

    try {
        this.hm= new HashMap(initialCapacity, loadFactor);
    }
    catch (IllegalArgumentException iae) {
        throw new ForwardStarException("Error creating Hash Map: \n\t" +
                                         iae.getMessage());
    }
    this.sourceNode= sourceNode;
}

/**
 * Inserts an edge in this forward star.
 *
 * @param e Edge to insert.
 *
 * @throws ForwardStarException If fail to insert the edge (e.g. edge
 *                               already in this forward star).
 */
public void insertEdge(Edge e) throws ForwardStarException {

    Integer key= new Integer(e.getDest());
    Object returnObject= this.hm.put(key, e);
    if (returnObject != null)
        throw new ForwardStarException("Destination node already present in
                                         this FS (" + e.getDest() + ")");
}

/**
 * Deletes an edge in this forward star.
 *
 * @param delIndex Destinatio node index of the edge to delete.
 *
 * @throws ForwardStarException If fail to delete the edge (e.g. edge not
 *                               in this forward star).
 */
public void deleteEdgeByDest(int delIndex) throws ForwardStarException {

    Integer key= new Integer(delIndex);
    Object returnObject= hm.remove(key);
    if (returnObject == null)
        throw new ForwardStarException("Null or no destination Node found in
                                         this FS (" + delIndex + ")");
}

/**
 * Return <code> true</code> if edge is in this forward star.
 *

```

```

    * @param searchIndex Destination node index of the edge to search in this
    *                      fs.
    * @return <code>True</code> if edge is in this forward star, false
    *         otherwise.
    */
public boolean isInByDest(int searchIndex) {

    Integer key= new Integer(searchIndex);
    if (this.hm.containsKey(key))
        return (true);
    else
        return (false);
}

/**
 * Deletes an edge in this forward star.
 *
 * @param e Edge to delete.
 *
 * @throws ForwardStarException If fail to delete the edge (e.g. edge not
 *                               in this forward star).
 */
public void deleteEdge(Edge e) throws ForwardStarException {

    Integer key= new Integer(e.getDest());
    Object returnObject = this.hm.remove(key);
    if (returnObject == null)
        throw new ForwardStarException("Null or no such node in this forward
                                         star");
}

/**
 * Returns <code>Iterator</code> object to scan this forward star node list.
 *
 * @return <code>Iterator</code> to scan the nodes as a list.
 */
public Iterator getEdgeIterator() {

    Collection collectionView= this.hm.values();
    Iterator returnIterator= collectionView.iterator();
    return (returnIterator);
}

/**
 * Searches for edge in this forward star starting from destination node
 * index of the edge.
 *
 * @param destIndex Destination node index of the edge to search.
 * @return Founded edge.
 *
 * @throws ForwardStarException If fails to find the edge (e.g. edge is not
 *                               inside this forward star).
 */
public Edge getEdgeByDest(int destIndex) throws ForwardStarException {

    Integer key= new Integer(destIndex);
    Object returnObject = this.hm.get(key);
    if (returnObject == null)

```

```

        throw new ForwardStarException("Null or no such node in this forward
                                         star (destination node " + destIndex
                                         + ") ");
    else
        return ((Edge) (returnObject));
}

/**
 * Returns number of edges in this forward star.
 *
 * @return Number of edges collected in this forward star.
 */
public int getSize() {
    return (this.hm.size());
}
}

```

```

/*
 * ForwardStarException.java
 *
 * Created on 31 ottobre 2002, 19.15
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.graph;

/**
 * Exception thrown by methods in class <code>graph.ForwardStar</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class ForwardStarException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>ForwardStarException</code> without
     * detail message.
     */
    public ForwardStarException() {
    }

    /**
     * Constructs an instance of <code>ForwardStarException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public ForwardStarException(String msg) {
        super(msg);
    }
}

```

C.3.2 Package minCostAssigning.

```
/*
 * Assigner.java
 *
 * Created on 30 ottobre 2002, 0.46
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.minCostAssigning;

/***** IMPORTS *****/
import matching.graph.*;
import matching.shortestPathTree.*;

/**
 * This class implements assignment algorithm.<p>
 * Makes auxiliary graph, runs SPT algorithm over it and invokes a change
 * assignment procedure passing found alternating-augmenting path. Assignment is
 * the maximum cardinality minimum cost one.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class Assigner {

    /**
     * Creates a new instance of Assigner
     */
    public Assigner() {

    }

    /**
     * Assigner method. Found a minimum cost assignment on bipartite passed
     * graph.
     *
     * @param graph Method <code>assign</code> finds a maximum cardinality
     * minimum cost assignemnt on this graph.
     * @return Data structure representing the found assignment.
     */
}
```

```

*
* @throws AssignerException If fails to find an assignment (e.g. error
*                               manipulating the graph)
*/
public MatchSet assign(BipartiteGraph graph) throws AssignerException {

    int firstSetSize= graph.getFirstSetNodeNumber();
    int secondSetSize= graph.getSecondSetNodeNumber();

    // Adding super-root to make auxiliary graph
    int newStartNode= graph.addNode();

    // Creating super-root forward star
    ForwardStar startNodeForwardStar= null;
    try {
        startNodeForwardStar= new ForwardStar(graph.getNodeNumber(),
                                              firstSetSize);
    }
    catch (ForwardStarException fse) {
        throw new AssignerException("Error creating super-root forward star:
                                   \n\t" + fse.getMessage());
    }

    // Creating cost for super-root forward star edges: set to zero.
    EdgeCost startNodeEdgeCost= new EdgeCost();

    // Inserting edges in super-root forward star: one edge for each first
    // set nodes.
    for (int i= 0; i < firstSetSize; i++) {
        Edge startNodeEdge= null;
        try {
            startNodeEdge= new Edge(newStartNode, i, startNodeEdgeCost);
        }
        catch (EdgeException ee) {
            throw new AssignerException("Error creating super-root edge with
                                       destination " + i + ": \n\t" +
                                       ee.getMessage());
        }

        try {
            startNodeForwardStar.insertEdge(startNodeEdge);
        }
        catch (ForwardStarException fse) {
            throw new AssignerException("Error inserting edge " + i + " in
                                       super-root forward star : \n\t" +
                                       fse.getMessage());
        }
    }
    // End of inserting edges in super-root forward star

    // Inserting super-root forward star in graph obtaining auxiliary graph
    try {
        graph.addForwardStar(newStartNode, startNodeForwardStar);
    }
    catch (BipartiteGraphException bge) {
        throw new AssignerException("Error inserting super-root forward star
                                   in bipartite graph : \n\t" +
                                   bge.getMessage());
    }

    // New number of node in graph after adding super-root
    int nodeNumber= newStartNode + 1;

    // Creating data structure to represent the assignment

```



```

MatchSet ms= null;
try {
    ms= new MatchSet(firstSetSize, secondSetSize);
}
catch (MatchSetException mse) {
    throw new AssignerException("Error creating data structure to
                                represent the assignment: \n\t" +
                                mse.getMessage());
}

// Number of loop iteration (to stop infinite loop caused by errors)
int i= 0;
boolean cond= true;

// Assignment loop
while (cond) {
    i++;
    if (i > firstSetSize) {
        throw new AssignerException("Not Expected error: too much
                                    iterations: " + i);
    }

    // Creating Shortest Path Tree Finder
    SPT spt= new SPT();

    // Parent vector representing shortest path tree
    int p[] = new int[nodeNumber];
    int newMatchedNode= 0;

    // Calculating SPT over auxiliary graph
    try {
        newMatchedNode= spt.calc(graph, newStartNode, p, ms);
    }
    catch (SPTException spte) {
        throw new AssignerException("Error calculating SPT at iteration
                                    " + i + ": \n\t" +
                                    spte.getMessage());
    }

    // Checking if has been found alternating-augmenting path
    // destination node
    if (newMatchedNode == -1)

        // Exiting assignment loop
        break;
    else

        // Changing current assignment using alternating-augmenting path
        // found by SPT
        try {
            ms.changeMatch(graph, p, newStartNode, newMatchedNode);
        }
        catch (MatchSetException mse) {
            throw new AssignerException("Error changing the assignment:
                                        \n\t" + mse.getMessage());
        }

    // Exiting from assigning cycle if one of two set is completely
    // assigned
    if ((ms.getPairsNumber() == ms.getFirstSetCard()) ||
        (ms.getPairsNumber() == ms.getSecondSetCard()))

        // Exiting assignment loop

```

```

        break;
    } // End of assignment loop

    // Controlling if all first set nodes are matched. In this software
    // configuration may not needed: throws exception for nodes with no arc
    // (low reliability node)
    /*if (ms.getPairsNumber() != ms.getFirstSetCard())
        throw new AssignerException("Matched only " + ms.getPairsNumber() +
                                     "pairs");*/

    // Returning found maximum cardinality minimum cost assignment
    return (ms);
}
}

```

```

/*
 * AssignerException.java
 *
 * Created on 30 ottobre 2002, 2.10
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package matching.minCostAssigning;

```

```

/**
 * Thrown by methods in <code>minCostAssigning.Assigner</code> class.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class AssignerException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>AssignerException</code> without detail
     * message.
     */
    public AssignerException() {
    }

    /**
     * Constructs an instance of <code>AssignerException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public AssignerException(String msg) {
        super(msg);
    }
}

```

```

/*
 * MatchSet.java
 *
 * Created on 30 ottobre 2002, 2.23
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package matching.minCostAssigning;

```

```

/***** IMPORTS *****/
import matching.graph.*;

```

```

/**
 * Class representing assignment data structure.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MatchSet {

    /**
     * First node set cardinality.
     */
    private int firstSetCard= 0;

    /**
     * Second node set cardinality.
     */
    private int secondSetCard= 0;

    /**
     * Flag representing tha a node is not assigned to another.
     */
    private static final int notMatched= -1;

    /**
     * Number of node pairs assigned.
     */
    private int pairsNumber= 0;

    /**
     * Assignment cost (total of edge costs in assignment).
     */

```

```

private EdgeCost matchCost= null;

/**
 * Vector containing second set node identifiers, assigned to first node
 * set: matchFirstSet[h] = k means tha node h in first
 * set is assigned to node k in second set.
 */
private int[] matchFirstSet;

/**
 * Vector indicating if a node in second set is assigned: true
 * is assigned, false is not assigned:
 * matchedSecondSet[k] = true means that node k in
 * second set is assigned with one node in first set.
 */
private boolean[] matchedSecondSet;

/**
 * Costs of edges making the assignment: field matchCost is the
 * sum of this edge costs.
 */
private EdgeCost[] matchEdgeCost;

/**
 * Creates a new instance of MatchSet.
 *
 * @param firstSetLength First node set cardinality.
 * @param secondSetLength Second node set cardinality.
 *
 * @throws MatchSetException If fails to create assignment data structure
 * (e.graph. bad set carinality passed parameter).
 */
public MatchSet(int firstSetLength, int secondSetLength) throws
    MatchSetException {

    if ((firstSetLength) <= 0 || (secondSetLength <= 0))
        throw new MatchSetException("Impossible to match these two sets (" +
            firstSetLength + ", " + secondSetLength
            + ")");

    // Initializing data structure.
    this.firstSetCard= firstSetLength;
    this.secondSetCard= secondSetLength;
    this.matchFirstSet= new int[firstSetLength];
    this.matchEdgeCost= new EdgeCost[firstSetLength];
    this.matchedSecondSet= new boolean[secondSetLength];
    int size2= this.matchedSecondSet.length;

    // Initializing matchedSecondSet vector
    for (int i= 0; i < size2; i++)
        this.matchedSecondSet[i]= false;
    int size1= this.matchFirstSet.length;

    // Initializing matchFirstSet vector
    for (int i= 0; i < size1; i++)
        this.matchFirstSet[i]= this.notMatched;

    this.matchCost= new EdgeCost();

```

```

}

/**
 * Returns <code>matchFirstSet</code> vector representing the assignment.
 *
 * @return Representation of the assignment.
 */
public int[] getMatch() {
    return (this.matchFirstSet);
}

/**
 * Returns vector containing costs of the edges making the assignment.
 *
 * @return <code>matchEdgeCost</code> assignment cost vector.
 */
public EdgeCost[] getMatchCost() {
    return (this.matchEdgeCost);
}

/**
 * Returns assignment cost.
 *
 * @return Cost of the assignment.
 */
public EdgeCost getMatchEdgeCost() {
    return (this.matchCost);
}

/**
 * Return first set cardinality.
 *
 * @return First set cardinality.
 */
public int getFirstSetCard() {
    return(this.firstSetCard);
}

/**
 * Return second set cardinality.
 *
 * @return Second set cardinality.
 */
public int getSecondSetCard() {
    return(this.secondSetCard);
}

```

```

/**
 * Returns number of node pairs assigned.
 *
 * @return Node pairs assigned number.
 */
public int getPairsNumber() {
    return (this.pairsNumber);
}

/**
 * Returns <code>true</code> if <code>secondSetIndex</code> node in second
 * set is assigned, <code>false</code> it is not assigned or if it's a first
 * set node identifier.
 *
 * @param secondSetIndex Node identifier in second node set.
 * @return <code>True</code> if <code>secondSetIndex</code> node is not
 *         assigned, and in second node set, <code>false</code> otherwise.
 *
 * @throws MatchSetException If fails to check if the node is assigned
 *                             (e.graph node identifier is greater than node
 *                             number).
 */
public boolean isNotMatchedSecondSet(int secondSetIndex) throws
    MatchSetException {
    if (secondSetIndex < this.firstSetCard)
        return (false);
    else {
        int index= secondSetIndex - this.firstSetCard;
        if (index < this.matchedSecondSet.length)
            return (!this.matchedSecondSet[index]);
        else
            throw new MatchSetException("Node out of bound (" +
                secondSetIndex + ")");
    }
}

/**
 * Returns <code>true</code> if <code>firstSetIndex</code> node in second
 * set is assigned, <code>false</code> it is not assigned.
 *
 * @param firstSetIndex Node identifier in first node set.
 * @return <code>True</code> if <code>firstSetIndex</code> node is assigned,
 *         <code>false</code> if it is not assigned.
 *
 * @throws MatchSetException If fails to check if the node is assigned
 *                             (e.graph node identifier is not in first set
 *                             bounds).
 */
public boolean isNotMatchedFirstSet(int firstSetIndex) throws
    MatchSetException {
    if ((firstSetIndex < 0) || (firstSetIndex >= this.firstSetCard))
        throw new MatchSetException("Node identifier " + firstSetIndex + "
            not in first set");
    else {
        if (matchFirstSet[firstSetIndex] == this.notMatched)
            return(true);
        else

```

```

        return(false);
    }
}

/**
 * Changes the assignment using alternating-augmenting path found by SPT.
 *
 * @param graph Auxiliary graph.
 * @param p Parent vector representin shortest path tree found by SPT
 *      procedure.
 * @param root Identifier of root node in shortest path tree.
 * @param dest Destination node of the alternating augmenting path.
 *
 * @throws MatchSetException If fails to change the assignment (e.g. not a
 *      valid augmenting path).
 */
public void changeMatch(BipartiteGraph graph, int[] p, int root, int dest)
    throws MatchSetException {

    // Controlling input parameters
    if ((dest < this.firstSetCard) || (dest >= (this.firstSetCard +
        this.secondSetCard)))
        throw new MatchSetException ("Error on the destination node (" +
            dest + ")");

    if (graph == null)
        throw new MatchSetException ("Not expected error: graph parameter is
            null.");

    if (p == null)
        throw new MatchSetException ("Not expected error: parent vector
            parameter is null.");

    if (p.length < graph.getNodeNumber())
        throw new MatchSetException ("Not a valid parent vector length (" +
            p.length + ")");

    int tempIndex= dest;

    // Flag to remember if processing edge of A-A path is internal or
    // external
    int oddEven= 1;
    ForwardStar forwardStarTemp= null;
    Edge e= null;

    // Getting root node forward star
    ForwardStar rootForwardStar= null;
    try {
        rootForwardStar= graph.getForwardStar(root);
    }
    catch (BipartiteGraphException bge) {
        throw new MatchSetException ("Error searching root fs (" + root +
            "): \n\t" + bge.getMessage());
    }

    // Scanning A-A path edges
    while (p[tempIndex] != root) {

        // External edge in A-A path case
        if ((oddEven % 2) != 0) {

            // Transforming tempIndex in corrispondent matchSecondSet vector

```



```

// index
int ti= tempIndex - this.firstSetCard;

// TempIndex is now matched

this.matchedSecondSet[ti]= true;

// Getting parent identifier
int parent= p[tempIndex];

// Parent (in first set) is now matched with tempIndex
this.matchFirstSet[parent]= tempIndex;

// Incrementing pairs matched
this.pairsNumber++;

    // Getting parent forward star
    try {
        forwardStarTemp= graph.getForwardStar(parent);
    }
    catch (BipartiteGraphException bge) {
        throw new MatchSetException("Error searching getting node "
                                     + parent + "-" + tempIndex + "
                                     forward star: " +
                                     bge.getMessage());
    }

    // Getting external edge in foward star
    try {
        e= forwardStarTemp.getEdgeByDest(tempIndex);
    }
    catch (ForwardStarException fse) {
        throw new MatchSetException("Error while searching the edge
                                     " + parent + "-" + tempIndex +
                                     ": " + fse.getMessage());
    }

    // Inserting egde cost in assignment cost vector
    try {
        this.matchEdgeCost[parent]= new EdgeCost(e.getCost());
    }
    catch (EdgeCostException ece) {
        throw new MatchSetException("Error inserting edge cost in
                                     assignment cost vector (node " +
                                     parent + " ): \n\t" +
                                     ece.getMessage());
    }

    // Adding edge cost to total assignment cost
    this.matchCost= this.matchCost.add(this.matchEdgeCost[parent]);

    // Removing this edge from forward star
    try {
        forwardStarTemp.deleteEdge(e);
    }
    catch (ForwardStarException fse) {
        throw new MatchSetException("Error while removing edge " +
                                     parent + "-" + tempIndex + "
                                     from forward star: \n\t" +
                                     fse.getMessage());
    }

    // Inverting edge cost sign.

```

```

e.getCost().invertCost();

// Getting edge destination and source nodes
int destinationNode= e.getDest();
int sourceNode= e.getSource();

if ((tempIndex != destinationNode) || (parent != sourceNode))
    throw new MatchSetException("Unexpected error on edge source
                                or destination node");

// Inverting edge direction
try {
    e.setDest(sourceNode);
}
catch (EdgeException ee) {
    throw new MatchSetException("Error while inverting the edge
                                " + sourceNode + "-" +
                                destinationNode + "(destination
                                error): \n\t" + ee.getMessage());
}

try {
    e.setSource(destinationNode);
}
catch (EdgeException ee) {
    throw new MatchSetException("Error while inverting the edge
                                " + sourceNode + "-" +
                                destinationNode + "(source
                                error): \n\t" + ee.getMessage());
}

// Searching destination node forward star
ForwardStar newEdgeForwardStar= null;
try {
    newEdgeForwardStar= graph.getForwardStar(tempIndex);
}
catch (BipartiteGraphException bge) {
    throw new MatchSetException("Error while searching forward
                                star of edge " + tempIndex + "
                                in the graph: \n\t" +
                                bge.getMessage());
}

// Inserting inverted edge in found forward star
try {
    newEdgeForwardStar.insertEdge(e);
}
catch (ForwardStarException fse) {
    throw new MatchSetException("Error while inserting inverted
                                " + parent + "-" + tempIndex + "
                                edge in forward star: \n\t" +
                                fse.getMessage());
}

// Removing super-root edge to first set node from auxiliary
// graph, if this is the last A-A path edge and the edge has not
// been yet removed
if ((p[parent] == root) && (rootForwardStar.isInByDest(parent)))
    try {
        rootForwardStar.deleteEdgeByDest(parent);
    }
    catch (ForwardStarException fse) {

```

```

        throw new MatchSetException("Error while deleting an
                                     edge in root fs to update
                                     auxiliary graph: \n\t" +
                                     fse.getMessage());
    }

} // End of external edge in A-A path case

// Internal edge in A-A path case
else {

    // Getting parent index
    int parent= p[tempIndex];

    // Transforming parent index in matchedSecondSet vector index
    int parentIndex= parent - this.firstSetCard;

    //if ((this.matchedSecondSet[parentIndex]) &&
    //((this.matchFirstSet[tempIndex] == parent)) {
        this.matchedSecondSet[parentIndex]= false;
        this.pairsNumber--;
        //this.matchEdgeCost[tempIndex]= null;

    // Getting parent forward star
    try {
        forwardStarTemp= graph.getForwardStar(parent);
    }
    catch (BipartiteGraphException bge) {
        throw new MatchSetException("Error searching " + parent
                                     + " forward star: \n\t" +
                                     bge.getMessage());
    }

    // Getting internal edge in forward star
    try {
        e= forwardStarTemp.getEdgeByDest(tempIndex);
    }
    catch (ForwardStarException fse) {
        throw new MatchSetException("Error while searching edge
                                     " + parent + "-" + tempIndex
                                     + " in forward star: \n\t" +
                                     fse.getMessage());
    }

    // Removing this edge from forward star
    try {
        forwardStarTemp.deleteEdge(e);
    }
    catch (ForwardStarException fse) {
        throw new MatchSetException("Error while removing edge "
                                     + parent + "-" + tempIndex +
                                     " from forward star: \n\t" +
                                     fse.getMessage());
    }

    // Inverting edge cost sign
    e.getCost().invertCost();

    // Getting edge destination and source nodes
    int destinationNode= e.getDest();
    int sourceNode= e.getSource();

    if ((tempIndex != destinationNode) || (parent !=

```

```

        sourceNode))
            throw new MatchSetException("Unexpected error on edge
                                         source or destination node");

// Inverting edge direction
try {
    e.setDest(sourceNode);
}
catch (EdgeException ee) {
    throw new MatchSetException("Error while inverting the
                                edge " + sourceNode + "-" +
                                destinationNode +
                                "(destination error): \n\t" +
                                ee.getMessage());
}

try {
    e.setSource(destinationNode);
}
catch (EdgeException ee) {
    throw new MatchSetException("Error while inverting the
                                edge " + sourceNode + "-" +
                                destinationNode + "(source
                                error): \n\t" + ee.getMessage());
}

// Searching destination node forward star
ForwardStar newEdgeForwardStar= null;
try {
    newEdgeForwardStar= graph.getForwardStar(tempIndex);
}
catch (BipartiteGraphException bge) {
    throw new MatchSetException("Error while searching " +
                                tempIndex + " forward star
                                in the graph: \n\t" +
                                bge.getMessage());
}

// Inserting inverted edge in found forward star
try {
    newEdgeForwardStar.insertEdge(e);
}
catch (ForwardStarException fse) {
    throw new MatchSetException("Error inserting inverted
                                edge " + parent + "-" +
                                tempIndex + " in node
                                forward star: \n\t" +
                                fse.getMessage());
}

// Subtracting edge cost from total assignment cost.
this.matchCost.sub(e.getCost());

}

// If it was an even iteration, the next is odd, and vice-versa
oddEven++;

// Going up on the tree
tempIndex= p[tempIndex];
}

}

```

```

/**
 * Returns <code>true</code> if assignment is empty: no pairs found.
 *
 * @return <code>True</code> for empty assignment, <code>false</code>
 * otherwise.
 */
public boolean isEmpty() {
    if (this.pairsNumber == 0)
        return (true);
    else
        return (false);
}

/**
 * Prints the assignment in minutiae format.
 */
public void printMinutiae() {
    double mean= 0;
    if (this.pairsNumber > 0)
        mean= this.matchCost.getSum() / this.pairsNumber;

    System.out.println("\n\n--- Printing results ---\n\n");

    if (this.pairsNumber < this.firstSetCard)
        System.out.println("- NOT all first fingerprint minutiae has been
                           matched.");
    else
        System.out.println("- All first fingerprint minutiae has been
                           matched.");
    System.out.println("- First minutiae set cardinality " +
                       this.firstSetCard);
    System.out.println("- Second minutiae set cardinality " +
                       this.secondSetCard);
    System.out.println("- Pairs matched " + this.pairsNumber);
    System.out.println("- Match cost: " + this.matchCost.getSum());
    System.out.println("- Average cost per edge: " + mean);

    // Printing informaion on matched pairs
    for (int i= 0; i < this.firstSetCard ; i++) {
        int m= this.matchFirstSet[i];
        if (m == this.notMatched)
            System.out.println("\n\t- Minutia " + i + " not matched!");
        else {
            System.out.println("\n\t- Minutia " + i + " matched with minutia
                               " + (this.matchFirstSet[i] -
                                   this.firstSetCard));
            System.out.print("\t- Match cost: " +
                             this.matchEdgeCost[i].getSum());

            // Reporting good matching score
            if (this.matchEdgeCost[i].getSum() < 20)
                System.out.println(" *****");
            else
                System.out.println("");
        }
    }

    if (this.secondSetCard > this.pairsNumber) {

```

```

        System.out.println("\nMinutiae in second fingerprint not matched
                                are:");
        for (int j= 0; j < this.secondSetCard; j++) {
            if (this.matchedSecondSet[j] == false)
                System.out.println("\n\t- Minutia " + j + " not matched.");
        }
    }
    else
        System.out.println("\nAll minutiae in second fingerprint are
                                matched!");
}

/**
 * Returns total match cost.
 */
public double returnMatchCost() {

    return (this.matchCost.getSum());

}

/**
 *
 * Prints the assignment in neighbors format.
 */
public void printNeighbors() {

    double mean= 0;
    if (this.pairsNumber > 0)
        mean= this.matchCost.getSum() / this.pairsNumber;
    System.out.println("\n\n--- Printing results ---\n\n");
    System.out.println("- First neighbor set cardinality " +
                        this.firstSetCard);
    System.out.println("- Second neighbor set cardinality " +
                        this.secondSetCard);
    System.out.println("- Pairs matched " + this.pairsNumber);
    System.out.println("- Match cost: " + this.matchCost.getSum());
    System.out.println("- Average cost per edge: " + mean);

    // Printing informaion on matched pairs
    for (int i= 0; i < this.firstSetCard ; i++) {
        int m= this.matchFirstSet[i];
        if (m == this.notMatched)
            System.out.println("\n\t- Neighbor " + i + " not matched!");
        else {
            System.out.println("\n\t- Neighbor " + i + " matched with
                                neighbor " + (this.matchFirstSet[i] -
                                this.firstSetCard));
            System.out.println("\t- Match cost: " +
                                this.matchEdgeCost[i].getSum());
        }
    }

    if (this.secondSetCard > this.pairsNumber) {
        System.out.println("\nNeighbors of second minutia not matched
                                are:");
        for (int j= 0; j < this.secondSetCard; j++) {
            if (this.matchedSecondSet[j] == false)
                System.out.println("\n\t- Neighbors " + j + " not
                                matched.");
        }
    }
}

```

```
        }  
    }  
    else  
        System.out.println("\nAll neighbors of the second minutia are  
                             matched!");  
}  
  
}
```

```

/*
 * MatchSetException.java
 *
 * Created on 30 ottobre 2002, 3.58
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.minCostAssigning;

/**
 * Thrown by methods in <code>minCostAssigning.MatchSet</code> class.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MatchSetException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>MatchSetException</code> without detail
     * message.
     */
    public MatchSetException() {
    }

    /**
     * Constructs an instance of <code>MatchSetException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public MatchSetException(String msg) {
        super(msg);
    }
}

```



```

/*
 * MinutiaeAssigner.java
 *
 * Created on 10 dicembre 2002, 12.08
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package matching.minCostAssigning;

```

```

/***** IMPORTS *****/

```

```

import matching.graph.*;
import matching.shortestPathTree.*;

```

```

/**

```

```

 * This class <code>Assigner</code> can be used to stop minutiae assigning if
 * match cost grows too much. It is very similar to
 * <code>minCostAssigning.Assigner</code>
 * class.

```

```

 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class MinutiaeAssigner {

```

```

    /**
     * The matching is interrupted if its cost increment is above this value.
     */
    private EdgeCost maxChangeMatchCost= null;

```

```

    /** Creates a new instance of Assigner */
    public MinutiaeAssigner() {

```

```

    }

```

```

    /**
     * Assigner method. Found a minimum cost assignment on bipartite passed
     * graph.
     *
     * @param graph Method <code>assign</code> finds a maximum cardinality
     * minimum cost assignemnt on this graph.
     */

```

```

* @return Data structure representing the found assignment.
*
* @throws AssignerException If fails to find an assignment (e.g. error
*                           manipulating the graph)
*/
public MatchSet assign(BipartiteGraph graph) throws AssignerException {

    int firstSetSize= graph.getFirstSetNodeNumber();
    int secondSetSize= graph.getSecondSetNodeNumber();

    System.out.println("\nMinutiae Assigner.\n");

    // Creating first maxMatchCost value.
    try {
        this.maxChangeMatchCost= new
            EdgeCost((float)5, (float)5, (float)5, (short)5,
                (float)5);
    }
    catch (EdgeCostException ece) {
        throw new AssignerException("Error creating max-change match cost:
            \n\t" + ece.getMessage());
    }

    // Adding super-root to make auxiliary graph
    int newStartNode= graph.addNode();

    // Creating super-root forward star
    ForwardStar startNodeForwardStar= null;
    try {
        startNodeForwardStar= new ForwardStar(graph.getNodeNumber(),
            firstSetSize);
    }
    catch (ForwardStarException fse) {
        throw new AssignerException("Error creating super-root forward star:
            \n\t" + fse.getMessage());
    }

    // Creating cost for super-root forward star edges: set to zero.
    EdgeCost startNodeEdgeCost= new EdgeCost();

    // Inserting edges in super-root forward star: one edge for each first
    // set nodes.
    for (int i= 0; i < firstSetSize; i++) {
        Edge startNodeEdge= null;
        try {
            startNodeEdge= new Edge(newStartNode, i, startNodeEdgeCost);
        }
        catch (EdgeException ee) {
            throw new AssignerException("Error creating super-root edge with
                destination " + i + ": \n\t" +
                ee.getMessage());
        }

        try {
            startNodeForwardStar.insertEdge(startNodeEdge);
        }
        catch (ForwardStarException fse) {
            throw new AssignerException("Error inserting edge " + i + " in
                super-root forward star : \n\t" +
                fse.getMessage());
        }
    }
} // End of inserting edges in super-root forward star

```

```

// Inserting super-root forward star in graph obtaining auxiliary graph
try {
    graph.addForwardStar(newStartNode, startNodeForwardStar);
}
catch (BipartiteGraphException bge) {
    throw new AssignerException("Error inserting super-root forward star
                                in bipartite graph :\n\t" +
                                bge.getMessage());
}

// New number of node in graph after adding super-root
int nodeNumber= newStartNode + 1;

// Creating data structure to represent the assignment
MatchSet ms= null;
try {
    ms= new MatchSet(firstSetSize, secondSetSize);
}
catch (MatchSetException mse) {
    throw new AssignerException("Error creating data structure to
                                represent the assignment: \n\t" +
                                mse.getMessage());
}

// Number of loop iteration (to stop infinite loop caused by errors)
int i= 0;
boolean cond= true;

// Assignment loop
while (cond) {
    i++;
    if (i > firstSetSize) {
        throw new AssignerException("Not Expected error: too much
                                    iterations: " + i);
    }

    // Creating Shortest Path Tree Finder
    SPT spt= new SPT();

    // Parent vector representing shortest path tree
    int p[] = new int[nodeNumber];
    int newMatchedNode= 0;

    // Calculating SPT over auxiliary graph
    try {
        newMatchedNode= spt.calc(graph, newStartNode, p, ms);
    }
    catch (SPTException spte) {
        throw new AssignerException("Error calculating SPT at iteration
                                    " + i + ": \n\t" + spte.getMessage());
    }

    // Checking if has been found alternating-augmenting path
    // destination node
    if (newMatchedNode == -1)

        // Exiting assignment loop
        break;
    else {

        // Getting Match cost before changing it.
        EdgeCost beforeMatchCost= null;
        try {

```

```

        beforeMatchCost= new EdgeCost(ms.getMatchEdgeCost());
    }
    catch (EdgeCostException ece) {
        throw new AssignerException("Error getting match cost before
                                   changing the path: \n\t" +
                                   ece.getMessage());
    }

    // Changing current assignment using alternating-augmenting path
    // found by SPT
    try {
        ms.changeMatch(graph, p, newStartNode, newMatchedNode);
    }
    catch (MatchSetException mse) {
        throw new AssignerException("Error changing the assignment:
                                   \n\t" + mse.getMessage());
    }

    // Getting Match cost after changing it
    EdgeCost afterMatchCost= null;
    try {
        afterMatchCost= new EdgeCost(ms.getMatchEdgeCost());
    }
    catch (EdgeCostException ece) {
        throw new AssignerException("Error getting match cost after
                                   changing the path: \n\t" +
                                   ece.getMessage());
    }

    afterMatchCost.sub(beforeMatchCost);

    if (afterMatchCost.isGreater(this.maxChangeMatchCost)) {
        System.out.println("Exited at iteration " + i);
        break;
    }

}
// Exiting from assigning cycle if one of two set is completely
// assigned
if ((ms.getPairsNumber() == ms.getFirstSetCard()) ||
    (ms.getPairsNumber() == ms.getSecondSetCard()))

    // Exiting assignment loop
    break;

} // End of assignment loop

// Controlling if all first set nodes are matched. In this software
// configuration may not needed:
// throws exception for nodes with no arc (low reliability node)
/*if (ms.getPairsNumber() != ms.getFirstSetCard())
    throw new AssignerException("Matched only " + ms.getPairsNumber() +
                                "pairs");*/

// Returning found maximum cardinality minimum cost assignment
return (ms);
}
}

```

C.3.3 Package shortestPathTree.

```
/*
 * NodeCollection.java
 *
 * Created on 28 ottobre 2002, 19.19
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.shortestPathTree;

/***** IMPORTS *****/
import java.util.LinkedList;

/**
 * This class represent a nodes set used by Shortest Path Tree algorithm.<p>
 * Different implementation of this collection (FIFO, ordered list, ...) affect
 * SPT (hence assignment algorithm) computational complexity.<p>
 * In this version, is implemented as a FIFO list.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class NodeCollection {

    /**
     * Element in position <i>i</i> is <code>true</code> if node of index
     * <i>i</i> is in this set of nodes.
     */
    private boolean[] presenceFlag;

    /**
     * Flag indicating <i>"no node"</i> in this node collection. Can be returned
     * by method <code>extract()</code>
     */
    private static final int noNodeNumber= -1;

    /**
     * Number of node inside this set.
     */
    private int nodeNumber= 0;
```

```

/**
 * Data structure used to collect nodes.
 */
private LinkedList queue= null;

/**
 * Creates a new instance of NodeCollection.
 *
 * @param maxNodeNumber Maximum node number in node collection.
 *
 * @throws NodeCollectionException If fails to create a set (e.g. negative
 *                                 maximum number of element inside the set)
 */
public NodeCollection(int maxNodeNumber) throws NodeCollectionException {
    if (maxNodeNumber < 2)
        throw new NodeCollectionException("Not a valid collection element
                                           number");
    this.presenceFlag= new boolean[maxNodeNumber];
    for (int i= 0; i < maxNodeNumber; i++)
        presenceFlag[i]= false;
    this.nodeNumber= maxNodeNumber;
    this.queue= new LinkedList();
}

/**
 * Inserts a node in this set. <code>NodeCollection</code> is a FIFO list.
 *
 * @param nodeId Identifier of the node to be inserted.
 *
 * @throws NodeCollectionException If fails to insert a node in this set.
 *                                 (e.g. negative node identifier passed)
 */
public void insert(int nodeId) throws NodeCollectionException {
    if ((nodeId < 0) || (nodeId >= this.nodeNumber))
        throw new NodeCollectionException("Node out of range");
    Integer ins= new Integer(nodeId);
    this.queue.addLast(ins);
    this.presenceFlag[nodeId]= true;
}

/**
 * Extracts the first node identifier from the list representing the
 * collection.
 *
 * @return Node identifier extracted.
 */
public int extract() {
    if (this.queue.isEmpty()) {
        return (this.noNodeNumber);
    }
    Integer extInt= (Integer) (this.queue.removeFirst());
    int ext= extInt.intValue();
    this.presenceFlag[ext]= false;
    return (ext);
}

```

```

}

/**
 * Checks the presence of a node in this <code>NodeCollection</code>.
 *
 * @param nodeId Node index to checked in this set.
 * @return <code>True</code> if the node is inside the collection,
 *         <code>false</code> otherwise
 *
 * @throws NodeCollectionException If fails to check a node in this set.
 *         (e.g. negative node identifier passed)
 */
public boolean isNotIn(int nodeId) throws NodeCollectionException {
    if ((nodeId < 0) || (nodeId >= this.nodeNumber))
        throw new NodeCollectionException("Node out of range");
    return (!this.presenceFlag[nodeId]);
}
}

```

```

/*
 * NodeCollectionException.java
 *
 * Created on 28 ottobre 2002, 21.17
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.shortestPathTree;

/**
 * Thrown by methods in <code>shortestPathTree.NodeCollection</code> class.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class NodeCollectionException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>QException</code> without detail message.
     */
    public NodeCollectionException() {
    }

    /**
     * Constructs an instance of <code>QException</code> with the specified
     * detail message.
     *
     * @param msg the detail message.
     */
    public NodeCollectionException(String msg) {
        super(msg);
    }
}

```



```

/*
 * SPT.java
 *
 * Created on 28 ottobre 2002, 17.56
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package matching.shortestPathTree;
```

```

/***** IMPORTS *****/
import matching.graph.*;
import matching.minCostAssigning.*;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 * This class implements Shortest Path Tree Algorithm needed by Assignment
 * Algorithm.
 * SPT is realized in Bellman version: set of candidate nodes is a queue (FIFO).
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class SPT {

    /**
     * Predecessor label value for root node in the tree.
     */
    private static final int nil= -1;

    /**
     * Returned by <code>NodeCollectio.extract()</code> method if candidate node
     * set is empty.
     */
    private static final int noNode= -1;

    /**
     * Returned by method <code>SPT</code> if no alternating-augmenting path
     * destination node has been found. All nodes in first set or all nodes in
     * second set has been assigned.
     */
    private static final int noNodeFound= -1;

    /**

```

```

    * Creates a new instance of SPT
    */
public SPT() {
}

/**
 * Calculates Shortest Path Tree on graph <i>g</i> starting from <i>root</i>
 * node. Parent vector <p></i> is returned to method
 * <code>MatchSet.changeMatch</code> that change the assignemnt with its
 * help. Current assignment <i>ms</i> is needed in SPT to find minimum of
 * the shortest paths towards a node not yet assigned. This choosen node
 * identifier is returned by SPT and characterizes the alternating-
 * augmenting path (from root to this identifier returned).
 * <code>MatchSet.changeMatch</code> changes the assignement according to
 * this A-A path.
 *
 * @param g SPT runs over this auxiliary graph.
 * @param root Root node of the shortest path tree.
 * @param p Parent vector representing resulting shortest path tree.
 * @param ms Current assignemnt over bipartite graph.
 * @return Node identifier with the smallest distance label in shortest path
 *         tree and not yet assigned.
 *
 * @throws SPTException If fails to calculate shortest path tree (e.g. error
 *         in parameter passed).
 */
public int calc(BipartiteGraph g, int root, int [] p, MatchSet ms) throws
    SPTException {

    int graphNodeNumber= g.getNodeNumber();
    int choosenNode= -1;
    EdgeCost choosenNodeEdgeCost= new EdgeCost();
    choosenNodeEdgeCost.costLimit();

    // Controlling input parameters
    if ((root < 0) || (root >= graphNodeNumber))
        throw new SPTException ("Wrong root node (" + root + ")");

    if ((p.length) != (graphNodeNumber))
        throw new SPTException ("Parent vector wrong size (" + p.length +
            ")");

    if ((g == null) || (ms == null))
        throw new SPTException ("Null input parameters (graph or match
            set)");

    // Creating and initializing parent vector and distance-label vector
    EdgeCost d[]= new EdgeCost[graphNodeNumber];
    for (int i= 0; i < graphNodeNumber; i++) {
        p[i]= root;
        d[i]= new EdgeCost();
        if (i != root)
            d[i].costLimit();
    }

    p[root]= nil;

    // Creating set of processing node used by SPT
    NodeCollection q= null;
    try {
        q= new NodeCollection(graphNodeNumber);
    }

```

```

catch (NodeCollectionException nce) {
    throw new SPTEException("Error creating new node collection: \n\t" +
                            nce.getMessage());
}

// Inserting root node in candidate node set
try {
    q.insert(root);
}
catch (NodeCollectionException nce) {
    throw new SPTEException("Error inserting root node in the collection:
                            \n\t" + nce.getMessage());
}

// Printing passed graph before running SPT algorithm over it
//g.print();

boolean cond= true;

// SPT loop
while (cond) {
    int eNode= 0;

    // Extracting node to process from candidate node set
    eNode= q.extract();

    //System.out.println("Extracted node " + eNode);

    // Exiting from loop if the set is empty
    if (eNode == this.noNode)
        break;
    ForwardStar candFS= null;

    // Retrieving extracted node forward star
    try {
        candFS= g.getForwardStar(eNode);
    }
    catch (BipartiteGraphException bge) {
        throw new SPTEException("Error getting extracted node forward
                                star: \n\t" + bge.getMessage());
    }

    int sizeFS= candFS.getSize();

    // Getting an iterator to scan all the forward star
    Iterator iterator= candFS.getEdgeIterator();

    // Processing all edges of extracted node forward star
    for (int j= 0; j < sizeFS; j++) {

        // Getting an edge from forward star
        Edge tempEdge= null;
        try {
            tempEdge= (Edge) (iterator.next());
        }
        catch (NoSuchElementException nsee) {
            throw new SPTEException("Unexpected error scanning forward
                                    star elements: \n\t" +
                                    nsee.getMessage());
        }

        // Getting cost of the edge
        EdgeCost nodeCost= tempEdge.getCost();
    }
}

```

```

// Getting destination node of this edge: source is the
// extracted node
int dest= tempEdge.getDest();
EdgeCost newCost= null;

//System.out.println(eNode + "(" + d[eNode].getSum() + ") toward
//" + dest + " costing " + nodeCost.getSum());

// Obtaining the path cost between extracted node and
// destination node of the edge connecting them
newCost= d[eNode].add(nodeCost);

//System.out.println("Old cost to arrive at " + dest + " from "
//+ p[dest] + " costing " + d[dest].getSum());

// Checking if new path cost is smaller than the old one
if ((newCost.isSmaller(d[dest]))) {

    // Changing parent vector and distance-label vector if new
    // path is smaller
    d[dest]= newCost;
    p[dest]= eNode;

    //System.out.println("\nIt's smaller so i change: from " +
    //eNode + " to " + dest + " costing " + d[dest].getSum());

    // Checking if destination node is not yet assigned
    boolean isNotMatched= false;
    try {
        isNotMatched= ms.isNotMatchedSecondSet(dest);
    }
    catch (MatchSetException mse) {
        throw new SPTException("Error cheching if node " + dest
                                + " is matched: \n\t" +
                                mse.getMessage());
    }

    if (isNotMatched)

        // Remebering its identifier if not yet assigned and its
        // distance label is the smallest
        if (d[dest].isSmaller(choosenNodeEdgeCost)) {
            choosenNode= dest;
            choosenNodeEdgeCost= d[dest];
        }
    boolean ini= false;

    // Checking the presence of destination node in candidate
    // node set
    try {
        ini= q.isNotIn(dest);
    }
    catch (NodeCollectionException nce) {
        throw new SPTException("Error checking if node " + dest
                                + " is inside candidate node set:
                                \n\t" + nce.getMessage());
    }

    // Inserting destination node if not present in candidate
    // node set
    if (ini)
        try{

```

```

        q.insert(dest);

        //System.out.println("From " + eNode + " i have
        //inserted " + dest);

    }
    catch (NodeCollectionException nce) {
        throw new SPTException("Error inserting node " +
                                dest + "in candidate node
                                set: \n\t"+ nce.getMessage());
    }

    } // End of checking if new path cost is smaller than the old
    // one

    } // End of processing all edges of extracted node forward star
} // End of SPT main loop

// Checking if choosenNode value is not changed
if (choosenNode == -1)

    // Returning noNodeFound flag if no A-A path destination node has
    // been found
    return (this.noNodeFound);
else

    // Returning A-A path destination node found
    return (choosenNode);

}

}

```

```

/*
 * SPTEException.java
 *
 * Created on 28 ottobre 2002, 18.08
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching.shortestPathTree;

/**
 * Thrown by methods in <code>shortestPathTree.SPT</code> class.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class SPTEException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>SPTEException</code> without detail
     * message.
     */
    public SPTEException() {
    }

    /**
     * Constructs an instance of <code>SPTEException</code> with the specified
     * detail message.
     *
     * @param msg the detail message.
     */
    public SPTEException(String msg) {
        super(msg);
    }
}

```

C.3.4 Classe CheckEdge.

```
/*
 * CheckEdge.java
 *
 * Created on 8 novembre 2002, 12.39
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching;

/***** IMPORTS *****/
import algMinInformation.*;

/**
 * Methods in this class helps <code>Matcher</code> deciding to insert an edge
 * between two minutiae. Having few edges is important for assignment algorithm
 * performamnce.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class CheckEdge {

    /**
     * Minimum reliability for source minutia in the edge.
     */
    private static final float sourceReliability= (float) 0.45;

    /**
     * Minimum reliability for destination minutia in the edge.
     */
    private static final float destReliability= (float) 0.45;

    /**
     * Creates a new instance of CheckEdge
     */
    public CheckEdge() {
    }
}
```

```

/**
 * Decides to insert or not an edge in minutiae bipartite graph. The edge
 * connects nodes representing parameter minutiae.
 *
 * @param sourceMin Source node of the edge.
 * @param destMin Destination node of the edge.
 * @return <code>True</code> if the edge must be inserted in the graph
 */
public boolean check(AlgMinutia sourceMin, AlgMinutia destMin) {

    //float sourceMinXPos= sourceMin.getXPos();
    //float sourceMinYPos= destMin.getYPos();
    //float destMinXPos= destMin.getXPos();
    //float destMinYPos= destMin.getYPos();
    //float distX= Math.abs(sourceMinXPos - destMinXPos);
    //float distY= Math.abs(sourceMinYPos - destMinYPos);
    //double radius= Math.sqrt((double)(distX * distX + distY * distY));
    float myReliability= sourceMin.getReliability();
    float otherReliability= destMin.getReliability();
    boolean reliabilityCheck= (myReliability < sourceReliability) ||
                               (otherReliability < destReliability);
    if (/*(radius > 70) ||*/ reliabilityCheck /*|| (!
    sourceMin.getType().equalType(destMin.getType())) || ((myReliability -
    otherReliability) > 0.4)*/)
        return (false);
    else
        return (true);
}

}

```


C.3.5 Classe Main Matcher.

```
/*
 * Matcher.java
 *
 * Created on 28 ottobre 2002, 17.04
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
```

```
package matching;
```

```
/****** IMPORTS *****/
import algMinInformation.*;
import matching.graph.*;
import matching.minCostAssigning.*;
import minutiaeSuperClasses.NeighborListException;
import minutiaeSuperClasses.MinutiaeListException;

/**
 * Matcher class. Gives a two fingerprints matching valuation.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class Matcher {

    /**
     * Set <i>true</i> to use a different assigner for minutiae match.
     */
    private boolean differentAssigner= false;

    /**
     * Parameter to use in ridge count difference normalization.
     */
    private static final int maxRC= 30;

    /**
     * Creates a new instance of Matcher.
     */
    public Matcher() {
```

```
}
```

```
/**
 * Builds bipartite minutiae graph and runs minimum cost assignment
 * algorithm over it. Edge cost of this graph are obtained running the same
 * minimum cost assignment algorithm over minutiae neighbors graph: this
 * assignment cost is the edge cost in minutiae graph.<p>
 * To make bipartite minutiae neighbors graph, takes first minutia neighbors
 * and second minutia neighbors, where these two minutiae are the ones
 * connected by an edge.<p>
 * At last prints resulting assignment.
 *
 * @param firstMinList First minutiae list.
 * @param secondMinList Second minutiae list.
 * @return Match set representing algorithm result.
 *
 * @throws MatcherException If fail to match minutiae. (e.g. error building
 *                           graph).
 */
public MatchSet match(AlgMinutiaeList firstMinList, AlgMinutiaeList
                      secondMinList) throws MatcherException {

    // Exchanging list: first list must have less minutiae
    /*if (firstMinList.size() > secondMinList.size()) {
        AlgMinutiaeList tempList= firstMinList;
        firstMinList= secondMinList;
        secondMinList= tempList;
        System.out.println("List positions exchanged.");
    }*/

    // Creating an Assigner
    Assigner asgn= new Assigner();

    // Crea minutiae bipartite graph.
    int n1= firstMinList.size();
    int n2= secondMinList.size();
    MatchSet minutiaeMatch= null;
    BipartiteGraph minGraph= null;
    try {
        minGraph= new BipartiteGraph(n1, n2);
    }
    catch (BipartiteGraphException bge) {
        throw new MatcherException("Error creating minutiae graph: \n\t" +
                                   bge.getMessage());
    }

    // Creating a Checker to check if edge between minutiae must be inserted
    // in graph.
    CheckEdge ce= new CheckEdge();
    boolean insert= false;

    // Retrieving global information about minutiae lists.
    float firstListMaxNeighDistance= firstMinList.getMaxNeighDistance();
    float secondListMaxNeighDistance= secondMinList.getMaxNeighDistance();
    short firstListMaxRidgeCount= firstMinList.getMaxRidgeCount();
    short secondListMaxRidgeCount= secondMinList.getMaxRidgeCount();

    // Obtaining values to normaliza euclidean distance between neighbors
    // and ridge count difference between neighbors
    float euclideanDistanceNormalization= (firstListMaxNeighDistance >
                                             secondListMaxNeighDistance)? firstListMaxNeighDistance
                                             : secondListMaxNeighDistance;
```

```

float ridgeCountNormalization= (firstListMaxRidgeCount >
                                secondListMaxRidgeCount)? firstListMaxRidgeCount :
                                secondListMaxRidgeCount;
//float ridgeCountNormalization= this.maxRC;

// If not using MINDTCT neighbors, ridge count field is filled with zero
// for all the neighbors. So it's not an error but can't divide by zero
if (ridgeCountNormalization == 0)
    ridgeCountNormalization= 1;

// Scanning minutiae in first minutiae bipartite set
for (int i= 0; i < n1; i++) {
    //System.out.println("minutia " + i);

    AlgMinutia minFromFirstList= null;
    try {
        minFromFirstList= firstMinList.get(i);
    }
    catch (MinutiaeListException mle) {
        throw new MatcherException("Error getting " + i + " element from
                                   minutiae first list: \n\t" +
                                   mle.getMessage());
    }

    int neigh1= minFromFirstList.getAlgNeighborList().size();
    ForwardStar fs= null;

    // Creating first set node forward star
    try {
        fs= new ForwardStar(30, i);
    }
    catch (ForwardStarException fse) {
        throw new MatcherException("Error creating forward star for
                                   minutia " + i + ": \n\t" +
                                   fse.getMessage());
    }

    // Scanning minutiae in second minutiae bipartite set
    for (int j= 0; j < n2; j++) {
        int dest= n1 + j;

        AlgMinutia minFromSecondList= null;
        try {
            minFromSecondList= secondMinList.get(j);
        }
        catch (MinutiaeListException mle) {
            throw new MatcherException("Error getting " + j + " element
                                       from minutiae second list: \n\t"
                                       + mle.getMessage());
        }

        // Controlling if the edge between minutia i in first set and
        // minutia j in second set must be inserted.
        insert= ce.check(minFromFirstList, minFromSecondList);
        if (! insert)
            continue;

        MatchSet ms= null;
        EdgeCost newEdgeCost= null;
        int neigh2= minFromSecondList.getAlgNeighborList().size();

        // Controlling if two minutiae neighbor number is the same.
        if (neigh1 != neigh2)

```

```

        throw new MatcherException("Minutiae have a different number
                                   of neighbors: " + neigh1 +
                                   " " + neigh2);

// Creating neighbors bipartite graph
BipartiteGraph neighborGraph= null;
try {
    neighborGraph= new BipartiteGraph(neigh1, neigh2);
}
catch (BipartiteGraphException bge) {
    throw new MatcherException("Error creating neighbor graph
                                between minutiae " +
                                minFromFirstList.getId()
                                + " and " + minFromSecondList.getId()
                                + ": \n\t" + bge.getMessage());
}

// Scanning minutia in first list neighbors
for (int k= 0; k < neigh1; k++) {
    ForwardStar neighfs= null;
    try {
        neighfs= new ForwardStar(neigh2, k);
    }
    catch (ForwardStarException fse) {
        throw new MatcherException("Error creating forward star
                                    for minutia in first list " +
                                    minFromFirstList.getId() +
                                    ", neighbor in " + k + "
                                    position: \n\t" +
                                    fse.getMessage());
    }

    // Retrieving neighbor k of first minutia
    AlgNeighbor f1= null;
    try {
        f1= minFromFirstList.getAlgNeighborList().get(k);
    }
    catch (NeighborListException nle) {
        throw new MatcherException("Error getting neighbor " + k
                                    + " for minutia in first list
                                    " + minFromFirstList.getId()
                                    + ": \n\t" + nle.getMessage());
    }

    // Retrieving minutia in first list neighbor features
    float ed1= f1.getDistance();
    float dra1= f1.getDistanceRelativeAngle();
    float ra1= f1.getRelativeAngle();
    short rc1= f1.getRidgeCount();
    float qf1= f1.fifthFeature;

    // Scanning neighbors of minutia in second list
    for (int h= 0; h < neigh2; h++) {

        AlgNeighbor f2= null;
        try {
            f2= minFromSecondList.getAlgNeighborList().get(h);
        }
        catch (NeighborListException nle) {
            throw new MatcherException("Error getting neighbor "
                                        + h + " for minutia in
                                        second list " +
                                        minFromSecondList.getId() +

```

```

        ": \n\t" + nle.getMessage());
    }

    int neighborDest= h + neigh1;

    // Weighting edges
    /*float weight=
        Math.abs(minFromFirstList.getReliability()
            - minFromSecondList.getReliability());
    if (weight == 0)
        weight= 1;
    else
        weight= weight * 100;*/

    // Calculating euclidean distance difference between
    // neighbors
    float edDiff= (Math.abs(ed1 - f2.getDistance()));
    // Normalizing
    edDiff= edDiff / euclideanDistanceNormalization;
    if (edDiff > 1)
        throw new MatcherException("Error normalizing ed: "
            + edDiff);
    //edDiff= (int) edDiff;

    // Calculating distance-relative angle difference
    // between neighbors
    float draDiff= dra1 - f2.getDistanceRelativeAngle();
    draDiff= draDiff % 360;
    if (draDiff < 0)
        draDiff= -draDiff;
    if (draDiff > 180)
        draDiff= 360 - draDiff;
    // Normalizing
    draDiff= draDiff / 180;
    if (draDiff > 1)
        throw new MatcherException("Error normalizing dra: "
            + draDiff);
    //draDiff= (int) draDiff;

    // Calculating relative angle difference between
    // neighbors
    float raDiff= ra1 - f2.getRelativeAngle();
    raDiff= raDiff % 360;
    if (raDiff < 0)
        raDiff= -raDiff;
    if (raDiff > 180)
        raDiff= 360 - raDiff;
    // Normalizing
    raDiff= raDiff / 180;
    if (raDiff > 1)
        throw new MatcherException("Error normalizing ra: "
            + raDiff);
    //raDiff= (int) raDiff;

    // Calculating ridge count difference
    float rcDiff= rc1 - f2.getRidgeCount();
    if (rcDiff < 0)
        rcDiff= -rcDiff;
    // Normalizing
    rcDiff= rcDiff / ridgeCountNormalization;
    if (rcDiff > 1)
        throw new MatcherException("Error normalizing rc: "

```

```

        + rcDiff));
//rcDiff= (int) rcDiff;

// Calculating fifth feature
// Uncomment to use fifth feature
/*
float qfDiff= 0;
float neighborDistance= 0;
float dra1Dra2Difference= Math.abs(dra1 -
                                f2.getDistanceRelativeAngle());
dra1Dra2Difference= dra1Dra2Difference % 360;
if (dra1Dra2Difference > 180)
    dra1Dra2Difference= 360 - dra1Dra2Difference;
if (dra1Dra2Difference != 180) {

    float tempCalc1= ed1 * ed1;
    float tempCalc2= f2.getDistance() *
                    f2.getDistance();
    float tempCalc3= (float) (2 * ed1 * f2.getDistance()
        * (Math.cos(Math.toRadians(dra1Dra2Difference))));
    neighborDistance= (float) (Math.sqrt(tempCalc1 +
        tempCalc2 - tempCalc3));
}
else
    neighborDistance= ed1 + f2.getDistance();
// Normalizing
float tempino= neighborDistance;
qfDiff= neighborDistance / (ed1 + f2.getDistance());
//System.out.println ("\ndraDiff = " +
//dra1Dra2Difference + " qfDiff= " + qfDiff);
if (qfDiff > 1) {
    //System.out.println ("\ndraDiff = " +
    //dra1Dra2Difference + " qfDiff= " + qfDiff + " ed1=
    //" + ed1 + " ed2= " + f2.getDistance() + " tempino=
    //" + tempino);
    //throw new MatcherException ("Error normalizing: "
    //+ qfDiff);
    qfDiff= 1;
}

// Printing neighbors differences between minutiae i and
// j
/*if ((i == 4) && (j == 4)) {
    System.out.println("\t\tMinutia " + i + " with " +
        j);
    System.out.println("Neighbor " + f1.getId() + " with
        " + f2.getId());
    System.out.println("- " + ed1 + " " +
        f2.getDistance());
    System.out.println("- " + dra1 + " " +
        f2.getDistanceRelativeAngle());
    System.out.println("- " + ra1 + " " +
        f2.getRelativeAngle());
    System.out.println("Differences are: ed- " + edDiff
        + ", dra-" + draDiff+ ", ra-" +
        raDiff);
    System.out.println("Euclidean normalization: " +
        euclideanDistanceNormalization +
        " " + firstListMaxXPos + " " +
        firstListMaxYPos);
    float sum= edDiff + draDiff + raDiff + rcDiff;
    System.out.println("The sum is " + sum);
}*/

```

```

/*
if ( (edDiff > 1) || (draDiff > 1) || (raDiff > 1) ||
(rcDiff > 1) || (edDiff < 0) || (draDiff < 0) || (raDiff
< 0) || (rcDiff < 0)) {
    System.err.println("Error: parameter not
                        normalized.");
    System.exit(-1);
}*/

//System.out.println(edDiff + " " + draDiff + " " +
//raDiff + " " + rcDiff);

// Creating edge cost for the edge between two neighbors
EdgeCost neighborEdgeCost= null;
try {
    neighborEdgeCost= new EdgeCost(edDiff, draDiff,
                                   raDiff, rcDiff, qfDiff);
}
catch (EdgeCostException ece) {
    throw new MatcherException("Error creating edge cost
                                between neighbors " + k +
                                " and " + h +
                                "(minutiae " + minFromFirstList.getId()
                                + ", " + minFromSecondList.getId() +
                                "): \n\t" + ece.getMessage());
}

// Creating the edge between two neighbors
Edge neighborEdge= null;
try {
    neighborEdge= new Edge(k, neighborDest,
                           neighborEdgeCost);
}
catch (EdgeException ee) {
    throw new MatcherException("Error creating edge
                                between neighbors " + k +
                                " and " + h + "(minutiae "
                                + minFromFirstList.getId() + ", " +
                                minFromSecondList.getId() +
                                "): \n\t" + ee.getMessage());
}

// Inserting created edge in source neighbors forward
// star.
try {
    neighfs.insertEdge(neighborEdge);
}
catch (ForwardStarException fse) {
    throw new MatcherException("Error inserting in
                                forward star, edge between
                                neighbors " + k + " and "
                                + h + "(minutiae " +
                                minFromFirstList.getId() + ", "
                                + minFromSecondList.getId() +
                                "): \n\t" + fse.getMessage());
}

} // End of scanning neighbors of minutia in second list

// Inserting forward star of neighbor k (minutia in first
// list), in bipartite graph
try {
    neighborGraph.addForwardStar(k, neighfs);
}

```

```

    }
    catch (BipartiteGraphException bge) {
        throw new MatcherException("Error inserting forward star
                                   of neighbor " + k + " of minutia " +
                                   minFromFirstList.getId() + " in
                                   first list: \n\t" + bge.getMessage());
    }
} // End of scanning neighbors of minutia in first list

// Adding empty forward-star for nodes in second set in
// neighbors bipartite graph
int t= neigh1 + neigh2;
for (int q= neigh2; q < t; q++) {
    ForwardStar fsTemp= null;
    try {
        fsTemp= new ForwardStar(1, q);
    }
    catch (ForwardStarException fse) {
        throw new MatcherException("Error creating empty forward
                                   star for neighbor " + q + " of minutia " +
                                   minFromSecondList.getId() + " in second
                                   list: \n\t" + fse.getMessage());
    }

    // Inserting empty forward-star for nodes in second set in
    // neighbors bipartite graph
    try {
        neighborGraph.addForwardStar(q, fsTemp);
    }
    catch (BipartiteGraphException bge) {
        throw new MatcherException("Error inserting empty
                                   forward star for neighbor " + q
                                   + " of minutia " +
                                   minFromSecondList.getId() + "
                                   in second list: \n\t" + bge.getMessage());
    }
}

} // End of adding empty forward stars

// Assingment between two minutiae neighbors
try {
    ms= asgn.assign(neighborGraph);
    //System.out.println("\n\n\nSono riuscito da assegnare i
    //vicini\n\n\n");
}
catch (AssignerException ae) {
    throw new MatcherException("Error in assigning neighbors of
                                minutia " + minFromFirstList.getId() +
                                " with neighbors of minutia " +
                                minFromSecondList.getId() + ":\n\t" +
                                ae.getMessage());
}

// Printing neighbors assignment
//ms.printNeighbors();

// Mean with neighbors graph edges (5)
try {
    ms.getMatchEdgeCost().mean(5);
}
catch (EdgeCostException ece) {
    throw new MatcherException("Error finding neighbor graph

```



```

                                assignment cost mean:\n\t" +
                                ece.getMessage());
        }

        // Creating edge cost with the cost of the assignment between
        // neighbors
        try {
            newEdgeCost= new EdgeCost(ms.getMatchEdgeCost());
        }
        catch (EdgeCostException ece) {
            throw new MatcherException("Error in assignment cost between
                                        neighbors of minutia " +
                                        minFromFirstList.getId() +
                                        " with neighbors of minutia " +
                                        minFromSecondList.getId() +
                                        ":\n\t" + ece.getMessage());
        }

        // Creating edge between two minutia in bipartite graph
        Edge newEdge= null;
        try {
            newEdge= new Edge(i, dest, newEdgeCost);
        }
        catch (EdgeException ee) {
            throw new MatcherException("Error creating edge between
                                        minutia " + minFromFirstList.getId()
                                        + " with neighbors of minutia " +
                                        minFromSecondList.getId() + ":\n\t"
                                        + ee.getMessage());
        }

        // Inserting edge between two minutia in bipartite graph
        try {
            fs.insertEdge(newEdge);
        }
        catch (ForwardStarException fse) {
            throw new MatcherException("Error inserting edge between
                                        minutia " + minFromFirstList.getId()
                                        + " with neighbors of minutia " +
                                        minFromSecondList.getId() + ":\n\t"
                                        + fse.getMessage());
        }

    } // End of scanning minutiae in second minutiae bipartite set

    // Inserting forward star in bipartite graph for minutiae in first
    // set
    try {
        minGraph.addForwardStar(i, fs);
    }
    catch (BipartiteGraphException bge) {
        throw new MatcherException("Error inserting forward star of
                                    minutia " + minFromFirstList.getId()
                                    + " in first list: \n\t" +
                                    bge.getMessage());
    }

} // End of scanning minutiae in first minutiae bipartite set

// Creating empty forward-star for nodes in second set in minutiae
// bipartite graph
int t= n1 + n2;

```

```

    for (int w= n1; w < t; w++) {
        ForwardStar fsTemp= null;
        try {
            fsTemp= new ForwardStar(1, w);
        }
        catch (ForwardStarException fse) {
            throw new MatcherException("Error creating empty forward star of
                                     minutia " + w + " in second list:
                                     \n\t" + fse.getMessage());
        }

        // Inserting empty forward-star for nodes in second set in minutiae
        // bipartite graph
        try {
            minGraph.addForwardStar(w, fsTemp);
        }
        catch (BipartiteGraphException bge) {
            throw new MatcherException("Error inserting empty forward star
                                     of minutia " + w + " in second list:
                                     \n\t" + bge.getMessage());
        }
    } // End of creating empty forward-star for nodes in second set in
        // minutiae bipartite graph

    // Checking if using another Assigner for minutiae
    if (this.differentAssigner) {
        MinutiaeAssigner diffAsgn= new MinutiaeAssigner();
        try {
            minutiaeMatch= diffAsgn.assign(minGraph);
        }
        catch (AssignerException ae) {
            throw new MatcherException("Error assigning minutiae: \n\t" +
                                     ae.getMessage());
        }
    }
    else {

        // Minimum cost assigning over minutiae bipartite graph.
        try {
            minutiaeMatch= asgn.assign(minGraph);
        }
        catch (AssignerException ae) {
            throw new MatcherException("Error assigning minutiae: \n\t" +
                                     ae.getMessage());
        }
    }

    // Returning match set
    return (minutiaeMatch);
}

/**
 * Returns a boolean value representing the assignment method to match
 * minutiae used by this <code>Matcher</code> object.
 *
 * @return <i>True</i> if using a different assigner for minutiae: different
 *         assigner stops when assignment cost grows too much; returns
 *         <i>false</i> if using neighbor assigner.
 */
public boolean getDifferentAssigner() {

```

```

        return (this.differentAssigner);
    }

    /**
     * Changes the assigner used by method Matcher.match() to match
     * minutiae. The assigner used to match neighbors can't be changed.
     *
     * @param value Setting this to true chooses to use a different
     * assigner for minutiae. If false method
     * Matcher.match() uses neighbor assigner.
     */
    public void setDifferentAssigner(boolean value) {
        this.differentAssigner= value;
    }
}

```

C.3.6 Classe MatcherException.

```
/*
 * MatcherException.java
 *
 * Created on 28 ottobre 2002, 17.05
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package matching;

/**
 * Exception thrown by methods in class <code>matching.Matcher</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MatcherException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>MatcherException</code> without detail
     * message.
     */
    public MatcherException() {
    }

    /**
     * Constructs an instance of <code>MatcherException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public MatcherException(String msg) {
        super(msg);
    }
}
```

C.4 Package minutiaeFileWriting.

```
/*
 * MinutiaeFileWriter.java
 *
 * Created on 15 ottobre 2002, 11.13
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeFileWriting;

/***** IMPORTS *****/
import java.io.PrintStream;
import algMinInformation.AlgMinutiaeList;
import algMinInformation.AlgMinutia;
import algMinInformation.AlgNeighborList;
import algMinInformation.AlgNeighbor;
import minutiaeSuperClasses.NeighborListException;
import minutiaeSuperClasses.MinutiaeListException;

/**
 * An Object from this class represents a minutiae list writer in output stream.
 * Information format is decided inside <code>MinutiaeFileWriter.write</code>
 * method.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MinutiaeFileWriter {

    /**
     * String used to separate information fields in text line ":".
     */
    private static final String sep= ":";

    /** Creates a new instance of MitutiaeFileWriter */
    public MinutiaeFileWriter() {
    }

    /**
```

```

* Writes matching algorithm compatible minutiae list in output stream
* (e.g. a text file). Text format is decided inside this method.
*
* @param ps Writes in this output stream
* @param mList Minutiae list method <code>MinutiaeFileWriter.write</code>
*             is going to write in output stream.
*
* @throws MinutiaeFileWriterException Thrown when fails to write in output
*                                     stream (e.g. IO error)
*/
public void write(PrintStream ps, AlgMinutiaeList mList) throws
    MinutiaeFileWriterException {

    if ((ps == null) || (mList == null))
        throw new MinutiaeFileWriterException("Null input parameters.");

    // Writing global minutiae list information
    String tempInitialString= Float.toString(mList.getMaxNeighDistance());
    String initialStringToWrite= new String(tempInitialString);
    initialStringToWrite= initialStringToWrite.concat(sep);
    tempInitialString= Short.toString(mList.getMaxRidgeCount());
    initialStringToWrite= initialStringToWrite.concat(tempInitialString);
    initialStringToWrite= initialStringToWrite.concat(sep);

    // Printing initial string
    ps.println (initialStringToWrite);

    int s= mList.size();

    // Writing all the lines in minutiae list
    for (int i= 0; i < s; i++) {

        AlgMinutia mw= null;
        try {
            mw= mList.get(i);
        }
        catch(MinutiaeListException mle) {
            throw new MinutiaeFileWriterException("Error getting " + i + "
                                                    element from minutiae
                                                    list: \n\t" +
                                                    mle.getMessage());
        }

        if (mw == null)
            throw new MinutiaeFileWriterException("Null element in minutiae
                                                    list.");

        // Creating text string to write

        // Writing Id
        String tempString= Short.toString(mw.getId());
        String stringToWrite= new String(tempString);
        stringToWrite= stringToWrite.concat(sep);

        // Writing x position
        tempString= Short.toString(mw.getXPos());
        stringToWrite= stringToWrite.concat(tempString);
        stringToWrite= stringToWrite.concat(sep);

        // Writing y position
        tempString= Short.toString(mw.getYPos());
        stringToWrite= stringToWrite.concat(tempString);
        stringToWrite= stringToWrite.concat(sep);
    }
}

```

```

// Writing direction angle
tempString= Float.toString(mw.getDirection());
stringToWrite= stringToWrite.concat(tempString);
stringToWrite= stringToWrite.concat(sep);

// Writing reliability
tempString= Float.toString(mw.getReliability());
stringToWrite= stringToWrite.concat(tempString);
stringToWrite= stringToWrite.concat(sep);

// Writing minutia type
String typeString= mw.getType().getMinTypeString();
if (typeString == null)
    throw new MinutiaeFileWriterException("Null minutia type
                                           string.");
else
    stringToWrite= stringToWrite.concat(typeString);

// Writing all the neighbors
AlgNeighborList fl= mw.getAlgNeighborList();
int sNList= fl.size();
for (int j= 0; j < sNList; j++) {

    AlgNeighbor fw= null;
    try {
        fw= fl.get(j);
    }
    catch(NeighborListException nle) {
        throw new MinutiaeFileWriterException("Error getting " + j +
                                                "neighbor: \n\t" +
                                                nle.getMessage());
    }

    if (fw == null)
        throw new MinutiaeFileWriterException("Null element in
                                                neighbor AlgNeighbor
                                                list.");

    // Concatenating neighbor information
    stringToWrite= stringToWrite.concat(sep);

    // Writing neighbor Id
    tempString= Short.toString(fw.getId());
    stringToWrite= stringToWrite.concat(tempString);
    stringToWrite= stringToWrite.concat(sep);

    // Writing minutia-neighbor distance
    tempString= Float.toString(fw.getDistance());
    stringToWrite= stringToWrite.concat(tempString);
    stringToWrite= stringToWrite.concat(sep);

    // Writing minutia-neighbor distance-relative angle
    tempString= Float.toString(fw.getDistanceRelativeAngle());
    stringToWrite= stringToWrite.concat(tempString);
    stringToWrite= stringToWrite.concat(sep);

    // Writing minutia-neighbor relative angle
    tempString= Float.toString(fw.getRelativeAngle());
    stringToWrite= stringToWrite.concat(tempString);
    stringToWrite= stringToWrite.concat(sep);

    // Writing minutia-neighbor relative angle

```

```
        tempString= Short.toString(fw.getRidgeCount());
        stringToWrite= stringToWrite.concat(tempString);
    }

    // Printing string in output stream
    ps.println (stringToWrite);
}

return;

}

}
```



```

/*
 * MinutiaeFileWriterException.java
 *
 * Created on 15 ottobre 2002, 11.14
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeFileWriting;

```

```

/**
 * Thrown by <code>MinutiaeFileWriter</code> when fails to write new format
 * file containing minutiae information compatible with matching algorithm.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MinutiaeFileWriterException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>MinutiaeFileWriterException</code>
     * without detail message.
     */
    public MinutiaeFileWriterException() {
    }

    /**
     * Constructs an instance of <code>MinutiaeFileWriterException</code> with
     * the specified detail message.
     *
     * @param msg the detail message.
     */
    public MinutiaeFileWriterException(String msg) {
        super(msg);
    }
}

```

C.5 Package minutiaeInformation.

```
/*
 * FeatureType.java
 *
 * Created on 11 ottobre 2002, 14.59
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeInformation;

/**
 * Class representing type of feature detected by <i>MINDTCT</i> algorithm.
 * <i>APP</i> stands for appearing and <i>DIS</i> stands for disappearing.<p>
 * See <i><b>MINDTCT.txt</b></i> attached to this project (row 146): used for
 * <i>MINDTCT</i> internal purposes.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class FeatureType {

    /**
     * String for appearing type.
     */
    private static final String appString= "APP";

    /**
     * String for disappearing type.
     */
    private static final String disString= "DIS";

    /**
     * No type inserted.
     */
    private static final String noFeatureTypeString= "NO TYPE";

    /**
     * Integer representation of type.
     */
    private byte type= -1;
```

```

/** Creates a new instance of FeatureType
 *
 * @throws FeatureTypeException If fail to create a new deature type.
 *                               (e.g. wrong type string passed: not "APP" or
 *                               "DIS").
 */
public FeatureType(String featureTypeString) throws FeatureTypeException {
    if (featureTypeString.equalsIgnoreCase(appString))
        this.type= 0;
    else
        if (featureTypeString.equalsIgnoreCase(disString))
            this.type= 1;
        else
            throw new FeatureTypeException("Feature Type not allowed.");
}

/**
 * Returns a byte value representing minutia feature type.
 *
 * @return Byte representing type
 */
public byte getFeatureType() {
    return (this.type);
}

/**
 * Method returning feature type string.
 *
 * @return Type string: "APP", "DIS", "NO TYPE".
 */
public String getFeatureTypeString() {
    switch (this.type) {
        case 0:
            return (appString);
        case 1:
            return (disString);
        default:
            return (noFeatureTypeString);
    }
}

/**
 * Sets feature type for this minutia.
 *
 * @param featureTypeString String representing minutia type: can be
 * <i>"APP" "DIS"</i> or <i>"NO TYPE"</i>.
 *
 * @throws FeatureTypeException If passed string is not one of those above.
 */
public void setType(String featureTypeString) throws FeatureTypeException {
    if (featureTypeString.equalsIgnoreCase(appString))
        this.type= 0;
    else
        if (featureTypeString.equalsIgnoreCase(disString))
            this.type= 1;

```

```

        else
            throw new FeatureTypeException("Minutia Type not allowed.");
    }

    /**
     * Returns a boolean true if feature type is appearing
     *
     * @return <i>True</i> if is a "APP" type, <i>false</i> otherwise.
     */
    public boolean isAppearing() {
        return (this.type == 0);
    }

    /**
     * Returns a boolean true if feature type is disappearing
     *
     * @return <i>True</i> if is a "DIS" type, <i>false</i> otherwise.
     */
    public boolean isDisappearing() {
        return (this.type == 1);
    }

    /**
     * Returns <i>true</i> if this feature type and minutia feature type passed
     * are the same.
     *
     * @param anotherFeatureType Feature type to confront with this.
     * @return <i>True</i> if feature type is the same.
     */
    public boolean equalType(FeatureType anotherFeatureType) {
        return (this.type == anotherFeatureType.getFeatureType());
    }
}

```

```

/*
 * FeatureTypeException.java
 *
 * Created on 11 ottobre 2002, 15.05
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeInformation;

/**
 * Thrown by methods in <code>minutiaeInformation.FeatureType</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class FeatureTypeException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>FeatureTypeException</code> without
     * detail message.
     */
    public FeatureTypeException() {
    }

    /**
     * Constructs an instance of <code>FeatureTypeException</code> with the
     * specified detail message.
     * @param msg the detail message.
     */
    public FeatureTypeException(String msg) {
        super(msg);
    }
}

```

```

/*
 * MinType.java
 *
 * Created on 9 ottobre 2002, 18.50
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package minutiaeInformation;
```

```
/* ***** IMPORTS ***** */
```

```
import minutiaeSuperClasses.MinTypeSuperClass;
import minutiaeSuperClasses.MinTypeException;
```

```

/**
 * Class extended from <code>minutiaeSuperClasses.MinTypeSuperClass</code>.
 * Represents minutia type for <i>MINDTCT</i> minutiae.
 */

```

```

 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class MinType extends MinTypeSuperClass {
```

```

    /**
     * Creates a new instance of MinType
     *
     * @param typeString String representing minutia type: can be <i>"BIF"
     * "RIG"</i> or <i>"NO TYPE"</i>.
     *
     * @throws MinTypeException If passed string is not one of those above.
     */

```

```
public MinType (String typeString) throws MinTypeException {
```

```

    if (typeString.equalsIgnoreCase(super.bifString))
        super.type= 0;
    else
        if (typeString.equalsIgnoreCase(super.ridgeEndingString))
            super.type= 1;
        else
            throw new MinTypeException("Minutia Type not allowed (" +
                                         typeString + ")");

```

```
}
```

```

/**
 * Returns a boolean true if type is bifurcation type
 *
 * @param anotherMinType Other minutia for the comparison.
 * @return <i>True</i> if is a "BIF" type, <i>false</i> otherwise.
 */
public boolean equalType(MinType anotherMinType) {
    return (super.type == anotherMinType.getMinType());
}

}

```

```

/*
 * Minutia.java
 *
 * Created on 9 ottobre 2002, 18.06
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeInformation;

```

```

/***** IMPORTS *****/

```

```

import algMinInformation.AlgNeighbor;
import minutiaeSuperClasses.MinutiaSuperClass;
import minutiaeSuperClasses.MinutiaException;
import minutiaeSuperClasses.NeighborListException;

```

```

/**
 * This class inherits from <code> minutiaeSuperClasses.MinutiaSuperClass
 * </code>.<p>
 * Represents minutia obtained from <i>MINDTCT</i> text file.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class Minutia extends MinutiaSuperClass{

```

```

    /**
     * Minutia type (bifurcation, ridge-ending....) <p>
     * See <code>minutiaeInformation.MinType</code> class.
     */
    private MinType type= null;

```

```

    /**
     * Says if minutia is <i>APP</i>earing or <i>DIS</i>appearing.<p>
     * See <i><b>MINDTCT.txt</b></i> attached to this project (row 146): used
     * for <i>MINDTCT</i> internal purposes.
     */
    private FeatureType fType= null;

```

```

    /**
     * Integer identifier of the type of AlgNeighbor detected.<p>
     * See <i><b>MINDTCT.txt</b></i> attached to this project (row 152): used
     * for <i>MINDTCT</i> internal purposes.
     */
    private short featureTypeIdentifier= 0;

```



```

/**
 * Minutia neighbors list.
 */
private NeighborsList nbors= null;

/**
 * Creates a new instance of Minutia.
 *
 * @throws MinutiaException Thrown after failing to create minutia
 * neighbors list.
 */
public Minutia() throws MinutiaException {
    try {
        this.nbors= new NeighborsList();
    }
    catch (NeighborListException nle) {
        throw new MinutiaException("Error creating neighbor list: \n\t" +
                                   nle.getMessage());
    }
}

/**
 * Sets minutia new type. (e.g. "BIF", "RIG").
 *
 * @param newType Minutia type to be set.
 */
public void setMinType(MinType newType) {
    this.type= newType;
    return;
}

/**
 * Sets new AlgNeighbor type field. See FeatureType field.
 *
 * @param newFeatureType AlgNeighbor type to be set.
 */
public void setFeatureType(FeatureType newFeatureType) {
    this.fType= newFeatureType;
    return;
}

/**
 * Controls and sets new AlgNeighbor type identifier field. See
 * featureTypeIdentifier field.
 *
 * @param newFeatureType AlgNeighbor type identifier to be set.
 *
 * @throws MinutiaException Invalid new AlgNeighbor type identifier (e.g.
 * less than zero)
 */
public void setFeatureTypeIdentifier(short newFeatureTypeIdentifier) throws
    MinutiaException {

```

```

        if (newFeatureTypeIdentifier < 0) {
            throw new MinutiaException("Invalid identifier of the AlgNeighbor
                                     type (" + newFeatureTypeIdentifier +
                                     ")");
        }
        else {
            this.featureTypeIdentifier= newFeatureTypeIdentifier;
        }
    }

    /**
     * Inserts neighbor passed in minutia neighbor list.
     *
     * @param nbor Neighbor to be inserted.
     */
    public void addNeighbor(Neighbor nbor) {
        nbors.add(nbor);
    }

    /**
     * Returns minutia neighbors.
     *
     * @return This minutia neighbor list.
     */
    public NeighborsList getNeighborsList() {
        return (this.nbors);
    }

    /**
     * Returns <code>minutiaeInformation.MinType</code> minutia type.
     *
     * @return Minutia type.
     */
    public MinType getType() {
        return(this.type);
    }

    /**
     * Returns neighbor from passed position.
     *
     * @param index Index of neighbor requested (from zero to <i>size - 1</i>).
     *
     * @return Returns neighbor <code>index</code>.
     */
    public Neighbor getNeighbor(int index) throws MinutiaException {
        try {
            return (nbors.get(index));
        }
        catch (NeighborListException nle) {
            throw new MinutiaException("Error getting element from list: \n\t" +
                                      nle.getMessage());
        }
    }
}

```

```

/**
 * Prints on <code>System.out</code> minutia and its neighbors.
 *
 */
public void print() throws MinutiaException {

    System.out.println("Minutia identifier = " + this.id);
    System.out.println("\t- x position = " + this.xPos);
    System.out.println("\t- y position = " + this.yPos);
    System.out.println("\t- direction angle = " + this.direction);
    System.out.println("\t- type = " + this.type.getMinTypeString());
    System.out.println("\t- reliability = " + this.reliability);
    System.out.println("\t- AlgNeighbor type = " +
        this.fType.getFeatureTypeString());
    System.out.println("\t- AlgNeighbor type identifier = " +
        this.featureTypeIdentifier);
    int s= this.nbors.size();
    System.out.println("\tMinutia has " + s + " neighbors.");
    Neighbor rifNeighbor= null;
    for (int i= 0; i < s; i++) {
        try {
            rifNeighbor= (Neighbor) (this.nbors.get(i));
        }
        catch (NeighborListException nle) {
            throw new MinutiaException(nle.getMessage());
        }

        System.out.println("\t\t- " + rifNeighbor.stringRepresentation());
    }
}
}

```

```

/*
 * MinutiaeList.java
 *
 * Created on 10 ottobre 2002, 18.23
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeInformation;

```

```

/***** IMPORTS *****/

```

```

import java.util.Vector;
import minutiaeSuperClasses.MinutiaListSuperClass;
import minutiaeSuperClasses.MinutiaException;
import minutiaeSuperClasses.MinutiaeListException;

```

```

/**
 * Class representing minutiae list extracted from <i>MINDTCT</i> text file.
 * Extended from <code>minutiaeSuperClasses.MinutiaeListSuperClass</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class MinutiaeList extends MinutiaListSuperClass {

```

```

    /**
     * Creates a new instance of MinutiaeList.
     *
     * @throws MinutiaeListException If fails to create a new list (e.g error in
     *                               new java.util.Vector parameters).
     */
    public MinutiaeList() throws MinutiaeListException {

        try {
            super.list= new Vector(super.initialCapacity,
                                   super.capacityIncrement);
        }
        catch (IllegalArgumentException iae) {
            throw new MinutiaeListException("Error creating vector: \n\t" +
                                             iae.getMessage());
        }
    }

```

```

}

/**
 * Add a new minutia to this minutiae list.
 *
 * @param min Minutia to insert.
 */
public void addMinutia(Minutia min) {

    list.add(min);
    super.elementNumber++;

}

/**
 * Gets minutia in position index from this minutia list.
 *
 * @param index Index of minutia requested.
 * @return Minutia in position index from list.
 *
 * @throws MinutiaeListException Error getting minutia from list (e.g. index
 *                                out of bounds).
 */
public Minutia getMinutia(int index) throws MinutiaeListException {

    Minutia retMin= null;
    try {
        retMin= (Minutia) (list.get(index));
    }
    catch (ArrayIndexOutOfBoundsException aioobe) {
        throw new MinutiaeListException("Error getting " + index + "
                                         element: \n\t" +
                                         aioobe.getMessage());
    }
    return (retMin);

}

/**
 * Finds a minutia from this list starting from its coordinates.
 *
 * @param x Minutia x coordinate.
 * @param y Minutia y coordinate.
 * @return Minutia found with coordinates x y.
 *
 * @throws MinutiaeListException If fails to scan minutiae list (e.g. index
 *                                out of bounds).
 */
public Minutia findIdMinutia(short x, short y) throws MinutiaeListException
{

    for (int i= 0; i < super.elementNumber; i++) {
        Minutia ext= null;
        try {
            ext= (Minutia) (super.list.get(i));
        }
        catch (ArrayIndexOutOfBoundsException aioobe) {
            throw new MinutiaeListException("Error getting " + i + "
                                             element: \n\t" +
                                             aioobe.getMessage());
        }
    }
}

```

```

        }
        if ((ext.getXPos() == x) && (ext.getYPos() == y))
            return (ext);
    }

    // No minutia found case
    return (null);
}

/**
 * Prints this minutiae list.
 *
 * @throws MinutiaeListException If fails to print list (e.g. error scanning
 *                                list).
 */
public void print() throws MinutiaeListException {

    int size= super.elementNumber;
    System.out.println("There are " + size + " minutiae in the list.\n");
    Minutia rifMin= null;
    for (int i= 0; i < size; i++) {
        try {
            rifMin= (Minutia) (super.list.get(i));
        }
        catch (ArrayIndexOutOfBoundsException aioobe) {
            throw new MinutiaeListException("Error getting " + i + "
                                             element: \n\t" +
                                             aioobe.getMessage());
        }

        try {
            rifMin.print();
        }
        catch (MinutiaException me) {
            throw new MinutiaeListException("Error printing minutia " +
                                             rifMin.getId() + ": \n\t" +
                                             me.getMessage());
        }
        System.out.println("");
    }
}

}

```

```

/*
 * Neighbor.java
 *
 * Created on 11 ottobre 2002, 19.16
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeInformation;

```

```

/***** IMPORTS *****/

```

```

import minutiaeSuperClasses.NeighborSuperClass;
import minutiaeSuperClasses.NeighborException;

```

```

/**

```

```

 * Class representing neighbor extracted from <i>MINDTCT</i> text file.<p>
 * Extended from abstract class in <code>
 * minutiaeSuperClasses.NeighborSuperClass</code>.

```

```

 * @author Francesco Santini
 * @version 1.0

```

```

 */
public class Neighbor extends NeighborSuperClass {

```

```

    /**
     * Neighbor x position.
     */

```

```

    private short xPos= 0;

```

```

    /**
     * Neighbor y position.
     */

```

```

    private short yPos= 0;

```

```

    /**
     * Creates a new instance of Neighbor.
     */

```

```

    public Neighbor() {
    }

```

```

    // Methods to set fields

```

```

    /**
     * Controls and sets x position for this minutia.

```

```

*
* @param newXPos Neighbor x position to be set.
*
* @throws NeighborException If it is a not valid x position (e.g. less than
*                             zero)
*/
public void setXPos(short newXPos) throws NeighborException {
    if (newXPos < 0) {
        throw new NeighborException ("Negative x position (" + newXPos +
                                     ")");
    }
    else {
        this.xPos= newXPos;
    }
    return;
}

/**
 * Controls and sets y position for this minutia.
 *
 * @param newXPos Neighbor y position to be set.
 *
 * @throws NeighborException If it is a not valid y position (e.g. less than
 *                             zero)
 */
public void setYPos(short newYPos) throws NeighborException {
    if (newYPos < 0) {
        throw new NeighborException ("Negative y position (" + newYPos +
                                     ")");
    }
    else {
        this.yPos= newYPos;
    }
    return;
}

// Methods to get fields

/**
 * Returns neighbor x position.
 *
 * @return Neighbor x position.
 */
public short getXPos() {
    return(this.xPos);
}

/**
 * Returns neighbor y position.
 *
 * @return Neighbor y position.
 */
public short getYPos() {
    return(this.yPos);
}

```



```

}

/**
 * Returns a text representing this neighbor.
 *
 * @return String representation of this neighbor.
 */
public String stringRepresentation() {
    String retString= new String("xPos = " + this.xPos + " , yPos = " +
                                this.yPos + " , ridge count = " +
                                this.ridgeCount);
    return (retString);
}
}

```

```

/*
 * NeighBorsList.java
 *
 * Created on 9 ottobre 2002, 22.39
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeInformation;

/***** IMPORTS *****/
import java.util.Vector;
import minutiaeSuperClasses.NeighborListSuperClass;
import minutiaeSuperClasses.NeighborListException;

/**
 * Class representing minutia neighbor list extracted from <i>MINDTCT</i> text
 * file. Extended from <code>minutiaeSuperClasses.NeighborListSuperClass</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class NeighBorsList extends NeighborListSuperClass {

    /**
     * Creates a new instance of NeighBorsList.
     *
     * @throws NeighborListException If fails to create a new list (e.g error in
     * new java.util.Vector parameters).
     */
    public NeighBorsList() throws NeighborListException {

        try {
            super.list= new Vector(super.initialCapacity,
                                   super.capacityIncrement);
        }
        catch (IllegalArgumentException iae) {
            throw new NeighborListException("Error creating vector: \n\t" +
                                             iae.getMessage());
        }
    }
}

```

```

/**
 * Add a new neighbor to this minutia neighbor list.
 *
 * @param nbor Neighbor to insert.
 */
public void add(Neighbor nbor) {

    super.list.add(nbor);
    super.elementNumber++;

}

/**
 * Gets neighbor in position index from this minutia neighbor list.
 *
 * @param index Index of neighbor requested.
 * @return Neighbor in position index from list.
 *
 * @throws NeighborListException Error getting neighbor from list (e.g.
 *                                 index out of bounds).
 */
public Neighbor get(int index) throws NeighborListException {

    Neighbor ret= null;
    try {
        ret= (Neighbor) (super.list.get(index));
    }
    catch (ArrayIndexOutOfBoundsException aioobe) {
        throw new NeighborListException("Error getting " + index + "
                                         element: \n\t" +
                                         aioobe.getMessage());
    }
    return ret;

}

}

```

C.6 Package minutiaeListConverting.

```
/*
 * GlobalInformationFinder.java
 *
 * Created on 11 dicembre 2002, 11.16
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeListConverting;

/***** IMPORTS *****/
import algMinInformation.*;
import minutiaeSuperClasses.MinutiaeListException;
import minutiaeSuperClasses.NeighborListException;

/**
 * Class used to get global information about minutiae list such as maximum
 * ridge count value in minutiae list.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class GlobalInformationFinder {

    /**
     * Creates a new instance of GlobalInformationFinder.
     */
    public GlobalInformationFinder() {

    }

    /**
     * This object finds information about passed minutiae list, as the maximum
     * ridge count value.
     *
     * @param mList list of minutiae converted where methos <code>find</code>
     * finds maximum ridge count value.
     *
     * @throws GlobalInformationFinderException If emthod fails to scan the
     * list.
     */
}
```

```

    */
    public void find(AlgMinutiaeList mList) throws
        GlobalInformationFinderException {

        if (mList == null)
            throw new GlobalInformationFinderException("Null input parameter");

        short maxRidgeCount= 0;
        int sizeMList= mList.size();

        // Searching maximum values in the list
        for (int i= 0; i < sizeMList; i++) {

            AlgMinutia amTemp= null;
            try {
                amTemp= mList.get(i);
            }
            catch (MinutiaeListException mle) {
                throw new GlobalInformationFinderException("Error getting
                                                            minutia " + i + "
                                                            from the list: \n\t"
                                                            + mle.getMessage());
            }

            AlgNeighborList flTemp= amTemp.getAlgNeighborList();
            int sizeFList= flTemp.size();

            // Searching maximum ridge count between neighbors
            for (int j= 0; j < sizeFList; j++) {

                AlgNeighbor nbrTemp= null;
                try {
                    nbrTemp= flTemp.get(j);
                }
                catch (NeighborListException nle) {
                    throw new GlobalInformationFinderException("Error getting "
                                                                + j + " neighbor
                                                                of minutia " + i
                                                                + ": \n\t" +
                                                                nle.getMessage());
                }

                short ridgeCount= nbrTemp.getRidgeCount();
                if (ridgeCount > maxRidgeCount)
                    maxRidgeCount= ridgeCount;

            }

        }

        // Setting found max ridge count
        try {
            mList.setMaxRidgeCount(maxRidgeCount);
        }
        catch (MinutiaeListException mle) {
            throw new GlobalInformationFinderException("Error setting maximum
                                                        neighbor ridge count of
                                                        the list: \n\t" +
                                                        mle.getMessage());
        }

        return;
    }
}

```

```

/*
 * GlobalInformationFinderException.java
 *
 * Created on 11 dicembre 2002, 11.27
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeListConverting;

/**
 * Thrown by <code>minutiaeListConverting.GlobalInformationFinder</code> after
 * failing to retrieve global information about minutiae list, such as maximum
 * ridge count value in the list.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class GlobalInformationFinderException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>GlobalInformationFinderException</code>
     * without detail message.
     */
    public GlobalInformationFinderException() {
    }

    /**
     * Constructs an instance of <code>GlobalInformationFinderException</code>
     * with the specified detail message.
     *
     * @param msg the detail message.
     */
    public GlobalInformationFinderException(String msg) {
        super(msg);
    }
}

```

```

/*
 * MinutiaeListConveter.java
 *
 * Created on 12 ottobre 2002, 10.04
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeListConverting;

```

```

/***** IMPORTS *****/

```

```

import minutiaeInformation.MinutiaeList;
import minutiaeInformation.Minutia;
import minutiaeInformation.MinType;
import minutiaeInformation.Neighbor;
import minutiaeInformation.NeighborsList;
import algMinInformation.AlgMinutiaeList;
import algMinInformation.AlgMinutia;
import minutiaeSuperClasses.MinutiaException;
import algMinInformation.AlgMinutiaType;
import minutiaeSuperClasses.MinTypeException;
import algMinInformation.AlgNeighbor;
import minutiaeSuperClasses.NeighborException;
import minutiaeSuperClasses.NeighborListException;
import minutiaeSuperClasses.MinutiaeListException;
import minutiaeInformation.Minutia;

```

```

/**
 * Class used to convert one minutiae list to another. The first list is derived
 * from <i>MINDTCT</i> information while the second is used by matching
 * algorithm so it has to contain different information.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class MinutiaeListConverter {

```

```

    /**
     * If this flag is true calculates new neighbors, if false uses MINDTCT
     * neighbors.
     */
    private boolean myNeighbors= false;

    /**

```

```

    * If this flag is true output minutiae list is reduced with the help of
    * reliability.
    */
private boolean cutWithReliability= true;

/**
 * Minutiae number in cut list, used if <code>cutWithReliability</code>
 * is <i>true</i>. This filed can be set with method
 * <code>setMinCutWithReliability</code>.
 * Default value is <i>25</i>.
 */
public static final int cutNumber= 25;

/** Creates a new instance of minutiaeListConveter */
public MinutiaeListConverter() {

}

/**
 * This method converts input list and return new matching algorithm
 * compatible list.
 *
 * @param lInput Minutiae list obtained from MINDTCT text file.
 * @return List converted to a new minutiae list.
 *
 * @throws MinutiaeListConverterException If it fails to convert input list
 *                                     (e.g. error finding minutia new
 *                                     neighbors)
 */
public AlgMinutiaeList convert(MinutiaeList lInput) throws
    MinutiaeListConverterException {

    if (this.myNeighbors)
        System.out.println("\nUsing my neighbors.\n");
    else
        System.out.println("\nUsing MINDTCT neighbors.\n");

    // Controlling input parameter
    if (lInput == null)
        throw new MinutiaeListConverterException("Null input parameter.");

    // Creating return minutiae list
    AlgMinutiaeList lOut= null;
    try {
        lOut= new AlgMinutiaeList();
    }
    catch (MinutiaeListException mle) {
        throw new MinutiaeListConverterException("Error creating minutiae
                                                    list: \n\t" +
                                                    mle.getMessage());
    }
    NeighborConverter nbc= new NeighborConverter();

    int s= lInput.size();
    for (int i= 0; i < s; i++) {
        AlgMinutia AlgMinInsert= null;
        try {
            AlgMinInsert= new AlgMinutia();
        }
        catch (MinutiaException me) {
            throw new MinutiaeListConverterException("Error creating new
                                                        minutia: \n\t" +

```



```

me.getMessage());
}

Minutia minCopy= null;
try {
    minCopy= lInput.getMinutia(i);
}
catch (MinutiaeListException mle) {
    throw new MinutiaeListConverterException("Error getting getting
                                             element " + i + ":
                                             \n\t" +
                                             mle.getMessage());
}

// Controlling input parameter
if (lInput == null)
    throw new MinutiaeListConverterException("Null minutia in input
                                             list at position " + i
                                             + ".");

// Copying all the fields
try {
    AlgMinInsert.setId(minCopy.getId());
}
catch (MinutiaException me) {
    throw new MinutiaeListConverterException("Error copying minutia
                                             id (" + minCopy.getId()
                                             + "): \n\t" +
                                             me.getMessage());
}

try {
    AlgMinInsert.setXPos(minCopy.getXPos());
}
catch (MinutiaException me) {
    throw new MinutiaeListConverterException("Error copying minutia
                                             x position (" +
                                             minCopy.getXPos() + "):
                                             \n\t" +
                                             me.getMessage());
}

try {
    AlgMinInsert.setYPos(minCopy.getYPos());
}
catch (MinutiaException me) {
    throw new MinutiaeListConverterException("Error copying y
                                             minutia position (" +
                                             minCopy.getYPos() + "):
                                             \n\t" + me.getMessage());
}

try {
    AlgMinInsert.setDirection(minCopy.getDirection());
}
catch (MinutiaException me) {
    throw new MinutiaeListConverterException("Error copying minutia
                                             direction angle (" +
                                             minCopy.getDirection()
                                             + "): \n\t" +
                                             me.getMessage());
}
}

```

```

try {
    AlgMinInsert.setReliability(minCopy.getReliability());
}
catch (MinutiaeException me) {
    throw new MinutiaeListConverterException("Error copying minutia
                                             reliability (" +
                                             minCopy.getReliability()
                                             + "): \n\t" +
                                             me.getMessage());
}

// Creating new minutia type
AlgMinutiaType AlgMinutiaTypeInsert= null;
try {
    AlgMinutiaTypeInsert= new
        AlgMinutiaType(minCopy.getType().getMinTypeString());
}
catch (MinTypeException mte) {
    throw new MinutiaeListConverterException("Error copying minutia
                                             type (" +
                                             minCopy.getType().getMinTypeString()
                                             + "): \n\t" + mte.getMessage());
}

// Inserting new type
AlgMinInsert.setMinType(AlgMinutiaTypeInsert);

// Inserting MINDTCT neighbors
if (!this.myNeighbors) {

    // Inserting all of the minutiae neighbors
    NeighborsList neighborCopyList= minCopy.getNeighborsList();
    int sizeNeighborCopy= neighborCopyList.size();
    for (int j= 0; j < sizeNeighborCopy; j++) {
        Neighbor neighborToInsert= null;

        try {
            neighborToInsert= neighborCopyList.get(j);
        }
        catch (NeighborListException nle) {
            throw new MinutiaeListConverterException("Error getting
                                                        neighbor " + j
                                                        + " while
                                                        searching my
                                                        neighbors.");
        }

        // Searching neighbor id from its coordinates (x, y
        // positions)
        short xToFind= neighborToInsert.getXPos();
        short yToFind= neighborToInsert.getYPos();
        Minutia minToFind= null;
        try {
            minToFind= lInput.findIdMinutia(xToFind, yToFind);
        }
        catch (MinutiaeListException mle) {
            throw new MinutiaeListConverterException("Error
                                                        searching minutia
                                                        in position " +
                                                        xToFind + yToFind
                                                        + "in minutiae list:
                                                        \n\t" + mle.getMessage());
        }
    }
}

```

```

// Converting neighbor (except ridge count)
AlgNeighbor AlgNeighborInsert= null;
try {
    AlgNeighborInsert= nbc.convert(minCopy, minToFind,
                                   lOut);
}
catch (NeighborConverterException nce) {
    throw new MinutiaeListConverterException("Error
                                             converting neighbors:
                                             \n\t" +
                                             nce.getMessage());
}

// Inserting also ridge count
try {
    AlgNeighborInsert.setRidgeCount
                      (neighborToInsert.getRidgeCount());
}
catch (NeighborException ne) {
    throw new MinutiaeListConverterException("Error copying
                                             neighbor ridge count (" +
                                             minToFind.getId() + "):
                                             \n\t" + ne.getMessage());
}

// Inserting new neighbor in output minutiae list
try {
    AlgMinInsert.addOrderedAlgNeighbor(AlgNeighborInsert);
}
catch (MinutiaException me) {
    throw new MinutiaeListConverterException("Error
                                             inserting neighbor: \n\t"
                                             + me.getMessage());
}
}

// Inserting my neighbors
else {

    NeighborsFinder nf= new NeighborsFinder();
    MinutiaeList neighbors= null;

    // Finding ny neighbors
    try {
        neighbors= nf.findNeighbors (minCopy, lInput);
    }
    catch (NeighborsFinderException nfe) {
        throw new MinutiaeListConverterException("Error finding
                                             minutia neighbor: " +
                                             nfe.getMessage());
    }

    int size= neighbors.size();

    for (int k= 0; k < size; k++) {

        Minutia NghMin= null;
        try {
            NghMin= neighbors.getMinutia(k);
        }
        catch (MinutiaeListException mle) {

```

```

        throw new MinutiaeListConverterException("Error getting
                                                    getting element " + k + ":
                                                    \n\t" + mle.getMessage());
    }

    // Converting neighbor (except ridge count)
    AlgNeighbor AlgNeighborInsert= null;
    try {
        AlgNeighborInsert= nbc.convert(minCopy, NghMin, lOut);
    }
    catch (NeighborConverterException nce) {
        throw new MinutiaeListConverterException("Error
                                                    converting neighbor " +
                                                    k + " at minutia " +
                                                    minCopy.getId() + ":
                                                    \n\t" + nce.getMessage());
    }

    // Ridge count is present only for MINDTCT neighbors, so not
    // in this case: here i find other neighbors. Inserting
    // ridge count equal to zero.
    try {
        AlgNeighborInsert.setRidgeCount((short) (0));
    }
    catch (NeighborException ne) {
        throw new MinutiaeListConverterException("Not expected
                                                    error copying neighbor
                                                    ridge count (0): \n\t"
                                                    + ne.getMessage());
    }

    // Inserting new neighbor in output minutiae list
    try {
        AlgMinInsert.addOrderedAlgNeighbor(AlgNeighborInsert);
    }
    catch (MinutiaException me) {
        throw new MinutiaeListConverterException("Error
                                                    inserting neighbor:
                                                    \n\t" + me.getMessage());
    }
}

// Inserting new minutia in output minutiae list
lOut.add(AlgMinInsert);

}

// If flag is true, cut the list using reliability: keep only minutiae
// with highest reliability
if (this.cutWithReliability) {
    try {
        lOut= lOut.cutByReliability(cutNumber);
    }
    catch (MinutiaeListException mle) {
        throw new MinutiaeListConverterException("Error while cutting
                                                    minutiae list with
                                                    reliability: \n\t" +
                                                    mle.getMessage());
    }
}
}

```

```

        // Retrieving information for the new created list: maximum x,y position
        // and ridge count.
        GlobalInformationFinder gif= new GlobalInformationFinder();
        try{
            gif.find(lOut);
        }
        catch(GlobalInformationFinderException gife) {
            throw new MinutiaeListConverterException("Error while retrieving
                                                    global information about new
                                                    list: \n\t" + gife.getMessage());
        }

        // Returning output minutiae list
        return (lOut);
    }// end of method

    /**
     * Changes the way to find minutiae neighbors. <i>True</i> to use
     * <code>minutiaeListConverting.NeighborsFinder</code> method or
     * <i>false</i> to use neighbors found by <i>MINDTCT</i>.
     *
     * @param flag Flag indicating if using method
     *             <code>minutiaeListConverting.NeighborsFinder</code>.
     */
    public void selectNeighborsSearch(boolean flag) {
        this.myNeighbors= flag;
    }

    /**
     * With this flag set to true only some minutiae with high reliability are
     * taken. See method <code>AlgMinutiaeList.cutByReliability</code>.
     *
     * @param flag True if only some minutiae with high reliability is needed to
     *             be taken.
     */
    public void selectListCutWithReliability(boolean flag) {
        this.cutWithReliability= flag;
    }

    /**
     * You can set number of minutiae in returned cut list, if flag
     * <code>cutWithReliability</code> is <i>true</i>. Sets field
     * <code>cutNumber</code>.
     *
     * @param minNumber Number in cut list.
     */
    public void setMinCutWithReliability(int minNumber) {

    }
}

```

```

/*
 * MinutiaeListConverterException.java
 *
 * Created on 12 ottobre 2002, 12.31
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeListConverting;

/**
 * Exception thrown by <code>MinutiaeListConverter</code> if fails to convert
 * a list of <i>MINDTCT</i> minutiae in a new file format minutiae list.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MinutiaeListConverterException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>MinutiaeListConverterException</code>
     * without detail message.
     */
    public MinutiaeListConverterException() {
    }

    /**
     * Constructs an instance of <code>MinutiaeListConverterException</code>
     * with the specified detail message.
     *
     * @param msg the detail message.
     */
    public MinutiaeListConverterException(String msg) {
        super(msg);
    }
}

```

```

/*
 * NeighborConverter.java
 *
 * Created on 12 ottobre 2002, 22.59
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeListConverting;

```

```

/***** IMPORTS *****/

```

```

import minutiaeInformation.Minutia;
import algMinInformation.AlgNeighbor;
import algMinInformation.AlgMinutiaeList;
import minutiaeSuperClasses.MinutiaeListException;
import minutiaeSuperClasses.NeighborException;

```

```

/**
 * Converts minutia eighbors from MINDTCT format to new matching algorithm
 * compatible format.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```

class NeighborConverter {

```

```

    /**
     * Creates a new instance of NeighborConverter.
     */
    public NeighborConverter() {
    }

```

```

    /**
     * This method calculates new matching algorithm compatible neighbor,
     * starting from one central minutia and one minutia belonging to central
     * minutia neighborhood.
     *
     * @param minCenter Central minutia.
     * @param minNeighbor Minutia in central minutia neighborhood.
     * @param minList minCenter and minNeighbor minutia list.
     * @return Neighbor compatible with matching algorithm.
     */

```

```

*
* @throws NeighborConverterException If it fails to create a matching
*                                     algorithm compatible neighbor.
*/
public AlgNeighbor convert(Minutia minCenter, Minutia minNeighbor,
                          AlgMinutiaeList minList) throws
                          NeighborConverterException {

    if ((minCenter == null) || (minNeighbor == null))
        throw new NeighborConverterException("Null input parameter");

    AlgNeighbor resultAlgNeighbor= new AlgNeighbor();

    // Copying id
    try {
        resultAlgNeighbor.setId(minNeighbor.getId());
    }
    catch (NeighborException ne) {
        throw new NeighborConverterException("Error copying neighbor id (" +
                                             minNeighbor.getId() + ")");
    }

    //System.out.println("Central minutia angle= " +
                        minCenter.getDirection() + " and neighbor angle= "
                        + minNeighbor.getDirection());

    // Retrieving minutiae positions
    short y2= minNeighbor.getYPos();
    short x2= minNeighbor.getXPos();
    short y1= minCenter.getYPos();
    short x1= minCenter.getXPos();

    // Getting minutiae direction angle
    float centerDirection= (minCenter.getDirection()) % 360;
    float neighborDirection= (minNeighbor.getDirection()) % 360;

    // Making the angle positive
    if (centerDirection < 0)
        centerDirection+= 360;

    if (neighborDirection < 0)
        neighborDirection+= 360;

    // Euclidean distance between minutiae
    double xDiff= (double)(x2 - x1);
    xDiff= xDiff * xDiff;
    double yDiff= (double)(y2 - y1);
    yDiff= yDiff * yDiff;
    float distance= (float) (Math.sqrt(xDiff + yDiff));
    try {
        resultAlgNeighbor.setDistance(distance);
    }
    catch (NeighborException ne) {
        throw new NeighborConverterException("Error copying neighbour
                                             distance (" + distance + ")");
    }

    try {
        if (distance > minList.getMaxNeighDistance())
            minList.setMaxNeighDistance(distance);
    }
    catch (MinutiaeListException mle) {
        throw new NeighborConverterException("Error setting new max distance

```



```

                                between central minutia and its
                                neighbor in conversion.");
}

// Calculating relative ridge angle between minutiae
float relativeRidgeAngle= centerDirection - neighborDirection;

/*if (relativeRidgeAngle < 0)
    relativeRidgeAngle= -relativeRidgeAngle;*/

// Angle distance can be only between 0 and 180 degrees
/*if (relativeRidgeAngle > 180)
    relativeRidgeAngle= 360 - relativeRidgeAngle;*/
relativeRidgeAngle= relativeRidgeAngle % 360;

//System.out.println("Relative ridge angle calculated= " +
                    relativeRidgeAngle);

// Inserting relative ridge angle
try {
    resultAlgNeighbor.setRelativeAngle(relativeRidgeAngle);
}
catch (NeighborException ne) {
    throw new NeighborConverterException("Error copying neighbor
                                        relative ridge angle (" +
                                        relativeRidgeAngle + ")");
}

// Calculating distance-relative angle between minutiae
float distanceRelativeAngle= 0;
float segmentAngle= 0;
float divisor= (float) (x2 - x1);
float dividend= (float) (y2 - y1);

// Controlling possible errors
if ((divisor == 0) && (dividend == 0))
    throw new NeighborConverterException("Unexpected error: central
                                        minutia and its neighbor are
                                        the same point");

// Calculating angular coefficient of the segment connecting central
// minutia and its neighbor
if (divisor == 0) {
    if (dividend > 0)
        segmentAngle= 90;
    else
        segmentAngle= 270;
}
else
    if (dividend == 0) {
        if (divisor > 0)
            segmentAngle= 0;
        else
            segmentAngle= 180;
    }
    else {
        float m= (float) (dividend / divisor);
        //System.out.println("Minutia Center " + minCenter.getId() + " ,
                            minutia Neighbor = " + minNeighbor.getId()
                            + " M = " + m);
        //System.out.println("DIVIDEND = " + dividend + " , DIVISOR = "

```

```

        //          + divisor + " , M = " + m);
        segmentAngle= (float) (Math.toDegrees(Math.atan(m)));
        //System.out.println("ArcTang " + distanceRelativeAngle);

        // Adapting found angle: above finds only angle between segment
        // and positive x direction
        if ((divisor > 0.0) && (dividend < 0))
            segmentAngle= (360 - segmentAngle) % 360;
        else
            if ((divisor < 0) && (dividend > 0))
                segmentAngle= (180 - segmentAngle) % 360;
            else
                if ((divisor < 0) && (dividend < 0))
                    segmentAngle= (180 - segmentAngle) % 360;
    }

    // Distance Relative Angle difference
    distanceRelativeAngle= segmentAngle - centerDirection;

    /*
    if (distanceRelativeAngle < 0)
        distanceRelativeAngle= -distanceRelativeAngle;*/

    // Angle distance can be only between 0 and 180 degrees
    /*
    if (distanceRelativeAngle > 180)
        distanceRelativeAngle= 360 - distanceRelativeAngle;*/
    distanceRelativeAngle= distanceRelativeAngle % 360;

    try {
        resultAlgNeighbor.setDistanceRelativeAngle(distanceRelativeAngle);
    }
    catch (NeighborException ne) {
        throw new NeighborConverterException("Error copying neighbor
                                             distance-relative angle (" +
                                             relativeRidgeAngle + ")");
    }

    // Fifth feature
    resultAlgNeighbor.fifthFeature= 0;

    // Returning new neighbor
    return (resultAlgNeighbor);
}
}

```

```

/*
 * NeighborConverterException.java
 *
 * Created on 12 ottobre 2002, 23.20
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeListConverting;

/**
 * This Exception is thrown by <code>NeighborConverter</code> if fails to
 * convert a neighbor from MINDTCT format to new file format neighbor.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class NeighborConverterException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>NeighborConverterException</code> without
     * detail message.
     */
    public NeighborConverterException() {
    }

    /**
     * Constructs an instance of <code>NeighborConverterException</code> with
     * the specified detail message.
     *
     * @param msg the detail message.
     */
    public NeighborConverterException(String msg) {
        super(msg);
    }
}

```

```

/*
 * NeighborsFinder.java
 *
 * Created on 21 ottobre 2002, 5.43
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package minutiaeListConverting;
```

```
/****** IMPORTS *****/
```

```
import minutiaeInformation.MinutiaeList;
import minutiaeInformation.Minutia;
import minutiaeSuperClasses.MinutiaeListException;
```

```
/**
```

```
 * This class implements functionalities to find minutiae neighbors if you don't
 * want to use default <i>MINDTCT</i> neighbors in <i>MINDTCT</i> file.
 *
```

```
 * @author Francesco Santini
```

```
 * @version 1.0
```

```
 */
```

```
public class NeighborsFinder {
```

```
    /**
```

```
     * Number of neighbors to find.
```

```
    */
```

```
    private static final short nn= 5;
```

```
    /**
```

```
     * <i>"Infinite distance"</i>: it'a higher bound.
```

```
    */
```

```
    private static final short max= 10000;
```

```
    /**
```

```
     * Reliability lower bound: only minutiae with reliability above this field
     * value are inserted in new minutiae list.
```

```
    */
```

```
    private static final float minReliability= (float) (0.2);
```

```
    /** Creates a new instance of NeighborsFinder */
```

```
    public NeighborsFinder() {
    }

```

```

/**
 * Returns a list of <code>nn</code> neighbors for passed minutia: needed if
 * not using <i>MINDTCT</i> neighbors.
 *
 * @param min Central minutia passed.
 * @param Minutiae list where searching neighbors.
 * @return Neighbors list of passed central minutia.
 *
 * @throws NeighborsFinderException If it fails to find a list of neighbors
 *                                     (e.g. <code>nn</code> is greater than
 *                                     input minutiae list size).
 */
public MinutiaeList findNeighbors(Minutia min, MinutiaeList list) throws
    NeighborsFinderException {

    // Result minutiae list
    MinutiaeList returnList= null;
    try {
        returnList= new MinutiaeList();
    }
    catch (MinutiaeListException mle) {
        throw new NeighborsFinderException("Error creating minutiae list:
                                           \n\t" + mle.getMessage());
    }

    // Controlling input parameters
    if ((min== null) || (list == null))
        throw new NeighborsFinderException("Null input parameters");

    // Retrieving central minutia position
    short xMin= min.getXPos();
    short yMin= min.getYPos();

    int size= list.size();
    if (size < nn)
        throw new NeighborsFinderException("Impossible to find " + this.nn +
                                           "nearest neighbors: " + size + "
                                           list size");

    float disVector []= new float[size];

    // Building distance vector between minutiae
    for (int i= 0; i < size; i++) {

        // Setting limit distance between minutia and itself
        if ((min.getId()) == (i)) {
            disVector[i]= this.max;
            continue;
        }
        Minutia minTemp= null;
        try {
            minTemp= list.getMinutia(i);
        }
        catch (MinutiaeListException mle) {
            throw new NeighborsFinderException("Error getting " + i + "
                                                element from minutiae list :
                                                \n\t" + mle.getMessage());
        }

        short xTemp= minTemp.getXPos();
        short yTemp= minTemp.getYPos();
    }
}

```

```

        // Euclidean distance between minutiae
        double xDiff= (double) (xMin - xTemp);
        xDiff= xDiff * xDiff;
        double yDiff= (double) (yMin - yTemp);
        yDiff= yDiff * yDiff;
        float distance= (float) (Math.sqrt(xDiff + yDiff));
        disVector[i]= distance;
    }

    boolean neighNotCompleted= false;

    // Searching "nn" number of nearest minutiae with reliability above
    // "minReliability"
    for (int count= 0; count < nn; count++) {

        int minPos= -1;
        float minDis= max;
        Minutia insMin= null;

        for (int k= 0; k < size; k++) {

            Minutia tempMin= null;
            try {
                tempMin= list.getMinutia(k);
            }
            catch (MinutiaeListException mle) {
                throw new NeighborsFinderException("Error getting " + k + "
                                                    element from minutiae
                                                    list : \n\t" +
                                                    mle.getMessage());
            }

            // Set cond to TRUE if not interested in reliability
            boolean cond= !(tempMin.getReliability() < minReliability);
            if ((disVector[k] < minDis) && (cond)) {
                minPos= k;
                minDis= disVector[k];
                insMin= tempMin;
            }
        }

        // No enough neighbors with high reliability in list (must be at
        // lest nn neighbors)
        if (minPos == -1) {
            neighNotCompleted= true;
            break;
        }

        // Setting high distance for minutia just taken
        disVector[minPos]= max;

        returnList.addMinutia(insMin);
    }

    // If there are not "nn" neighbors with reliability above
    // "minReliability", i fill the list with nearest neighbors not yet
    // taken without considering reliability.
    if (neighNotCompleted) {

        int nToInsert= nn - returnList.size();

        for (int count= 0; count < nToInsert; count++) {

```

```

int minPos= -1;
float minDis= max;
Minutia insMin= null;

for (int k= 0; k < size; k++) {

    Minutia tempMin= null;
    try {
        tempMin= list.getMinutia(k);
    }
    catch (MinutiaeListException mle) {
        throw new NeighborsFinderException("Error getting " + k
                                           + " element from
                                           minutiae list : \n\t"
                                           + mle.getMessage());
    }

    if (disVector[k] < minDis) {
        minPos= k;
        minDis= disVector[k];
        insMin= tempMin;
    }
}

// No enough neighbors in list (impossible)
if (minPos == -1)
    throw new NeighborsFinderException("Impossible to find " +
                                       this.nn + " nearest
                                       neighbors (Impossible
                                       error!)");

// Setting high distance for minutia just taken
disVector[minPos]= max;
returnList.addMinutia(insMin);

}

}

// Returning result list with "this.nn" nearest neighbors
return (returnList);

}

}

```

```

/*
 * NeighborsFinderException.java
 *
 * Created on 16 novembre 2002, 3.03
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeListConverting;

/**
 * Thrown by <code>NeighborsFinder</code> after failing to find alternative minutiae
 * neighbors.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class NeighborsFinderException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>NeighborsFinderException</code> without
     * detail message.
     */
    public NeighborsFinderException() {
    }

    /**
     * Constructs an instance of <code>NeighborsFinderException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public NeighborsFinderException(String msg) {
        super(msg);
    }
}

```


C.7 Package minutiaeSuperClasses.

```
/*
 * MinTypeException.java
 *
 * Created on 10 ottobre 2002, 23.12
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeSuperClasses;

/**
 * Exception thrown by methods in <code>minutiaeSuperClasses.MinTypeSuperClass
 * </code>, <code>minutiaeInformation.MinType</code> and
 * <code>algMinutiaInformation.AlgMinutiaType</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MinTypeException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>MinTypeException</code> without detail
     * message.
     */
    public MinTypeException() {
    }

    /**
     * Constructs an instance of <code>MinTypeException</code> with the
     * specified detail message.
     * @param msg the detail message.
     */
    public MinTypeException(String msg) {
        super(msg);
    }
}
```

```

/*
 * MinTypeSuperClass.java
 *
 * Created on 18 novembre 2002, 11.44
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeSuperClasses;

```

```

/**
 * Class representing minutia type: "BIF" stands fo bifurcation and "RIG" for
 * ridge ending. This class is extended by
 * <code>minutiaeInformation.MinType</code> and
 * <code>algMinInformation.AlgMinutiaType</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```

public abstract class MinTypeSuperClass {

```

```

    /**
     * Bifurcation type.
     */

```

```

    protected static final String bifString= "BIF";

```

```

    /**
     * Ridge-ending type.
     */

```

```

    protected static final String ridgeEndingString= "RIG";

```

```

    /**
     * No type inserted.
     */

```

```

    protected static final String noType= "NO TYPE";

```

```

    /**
     * Integer representation of type.
     */

```

```

    protected byte type= -1;

```

```

    /**
     * Creates a new instance of MinTypeSuperClass
     */

```

```

    * @param typeString String representing minutia type: can be <i>"BIF"
    * "RIG"</i> or <i>"NO TYPE"</i>.

```

```

*
* @throws MinTypeException If passed string is not one of those above.
*/
public MinTypeSuperClass(String typeString) throws MinTypeException {

    if (typeString.equalsIgnoreCase(bifString))
        this.type= 0;
    else
        if (typeString.equalsIgnoreCase(ridgeEndingString))
            this.type= 1;
        else
            throw new MinTypeException("Minutia Type not allowed (" +
                                        typeString + ")");
}

/** Creates a new instance of MinTypeSuperClass */
public MinTypeSuperClass() {}

/**
 * Returns a byte value representing minutia type.
 *
 * @return Byte representing type
 */
public byte getMinType() {

    return (this.type);

}

/**
 * Method returning type string.
 *
 * @return Type string: "BIF", "RIG", "NO TYPE".
 */
public String getMinTypeString() {

    switch (this.type) {
        case 0:
            return (bifString);
        case 1:
            return (ridgeEndingString);
        default:
            return (noType);
    }
}

/**
 * Sets type of this minutia type.
 *
 * @param typeString String representing minutia type: can be <i>"BIF"
 *                  "RIG"</i> or <i>"NO TYPE"</i>.
 *
 * @throws MinTypeException If passed string is not one of those above
 *                  (<i>"BIF", RIG"</i> or <i>"NO TYPE"</i>).
 */
public void setType(String typeString) throws MinTypeException {

    if (typeString.equalsIgnoreCase(bifString))
        this.type= 0;
    else
        if (typeString.equalsIgnoreCase(ridgeEndingString))
            this.type= 1;

```

```

        else
            throw new MinTypeException("Minutia Type not allowed");
    }

    /**
     * Returns a boolean true if type is bifurcation type
     *
     * @return <i>True</i> if is a "BIF" type, <i>false</i> otherwise.
     */
    public boolean isBifurcation() {
        return (this.type == 0);
    }

    /**
     * Returns a boolean true if type is ridge-ending type
     *
     * @return <i>True</i> if is a "RIG" type, <i>false</i> otherwise.
     */
    public boolean isRidgeEnding() {
        return (this.type == 1);
    }

    /**
     * Returns <i>true</i> if this minutia type and minutia type passed
     * are the same.
     *
     * @param anotherMinType Type to confront with this.
     * @return <i>True</i> if type is the same.
     */
    public boolean equalType(MinTypeSuperClass anotherMinType) {
        return (this.type == anotherMinType.getMinType());
    }
}

```

```

/*
 * MinutiaeListException.java
 *
 * Created on 18 novembre 2002, 19.05
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeSuperClasses;

/**
 * Thrown by methods in <code>algMinInformation.AlgMinutiaeList</code> and
 * <code>minutiaeInformation.MinutiaeList</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MinutiaeListException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>MinutiaeListException</code> without
     * detail message.
     */
    public MinutiaeListException() {
    }

    /**
     * Constructs an instance of <code>MinutiaeListException</code> with the
     * specified detail message.
     * @param msg the detail message.
     */
    public MinutiaeListException(String msg) {
        super(msg);
    }
}

```

```

/*
 * MinutiaException.java
 *
 * Created on 11 ottobre 2002, 15.15
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeSuperClasses;

/**
 * Exception thrown by methods in
 * <code>minutiaeSuperClasses.MinutiaSuperClass</code>,
 * <code>minutiaeOnformation.Minutia</code> and
 * <code>algMinInformation.AlgMinutia</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class MinutiaException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>MinutiaException</code> without detail
     * message.
     */
    public MinutiaException() {
    }

    /**
     * Constructs an instance of <code>MinutiaException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public MinutiaException(String msg) {
        super(msg);
    }
}

```

```

/*
 * MinutiaeListSuperClass.java
 *
 * Created on 18 novembre 2002, 16.33
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package minutiaeSuperClasses;
```

```

/***** IMPORTS *****/
import java.util.Vector;

```

```

/**
 * Represents a list of minutiae. Extended by
 * <code>minutiaeInformation.MinutiaeList</code> and
 * <code>algMinInformation.AlgMinutiaeList</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public abstract class MinutiaeListSuperClass {

    /**
     * Number of minutiae in this list
     */
    protected int elementNumber= 0;

    /**
     * Data structure for minutiae list.
     */
    protected Vector list= null;

    /**
     * Vector initial capacity: see class Vector in API
     */
    protected static final int initialCapacity= 30;

    /**
     * Vector capacityIncrement: see class Vector in API
     */
    protected static final int capacityIncrement= 0;

    /** Creates a new instance of MinutiaeListSuperClass */
    public MinutiaeListSuperClass() {

```

```

        this.list= new Vector();
    }

    /**
     * Returns minutiae number in this list.
     *
     * @return Number of minutiae in this list.
     */
    public int size() {
        return (list.size());
    }
}

```



```

/*
 * MinutiaSuperClass.java
 *
 * Created on 18 novembre 2002, 14.24
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package minutiaeSuperClasses;
```

```

/**
 * Class representing minutia. It is super class of
 * <code>minutiaInformation.Minutia</code> and
 * <code>algMinInformation.AlgMinutia</code>. For more information about class
 * fields, see <i><b>MINDTCT.txt</b></i> attached to this project (row 98).
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public abstract class MinutiaSuperClass {
```

```

    /**
     * Minutia identifier
     */
    protected short id= 0;

    /**
     * Minutia x position
     */
    protected short xPos= 0;

    /**
     * Minutia y position
     */
    protected short yPos= 0;

    /**
     * Minutia ridge angle
     */
    protected float direction= 0;

    /**
     * Minutia reliability
     */
    protected float reliability= 0;

```

```

/**
 * Creates a new instance of MinutiaSuperClass
 */
public MinutiaSuperClass() {
}

// Methods to set fields

/**
 * Controls and sets identifier for this minutia.
 *
 * @param newId Minutia identifier to be set.
 *
 * @throws MinutiaException If it is a not valid identifier (e.g. less than
 *                           zero)
 */
public void setId(short newId) throws MinutiaException {
    if (newId < 0) {
        throw new MinutiaException("Negative minutia identification number
                                   (" + newId + ")");
    }
    else {
        this.id= newId;
    }
    return;
}

/**
 * Controls and sets x position for this minutia.
 *
 * @param newXPos Minutia x position to be set.
 *
 * @throws MinutiaException If it is a not valid x position (e.g. less than
 *                           zero)
 */
public void setXPos(short newXPos) throws MinutiaException {
    if (newXPos < 0) {
        throw new MinutiaException ("Negative x position (" + newXPos +
                                   ")");
    }
    else {
        this.xPos= newXPos;
    }
    return;
}

/**
 * Controls and sets y position for this minutia.
 *
 * @param newYPos Minutia y position to be set.
 *
 * @throws MinutiaException If it is a not valid y position (e.g. less than
 *                           zero)
 */
public void setYPos(short newYPos) throws MinutiaException {

```

```

        if (newYPos < 0) {
            throw new MinutiaException ("Negative y position (" + newYPos +
                                         ")");
        }
        else {
            this.yPos= newYPos;
        }
        return;
    }

    /**
     * Controls and sets direction angle for this minutia.
     *
     * @param newDirection Minutia direction angle to be set.
     *
     * @throws MinutiaException If it is a not valid direction angle (e.g. less
     *                           than zero)
     */
    public void setDirection(float newDirection) throws MinutiaException {

        if ((newDirection < 0.0) || (newDirection > 31.0)) {
            throw new MinutiaException("Direction angle out of range [0..31] ("
                                         + newDirection + ")");
        }
        else {

            // MINDTCT angle is between 0 and 31 starting from up direction
            // Adapting angle
            float insAng= 0;
            insAng= (float) ((newDirection * 11.25) % 360);
            if ((insAng >= 0) && (insAng <= 90))
                insAng-= 90;
            else
                if ((insAng > 90) && (insAng <= 180))
                    insAng= 360 - (insAng - 90);
                else
                    if ((insAng > 180) && (insAng <= 270))
                        insAng= 270 - (insAng - 180);
                    else
                        if ((insAng > 270) && (insAng <= 360))
                            insAng= 90 + (360 - insAng);
                        else
                            throw new MinutiaException("Error converting
                                                            angle");

            this.direction= insAng;
            //System.out.println("Parsed angle " + newDirection + " and
                                converted in " + insAng);
        }
        return;
    }

    /**
     * Controls and sets reliability for this minutia.
     *
     * @param newYPos Minutia y position to be set.
     *
     * @throws MinutiaException If it is a not valid y position (e.g. less than
     *                           zero)
     */

```

```

public void setReliability(float newReliability) throws MinutiaException {
    if ((newReliability < 0.0) || (newReliability > 1.0)) {
        throw new MinutiaException ("Reliability not between 0.0 and 1.0 ("
                                     + newReliability + ")");
    }
    else {
        this.reliability= newReliability;
    }
    return;
}

// Methods to get field values

/**
 * Returns minutia identifier.
 *
 * @return Minutia identifier.
 */
public short getId() {
    return(this.id);
}

/**
 * Returns minutia x position.
 *
 * @return Minutia x position.
 */
public short getXPos() {
    return(this.xPos);
}

/**
 * Returns minutia y position.
 *
 * @return Minutia y position.
 */
public short getYPos() {
    return(this.yPos);
}

/**
 * Returns minutia ridge direction angle..
 *
 * @return Minutia direction angle.
 */
public float getDirection() {
    return(this.direction);
}

```

```
/**
 * Returns minutia reliability.
 *
 * @return minutia reliability.
 */
public float getReliability() {
    return(this.reliability);
}
}
```

```

/*
 * NeighborException.java
 *
 * Created on 11 ottobre 2002, 19.28
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package minutiaeSuperClasses;

```

```

/**
 * Thrown by methods in <code>algMinInformation.Feature</code> and
 * <code>minutiaeInformation.Neighbor</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class NeighborException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>NeighborException</code> without detail
     * message.
     */
    public NeighborException() {
    }

    /**
     * Constructs an instance of <code>NeighborException</code> with the
     * specified detail message.
     * @param msg the detail message.
     */
    public NeighborException(String msg) {
        super(msg);
    }
}

```

```

/*
 * NeighborListException.java
 *
 * Created on 18 novembre 2002, 17.42
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeSuperClasses;

/**
 * Thrown by methods in <code>algMinInformation.AlgNeighborList</code> and
 * <code>minutiaeSuperClasses.NeighborListException</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class NeighborListException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>NeighborListException</code> without
     * detail message.
     */
    public NeighborListException() {
    }

    /**
     * Constructs an instance of <code>NeighborListException</code> with the
     * specified detail message.
     * @param msg the detail message.
     */
    public NeighborListException(String msg) {
        super(msg);
    }
}

```

```

/*
 * NeighborListSuperClass.java
 *
 * Created on 18 novembre 2002, 17.04
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package minutiaeSuperClasses;
```

```

/***** IMPORTS *****/
import java.util.Vector;

```

```

/**
 * Represents a list of minutia neighbors. Extended by
 * <code>minutiaeInformation.NeighborList</code> and
 * <code>algMinInformation.FeatureList</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public abstract class NeighborListSuperClass {

    /**
     * Number of neighbors in this list
     */
    protected int elementNumber= 0;

    /**
     * Data structure for neighbor list.
     */
    protected Vector list= null;

    /**
     * Vector initial capacity: see class Vector in API
     */
    protected static final int initialCapacity= 5;

    /**
     * Vector capacityIncrement: see class Vector in API
     */
    protected static final int capacityIncrement= 0;

    /** Creates a new instance of NeighborListSuperClass */
    public NeighborListSuperClass() {

```



```
}  
  
/**  
 * Returns neighbors number in this list.  
 *  
 * @return Number of neighbors in this list.  
 */  
public int size() {  
    return (list.size());  
}  
  
}
```

```

/*
 * NeighborSuperClass.java
 *
 * Created on 18 novembre 2002, 16.49
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package minutiaeSuperClasses;

/**
 * Abstract super class for minutia neighbor. Coded only to be extended by
 * minutiaeInformation.Neighbor and
 * algMinInformation.Feature.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public abstract class NeighborSuperClass {

    /**
     * Ridge count between neighbor and its central minutia.
     */
    protected short ridgeCount= 0;

    /**
     * Creates a new instance of NeighborSuperClass
     */
    public NeighborSuperClass() {
    }

    /**
     * Controls and sets ridge count between this neighbor and its central
     * minutia.
     *
     * @param newRidgeCount Neighbor ridge count to be set.
     *
     * @throws NeighborException If it is a not valid newRidgeCount (e.g. less
     * than zero)
     */
    public void setRidgeCount(short newRidgeCount) throws NeighborException {
        if (newRidgeCount < 0) {
            throw new NeighborException("Negative ridge count (" + newRidgeCount

```

```

+ "));
    }
    else {
        this.ridgeCount= newRidgeCount;
    }
}

/**
 * Returns neighbor ridge count.
 *
 * @return Neighbor ridge count.
 */
public short getRidgeCount() {
    return(this.ridgeCount);
}

}

```

C.8 Package parsing.

```
/*
 * LineParser.java
 *
 * Created on 10 ottobre 2002, 20.52
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
```

```
package parsing;
```

```
/****** IMPORTS *****/
import java.util.StringTokenizer;
import java.util.NoSuchElementException;
import minutiaeInformation.MinutiaeList;
import minutiaeInformation.Minutia;
import minutiaeInformation.MinType;
import minutiaeSuperClasses.MinTypeException;
import minutiaeSuperClasses.MinutiaException;
import minutiaeInformation.FeatureType;
import minutiaeInformation.FeatureTypeException;

/**
 * Helps <code>Parser</code> during parsing: parses on text line from MINDTCT
 * file. Uses <code>NeighborsParser</code> to parse minutiae neighbors.
 *
 * @author Francesco Santini
 * @version 1.0
 */
```

```
class LineParser {

    /**
     * First information separator ":".
     */
    private static final String sep1= ":";

    /**
     * Second Information separator ";".
     */
```

```

private static final String sep2= ",";

/**
 * Creates a new instance of LineParser.
 */
public LineParser() {
}

/**
 * Even <code>LineParser.lineParse</code> throws <code>ParserException</code>
 * like <code>Parser.parse</code>. This method parses one line of text,
 * builds a new minutia from retrieved information and inserts it in the
 * list.
 *
 * @param line The text line to parse
 * @param ml Inserts here parsed minutia
 *
 * @throws ParserException If can't parse the line (e.g. error on input
 *                          minutia identifier: not in sequence).
 */
public void lineParse(String line, MinutiaeList ml) throws ParserException {
    if ((line == null) || (ml == null))
        throw new ParserException("Null input parameter.");

    // Removing spaces from the line
    String myLine= new String("");
    StringTokenizer blankRemoveTokenizer= new StringTokenizer(line);

    // New minutia to be inserted
    Minutia min= null;
    try {
        min= new Minutia();
    }
    catch (MinutiaException me) {
        throw new ParserException("Error creating new minutia : \n\t" +
                                   me.getMessage());
    }

    try {
        while (blankRemoveTokenizer.hasMoreTokens())
            myLine= myLine.concat(blankRemoveTokenizer.nextToken());

        // method replaceAll not present in 1.2 API
        //String myLine= line.replaceAll(" ", "");

        // Dividing line into tokens
        StringTokenizer lineTokenizer= new StringTokenizer(myLine, ".,;",
                                                            true);

        // Retrieving minutia id
        short id= 0;
        if (lineTokenizer.hasMoreTokens()) {
            Short idTemp= null;
            String tokenString= null;
            try {
                tokenString= lineTokenizer.nextToken();
                idTemp= new Short(tokenString);
            }
            catch (NumberFormatException nfe) {
                throw new ParserException("Invalid number format (" +

```

```

        tokenString + ").");
    }
    id= idTemp.shortValue();
    if (id != ml.size())
        throw new ParseException("Error on minutia identification
                                number (not a sequence).");
}
else
    throw new ParseException("Unexpected line termination.");

// Inserting id in minutia field
try {
    min.setId(id);
}
catch (MinutiaException me) {
    throw new ParseException("Error setting id at minutia " + id +
                            ": \n\t" + me.getMessage());
}

// Controlling separator(sep1)
if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep1.equalsIgnoreCase(lineTokenizer.nextToken()))
        throw new ParseException("Missing separator " + this.sep1 +
                                " at minutia " + id + ".");
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Retrieving minutia x position
short xPos= 0;
if (lineTokenizer.hasMoreTokens()) {
    Short xPosTemp= null;
    String tokenString= null;
    try {
        tokenString= lineTokenizer.nextToken();
        xPosTemp= new Short(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new ParseException("Invalid number format (" +
                                tokenString + ") at minutia " + id
                                + ".");
    }
    xPos= xPosTemp.shortValue();
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Inserting x position in minutia field
try {
    min.setXPos(xPos);
}
catch (MinutiaException me) {
    throw new ParseException("Error setting x position at minutia "
                            + id + ": " + xPos + "\n\t" +
                            me.getMessage());
}

// Controlling separator(sep2)

```

```

if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep2.equalsIgnoreCase(lineTokenizer.nextToken()))
        throw new ParseException("Missing separator " + this.sep2 +
                                " at minutia " + id + ".");
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Retrieving minutia y position
short yPos= 0;
if (lineTokenizer.hasMoreTokens()) {
    Short yPosTemp= null;
    String tokenString= null;
    try {
        tokenString= lineTokenizer.nextToken();
        yPosTemp= new Short(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new ParseException("Invalid number format (" +
                                tokenString + ") at minutia " + id
                                + ".");
    }
    yPos= yPosTemp.shortValue();
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Inserting y position in minutia field
try {
    min.setYPos(yPos);
}
catch (MinutiaException me) {
    throw new ParseException("Error setting y position at minutia "
                            + id + ": " + yPos + " \n\t" +
                            me.getMessage());
}

// Controlling separator (sep1)
if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep1.equalsIgnoreCase(lineTokenizer.nextToken()))
        throw new ParseException("Missing separator " + this.sep1 +
                                " at minutia " + id + ".");
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Retrieving minutia direction angle
float direction= 0;
if (lineTokenizer.hasMoreTokens()) {
    Float directionTemp= null;
    String tokenString= null;
    try {
        tokenString= lineTokenizer.nextToken();
        directionTemp= new Float(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new ParseException("Invalid number format (" +

```

```

                                tokenString + ") at minutia " + id
                                + ".");
        }
        direction= directionTemp.floatValue();
    }
    else
        throw new ParseException("Unexpected line termination at
                                minutia " + id + ".");

    // Inserting direction angle in minutia field
    try {
        min.setDirection(direction);
    }
    catch (MinutiaException me) {
        throw new ParseException("Error setting direction at minutia "
                                + id + ": " + direction + "\n\t" +
                                me.getMessage());
    }

    // Controlling separator (sep1)
    if (lineTokenizer.hasMoreTokens()) {
        if (! this.sep1.equalsIgnoreCase(lineTokenizer.nextToken()))
            throw new ParseException("Missing separator " + this.sep1 +
                                    " at minutia " + id + ".");
    }
    else
        throw new ParseException("Unexpected line termination at
                                minutia " + id + ".");

    // Retrieving minutia reliability
    float reliability= 0;
    if (lineTokenizer.hasMoreTokens()) {
        Float reliabilityTemp= null;
        String tokenString= null;
        try {
            tokenString= lineTokenizer.nextToken();
            reliabilityTemp= new Float(tokenString);
        }
        catch (NumberFormatException nfe) {
            throw new ParseException("Invalid number format (" +
                                    tokenString + ") at minutia " + id
                                    + ".");
        }
        reliability= reliabilityTemp.floatValue();
    }
    else
        throw new ParseException("Unexpected line termination at
                                minutia " + id + ".");

    // Inserting reliability in minutia field
    try {
        min.setReliability(reliability);
    }
    catch (MinutiaException me) {
        throw new ParseException("Error setting reliability at minutia
                                " + id + ": " + reliability + "\n\t" +
                                me.getMessage());
    }

    // Controlling separator (sep1)

```



```

if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep1.equalsIgnoreCase(lineTokenizer.nextToken()))
        throw new ParseException("Missing separator " + this.sep1 +
                                " at minutia " + id + ".");
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Retrieving minutia type
MinType minTypeTemp= null;
if (lineTokenizer.hasMoreTokens()) {
    String typeString= lineTokenizer.nextToken();
    try {
        minTypeTemp= new MinType(typeString);
    }
    catch (MinTypeException mte) {
        throw new ParseException("Error on minutia " + id + "
                                type: " + typeString + " \n\t" +
                                mte.getMessage());
    }
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Inserting type in minutia field
min.setMinType(minTypeTemp);

// Controlling separator (sep1)
if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep1.equalsIgnoreCase(lineTokenizer.nextToken()))
        throw new ParseException("Missing separator " + this.sep1 + "
                                at minutia " + id + ".");
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Retrieving minutia feature type
FeatureType featureTypeTemp= null;
if (lineTokenizer.hasMoreTokens()) {
    String featureString= lineTokenizer.nextToken();
    try {
        featureTypeTemp= new FeatureType(featureString);
    }
    catch (FeatureTypeException fte) {
        throw new ParseException("Error on minutia " + id + "
                                setting feature type: " +
                                featureString + "\n\t" +
                                fte.getMessage());
    }
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Inserting feature type della in minutia field
min.setFeatureType(featureTypeTemp);

```

```

// Controlling separator (sep1)
if (lineTokenizer.hasMoreTokens()) {
    if (! this.sep1.equalsIgnoreCase(lineTokenizer.nextToken()))
        throw new ParseException("Missing separator " + this.sep1 +
                                " at minutia " + id + ".");
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Retrieving integer identifier of the type of feature detected
short featureTypeIdentifier= 0;
if (lineTokenizer.hasMoreTokens()) {
    Short featureTypeIdentifierTemp= null;
    String tokenString= null;
    try {
        tokenString= lineTokenizer.nextToken();
        featureTypeIdentifierTemp= new Short(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new ParseException("Invalid number format (" +
                                tokenString + ") at minutia " + id
                                + ".");
    }
}
else
    throw new ParseException("Unexpected line termination at
                            minutia " + id + ".");

// Inserting integer identifier of the type of feature detected
// in minutia field
try {
    min.setFeatureTypeIdentifier(featureTypeIdentifier);
}
catch (MinutiaException me) {
    throw new ParseException("Error on minutia " + id + " setting
                            feature type identifier: " +
                            featureTypeIdentifier + "\n\t" +
                            me.getMessage());
}

// Parsing all the minutiae neighbors calling NeighborsParser
// There is no fixed number of neighbors: minutiae can contain
// variable number of them
if (lineTokenizer.hasMoreTokens()) {

    // Remebering neighbor parsed.
    int nNeigh= 0;

    try {
        NeighborsParser nb= new NeighborsParser(lineTokenizer, min);
        do {
            nNeigh++;
            if (! this.sep1.equalsIgnoreCase
                (lineTokenizer.nextToken()))
                throw new ParseException("Missing separator " +
                                        this.sep1 + " at minutia "
                                        + id + ".");

            nb.parse();
        }
    }
}

```

```

        } while (lineTokenizer.hasMoreTokens());
    }
    catch (ParserException pe) {
        throw new ParserException("Error parsing neighbor in
                                position " + nNeigh + ":\n\t" +
                                pe.getMessage());
    }
}

}
catch (NoSuchElementException nsee) {
    throw new ParserException("Unexpected no more token error: \n\t" +
                              nsee.getMessage());
}
// Inserting new minutia parsed in minutia list parameter
ml.addMinutia(min);

//System.out.println("Minutia " + id + " successfully inserted.");
}

}

```

```

/*
 * NeighborsParser.java
 *
 * Created on 11 ottobre 2002, 18.19
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package parsing;
```

```

/***** IMPORTS *****/
import java.util.StringTokenizer;
import java.util.NoSuchElementException;
import minutiaeInformation.Minutia;
import minutiaeInformation.Neighbor;
import minutiaeSuperClasses.NeighborException;

```

```

/**
 * Helps <code>LineParser.lineParse</code> parsing a string containing minutiae
 * information. Exactly, method <code>parse</code> of this class, parses a
 * minutia neighbor.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
class NeighborsParser {
```

```

    /**
     * <code>java.util.StringTokenizer</code> used in parsing.
     */
    private java.util.StringTokenizer neighborTokenizer= null;

```

```

    /**
     * Minutia to be filled with its neighbors.
     */
    private Minutia min= null;

```

```

    /**
     * First information separator ", ".
     */
    private static final String sep1= ", ";

```

```

    /**
     * First information separator "; ".

```

```

    */
    private static final String sep2= ";";

    /**
     * Creates a new instance of NeighborsParser.
     *
     * @param newStringTokenizer <code>StringTokenizer</code> created by <code>
     *                               Parser.parse</code>: contains tokenized line
     *                               from input stream.
     * @param newMin <code>Minutia</code> the parser is going to insert in the
     *                               list.
     *
     * @throws ParserException Thrown if it fails to create a minutia neighbor
     *                               parser.
     */
    public NeighborsParser(StringTokenizer newStringTokenizer, Minutia newMin)
        throws ParserException {

        if ((newStringTokenizer == null) || (newMin == null))
            throw new ParserException("Null input parameter");
        this.neighborTokenizer= newStringTokenizer;
        this.min= newMin;
    }

    /**
     * Parse a minutia neighbor.
     *
     * @throws ParserException Thrown if fails to parse a minutia neighbor
     *                               (e.g. invalid format).
     */
    public void parse() throws ParserException {

        Neighbor nbor= new Neighbor();

        try {

            // Retrieving neighbor x position
            short xPos= 0;
            if (neighborTokenizer.hasMoreTokens()) {
                Short xPosTemp= null;
                String tokenString= null;
                try {
                    tokenString= neighborTokenizer.nextToken();
                    xPosTemp= new Short(tokenString);
                }
                catch (NumberFormatException nfe) {
                    throw new ParserException("Invalid number format (" +
                                                tokenString + ") at minutia " +
                                                min.getId() + ".");
                }
                xPos= xPosTemp.shortValue();
            }
            else
                throw new ParserException("Unexpected line termination at minutia "
                                            + min.getId() + ".");

            // Inseriting x position in neighbor field
            try {

```

```

        nbor.setXPos(xPos);
    }
    catch (NeighborException ne) {
        throw new ParseException("Error at minutia " + min.getId() + "
                                setting neighbor x position: " + xPos +
                                "\n\t" + ne.getMessage());
    }

    // Controlling separator (sep1)
    if (neighborTokenizer.hasMoreTokens()) {
        if (! this.sep1.equalsIgnoreCase(neighborTokenizer.nextToken()))
            throw new ParseException("Missing separator " + this.sep1 + "
                                    at minutia " + min.getId() + ".");
    }
    else
        throw new ParseException("Unexpected line termination at minutia "
                                + min.getId() + ".");

    // Retrieving neighbor y position
    short yPos= 0;
    if (neighborTokenizer.hasMoreTokens()) {
        Short yPosTemp= null;
        String tokenString= null;
        try {
            tokenString= neighborTokenizer.nextToken();
            yPosTemp= new Short(tokenString);
        }
        catch (NumberFormatException nfe) {
            throw new ParseException("Invalid number format (" +
                                    tokenString + ") at minutia " +
                                    min.getId() + ".");
        }
        yPos= yPosTemp.shortValue();
    }
    else
        throw new ParseException("Unexpected line termination at minutia "
                                + min.getId() + ".");

    // Inserting y position in neighbor field
    try {
        nbor.setYPos(yPos);
    }
    catch (NeighborException ne) {
        throw new ParseException("Error at minutia " + min.getId() + "
                                setting neighbor y position: " + yPos +
                                "\n\t" + ne.getMessage());
    }

    // Controlling separator (sep2)
    if (neighborTokenizer.hasMoreTokens()) {
        if (! this.sep2.equalsIgnoreCase(neighborTokenizer.nextToken()))
            throw new ParseException("Missing separator " + this.sep1 + "
                                    at minutia " + min.getId() + ".");
    }
    else
        throw new ParseException("Unexpected line termination at minutia "
                                + min.getId() + ".");

    // Retrieving ridge count between minutia and this neighbor

```

```

short ridgeCount= 0;
if (neighborTokenizer.hasMoreTokens()) {
    Short ridgeCountTemp= null;
    String tokenString= null;
    try {
        tokenString= neighborTokenizer.nextToken();
        ridgeCountTemp= new Short(tokenString);
    }
    catch (NumberFormatException nfe) {
        throw new ParseException("Invalid number format (" +
                                tokenString + ") at minutia " +
                                min.getId() + ".");
    }
    ridgeCount= ridgeCountTemp.shortValue();
}
else
    throw new ParseException("Unexpected line termination at minutia "
                            + min.getId() + ".");

// Inserting ridge count in neighbor field
try {
    nbor.setRidgeCount(ridgeCount);
}
catch (NeighborException ne) {
    throw new ParseException("Error at minutia " + min.getId() + "
                            setting neighbor ridge count: " +
                            ridgeCount + "\n\t" + ne.getMessage());
}

}
catch (NoSuchElementException nsee) {
    throw new ParseException("Unexpected no more token error: \n\t" +
                            nsee.getMessage());
}

min.addNeighbor(nbor);
}
}

```

```

/*
 * Parser.java
 *
 * Created on 9 ottobre 2002, 22.18
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package parsing;
```

```

/***** IMPORTS *****/
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.NoSuchElementException;
import java.util.NoSuchElementException;
import minutiaeInformation.MinutiaeList;

/**
 * An object instantiated from this class, handles the <i>MINDTCT</i> file
 * parsing. Gets minutiae number from the first line and reads minutiae
 * information skipping blank lines (even at the top of the file).
 * Parsing stops when has reached valid lines number equal to minutiae number
 * specified in MINDTCT file.<p>
 * <code>Parser</code> uses <code>LineParser</code> to parse a line and throws
 * <code>ParserException</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```

public class Parser {

    /**
     * Number of tokens in MINDTCT text first line.
     */
    private static final int firstLineTokens= 3;

    /**
     * Second string token in MINDTCT text first line.
     */
    private static final String secondTokenFirstLine= "minutiae";

    /**

```



```

    * Third string token in MINDTCT text first line.
    */
private static final String thirdTokenFirstLine= "detected";

/**
 * Empty string "".
 */
private static final String blankString= "";

/** Creates a new instance of Parser */
public Parser() {
}

/**
 * Contains parsing central loop. Parses first line and calls <code>
 * LineParser</code> for each non-empty line.
 *
 * @param fr <code>FileReader</code> for the file to be parsed
 * @param ml Result minutiae list: method <code>Parser.parse</code> puts
 *           here founded minutiae
 *
 * @throws ParserException If <code>Parser</code> has found an Exception
 *           during parsing.
 */
public void parse(FileReader fr, MinutiaeList ml) throws ParserException {

    if ((fr == null) || (ml == null))
        throw new ParserException("Null parameter passed.");

    // Adapting input stream
    BufferedReader br= new BufferedReader(fr);

    // Skipping blank lines
    String buffString= null;
    try {
        do {
            buffString= br.readLine();
        } while (blankString.equalsIgnoreCase(buffString));
    }
    catch(Exception e) {
        throw new ParserException("Error skipping blank line at the
                                   beginning: \n\t" + e.getMessage());
    }

    // Empty file
    if (buffString == null)
        throw new ParserException("Empty File.");

    // Parsing MINDTCT first line and controlling format
    StringTokenizer firstLine= new StringTokenizer(buffString);
    if ((firstLine.countTokens()) != 3)
        throw new ParserException("Not a valid file format (first line).");

    Integer numMin= null;
    try {
        try {
            numMin= new Integer(firstLine.nextToken());
        }
        catch (NumberFormatException nfe) {
            throw new ParserException("Not a valid minutiae number
                                       format (first line).");
        }
    }

```

```

        if ((!
            secondTokenFirstLine.equalsIgnoreCase(firstLine.nextToken()))
            || (!
                thirdTokenFirstLine.equalsIgnoreCase(firstLine.nextToken())))
            throw new ParseException("Not a valid file format (first
                                    line).");
    }
    catch (NoSuchElementException nsee) {
        throw new ParseException("Not Expected error in string
                                tokenizing.");
    }
}
catch (NoSuchElementException nsee) {
    throw new ParseException("Unexpected no more token error: \n\t" +
                            nsee.getMessage());
}

// Minutiae number expected to be found at the end of the parsing
int nm= numMin.intValue();
if (nm <= 0)
    throw new ParseException("Not a valid minutiae number in first line
                            (" + nm + ").");

System.out.println("File with " + numMin + " minutiae.\n");

// Loop: parsing line after line
LineParser lp= new LineParser();
int countLine= 0;
for (int i= 0; i < nm; i++) {
    do {
        try {
            countLine++;
            buffString= br.readLine();
        }
        catch (IOException ioe) {
            throw new ParseException("Error reading " + countLine + "
                                    line: \n\t" + ioe.getMessage());
        }
    } while (blankString.equalsIgnoreCase(buffString));

    if (buffString == null)
        throw new ParseException("Truncated file.");

    // Giving control to LineParser
    try {
        lp.lineParse(buffString, ml);
    }
    catch (ParseException pe) {
        throw new ParseException("Error parsing " + countLine + " line:
                                \n\t" + pe.getMessage());
    }
}
}
}

```

```

/*
 * ParseException.java
 *
 * Created on 10 ottobre 2002, 17.38
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package parsing;

/**
 * Exception thrown by <code>Parser</code> and <code>LineParser</code>
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class ParseException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>ParseException</code> without detail
     * message.
     */
    public ParseException() {
    }

    /**
     * Constructs an instance of <code>ParseException</code> with the specified
     * detail message.
     *
     * @param msg the detail message.
     */
    public ParseException(String msg) {
        super(msg);
    }
}

```

C.9 Classe Main FinMatchTest.

```
/*
 * FinMatch.java
 *
 * Created on 9 ottobre 2002, 21.19
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/***** IMPORTS *****/
import java.io.FileReader;
import java.io.FileOutputStream;
import java.io.PrintStream;
import parsing.Parser;
import parsing.ParserException;
import minutiaeListConverting.MinutiaeListConverter;
import minutiaeListConverting.MinutiaeListConverterException;
import algMinInformation.AlgMinutiaeList;
import matching.Matcher;
import matching.minCostAssigning.MatchSet;
import matching.MatcherException;
import minutiaeInformation.MinutiaeList;
import minutiaeFileWriting.MinutiaeFileWriter;
import minutiaeFileWriting.MinutiaeFileWriterException;
import algParsing.AlgParser;
import minutiaeSuperClasses.MinutiaeListException;

/**
 * This is one of the class containing main method for the application.<p>
 * The aim of the application is to compare two fingerprints and to give a
 * matching rating. The extraction of the singularity points (called also<i>
 * minutiae</i>) is provided by <i>NIST</i> (National Standard for Information
 * Systems) program <MINDTCT> which is released under a large public domain
 * license.<p>
 * The two parameters are the MINDTCT file where the application can find
 * information about the two minutiae sets which is going to compare.
 * The application parses these files, turns information retrieved in a form
 * that can be used by the matching algorithm, writes two new files containing
 * this new information, parses these two new files and, at last, tries to match
 * the fingerprints. <p>
```

```

* The code was writed thinking about testing this matching algorithm before
* porting it on JavaCard.<p>
* This main code was writed to test all of the application phases. So if you
* want only to use the application, it'better you run first
* <i>FinMatchWriter</i> to convert minutiae information in a new file, and then
* run <i>FinMatch</i> on two of these new file to have a minutiae comparison.
*
* @author Francesco Santini
* @version 1.0
*/
public class FinMatchTest {

    /** Creates a new instance of FinMatch */
    public FinMatchTest() {
    }

    /**
     * @param args the command line arguments: two .txt files containig MINDTCT
     *          minutiae
     */
    public static void main(String[] args) {

        // Application files subfixes
        String oldSubfix= ".txt";
        String newSubfix= "_out.txt";

        System.out.println("Fingerprint matching application started.\n");

        if ((args.length > 3) || (args.length < 2)) {
            System.err.println("Error on input parameter number.");
            System.err.println("\tUsage: java FinMatch \"filename2\"
                               \"filename2\"");
            System.exit(-1);
        }

        System.out.println("File opening phase.\n");

        // Parsing input files
        FileReader fileModelInput= null;
        FileReader fileTestInput= null;

        try {
            fileModelInput= new FileReader(args[0]);
        }
        catch (Exception e) {
            System.err.println("Error opening first file (" + args[0] + "):");
            System.err.println("\t" + e.getMessage());
            System.exit(-1);
        }
        try {
            fileTestInput= new FileReader(args[1]);
        }
        catch (Exception e) {
            System.err.println("Error opening second file (" + args[1] + "):");
            System.err.println("\t" + e.getMessage());
            System.exit(-1);
        }

        // Creating Parser
        Parser minParser= new Parser();

        System.out.println(args[0] + " parsing phase.\n");
    }
}

```

```

// Creating minutiae list for the first file
MinutiaeList minutiaeModelList= null;
try {
    minutiaeModelList= new MinutiaeList();
}
catch (MinutiaeListException mle) {
    System.err.println("Error creating first minutiae list: " +
        mle.getMessage());
    System.exit(-1);
}

// Parsing first file
try {
    minParser.parse(fileModelInput, minutiaeModelList);
}
catch (ParserException pe) {
    System.err.println("Error parsing first file (" + args[0] + "):");
    System.err.println("\t" + pe.getMessage());
    try {
        fileModelInput.close();
    }
    catch (Exception e) {
        System.err.println("Error closing " + args[0] + " file.");
    }
    System.exit(-1);
}

// Closing first file
try {
    fileModelInput.close();
}
catch (Exception e) {
    System.err.println ("Error closing " + args[0] + " file.");
    System.exit(-1);
}

System.out.println(args[1] + " parsing phase.\n");

// Creating minutiae list for the second file
MinutiaeList minutiaeTestList= null;
try {
    minutiaeTestList= new MinutiaeList();
}
catch (MinutiaeListException mle) {
    System.err.println("Error creating second minutiae list: " +
        mle.getMessage());
    System.exit(-1);
}

// Parsing second file
try {
    minParser.parse(fileTestInput, minutiaeTestList);
}
catch (ParserException pe) {
    System.err.println("Error parsing second file (" + args[1] + "):");
    System.err.println("\t" + pe.getMessage());
    try {
        fileTestInput.close();
    }
    catch (Exception e) {
        System.err.println("Error closing " + args[1] + " file.");
    }
    System.exit(-1);
}

```

```

}

// Closing second file
try {
    fileTestInput.close();
}
catch (Exception e) {
    System.err.println ("Error closing " + args[1] + " file.");
    System.exit(-1);
}

// Printing minutiae extracted from the first file
System.out.println("***** Printing " + args[0] + " minutiae list
*****");
try {
    minutiaeModelList.print();
}
catch (MinutiaeListException mle) {
    System.err.println("Error printing " + args[0] + " minutiae list:
    \n\t" + mle.getMessage());
    System.exit(-1);
}

// Printing minutiae extracted from the second file
System.out.println("***** Printing " + args[1] + " minutiae list
*****");
try {
    minutiaeTestList.print();
}
catch (MinutiaeListException mle) {
    System.err.println("Error printing " + args[1] + " minutiae list:
    \n\t" + mle.getMessage());
    System.exit(-1);
}

System.out.println("Converting information for matching algorithm");

AlgMinutiaeList matchListModel= null;
MinutiaeListConverter mlc= new MinutiaeListConverter();
try {
    matchListModel= mlc.convert(minutiaeModelList);
}
catch (MinutiaeListConverterException mlce) {
    System.err.println ("Error converting " + args[0] + "
    information.");
    System.err.println ("\t" + mlce.getMessage());
    System.exit(-1);
}

AlgMinutiaeList matchListTest= null;
try {
    matchListTest= mlc.convert(minutiaeTestList);
}
catch (MinutiaeListConverterException mlce) {
    System.err.println ("Error converting " + args[1] + "
    information.");
    System.err.println ("\t" + mlce.getMessage());
    System.exit(-1);
}

// Printing minutiae converted from the first file
System.out.println("***** Printing file " + args[0] + " new information

```

```

        *****");
    try {
        matchListModel.print();
    }
    catch(MinutiaeListException mle) {
        System.err.println("Error printing " + args[0] + " minutiae list:
            \n\t" + mle.getMessage());
        System.exit(-1);
    }

    // Printing minutiae converted from the second file
    System.out.println("***** Printing file " + args[1] + " new information
        *****");
    try {
        matchListTest.print();
    }
    catch(MinutiaeListException mle) {
        System.err.println("Error printing " + args[1] + " minutiae list:
            \n\t" + mle.getMessage());
        System.exit(-1);
    }

    FileOutputStream fileModelOutput= null;
    FileOutputStream fileTestOutPut= null;

    // Creating file name for minutiae converted from the first file
    String outModelFileName= null;
    int tempIndex= 0;
    try {
        tempIndex= args[0].indexOf(oldSubfix);
    }
    catch(Exception e) {
        System.err.println ("Error creating new file name (" + args[0] +
            ").");
        System.err.println ("\t" + e.getMessage());
        System.exit(-1);
    }

    if (tempIndex == -1) {
        System.err.println ("Error creating new file name (" + args[0] +
            ").");
        System.err.println ("\tNot a .txt input file");
        System.exit(-1);
    }

    try {
        String prefixString= args[0].substring(0, tempIndex);
        outModelFileName= prefixString.concat(newSubfix);

        // Code above is for 1.2 API compatibility. Below is only for 1.4
        //outFile= args[parameter].replaceAll(".txt", "_out.txt");
    }
    catch (Exception e) {
        System.err.println ("Error creating new file name (" + args[0] +
            ").");
        System.err.println ("\t" + e.getMessage());
        System.exit(-1);
    }

    // Creating file name for minutiae converted from the second file
    String outTestFileName= null;
    try {
        tempIndex= args[1].indexOf(oldSubfix);

```



```

}
catch(Exception e) {
    System.err.println ("Error creating new file name (" + args[1] +
        ").");
    System.err.println ("\t" + e.getMessage());
    System.exit(-1);
}

if (tempIndex == -1) {
    System.err.println ("Error creating new file name (" + args[1] +
        ").");
    System.err.println ("\tNot a .txt input file");
    System.exit(-1);
}

try {
    String prefixString= args[1].substring(0, tempIndex);
    outTestFileName= prefixString.concat(newSubfix);

    // Code above is for 1.2 API compatibility. Below is only for 1.4
    //outFile= args[parameter].replaceAll(".txt", "_out.txt");
}
catch (Exception e) {
    System.err.println ("Error creating new file name (" + args[1] +
        ").");
    System.err.println ("\t" + e.getMessage());
    System.exit(-1);
}

// Creating FileReader to write in first new file
PrintStream pSModel= null;
try {
    FileOutputStream outM= new FileOutputStream(outModelFileName);
    pSModel= new PrintStream(outM);
}
catch (Exception e) {
    System.err.println("Error creating " + outModelFileName + "
        stream.");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}

// Creating FileReader to write in second new file
PrintStream pSTest= null;
try {
    FileOutputStream outT= new FileOutputStream(outTestFileName);
    pSTest= new PrintStream(outT);
}
catch (Exception e) {
    System.err.println("Error creating " + outTestFileName + "
        stream.");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}

// Creating the file writer object
MinutiaeFileWriter mfw= new MinutiaeFileWriter();

System.out.println("Writing information in " + outModelFileName + "
    file.\n");

```

```

// Writing in first new file
try {
    mfw.write(pSModel, matchListModel);
}
catch (MinutiaeFileWriterException mfwe) {
    System.err.println("Error writing information in " +
        outModelFileName + " file.");
    System.err.println("\t" + mfwe.getMessage());
    System.exit(-1);
}

// Closing first new file
try {
    pSModel.flush();
    pSModel.close();
}
catch (Exception e) {
    System.err.println("Error closing " + outModelFileName + " file.");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}

System.out.println(outModelFileName + " wrote successfully.\n");
System.out.println("Writing information in " + outTestFileName + "
    file.\n");

// Writing in second new file
try {
    mfw.write(pSTest, matchListTest);
}
catch (MinutiaeFileWriterException mfwe) {
    System.err.println("Error writing information in " + outTestFileName
        + " file.");
    System.err.println("\t" + mfwe.getMessage());
    System.exit(-1);
}

// Closing second new file
try {
    pSTest.flush();
    pSTest.close();
}
catch (Exception e) {
    System.err.println("Error closing " + outTestFileName + " file.");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}

System.out.println(outTestFileName + " wrote successfully.\n");

// Parsing new files (to see if the parser works well)
System.out.println("Parsing two new files phase.\n");

// Creating the parser for new file format
AlgParser ap= new AlgParser();
FileReader file1= null;
try {
    file1= new FileReader(outModelFileName);
    AlgMinutiaeList l1= new AlgMinutiaeList();
    ap.parse(file1, l1);
    file1.close();
    l1.print();
}

```

```

    }
    catch (Exception e) {
        System.err.println("Errore su chiusura stream per la scrittura sul
                           file di test.");
        System.err.println("\t" + e.getMessage());
        System.exit(-1);
    }

    FileReader file2= null;
    try {
        file2= new FileReader(outTestFileName);
        AlgMinutiaeList l2= new AlgMinutiaeList();
        ap.parse(file2, l2);
        file2.close();
        l2.print();
    }
    catch (Exception e) {
        System.err.println("Errore su chiusura stream per la scrittura sul
                           file di test.");
        System.err.println("\t" + e.getMessage());
        System.exit(-1);
    }

    // *****
    //                               MATCHING
    // *****

    System.out.println("Fingerprint matching phase.\n");

    // Creating the Matcher
    Matcher minMatcher= new Matcher();

    // Matching
    MatchSet result= null;
    try {
        result= minMatcher.match(matchListModel, matchListTest);
    }
    catch (MatcherException me) {
        System.err.println("Error matching fingerprint.");
        System.err.println("\t" + me.getMessage());
        System.exit(-1);
    }

    // Printing result.
    result.printMinutiae();

}

}

```

C.10 Classe Main FinalTest.

```
/*
 * FinalTest.java
 *
 * Created on 16 luglio 2003, 17.01
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 221, Boston, MA 02222-2217 USA
 */

/***** IMPORTS *****/
import java.io.*;
import algParsing.AlgParser;
import algParsing.AlgParserException;
import algMinInformation.AlgMinutiaeList;
import matching.Matcher;
import matching.minCostAssigning.MatchSet;
import matching.MatcherException;
import minutiaeSuperClasses.MinutiaeListException;

/**
 * This class has a <i>main</i> method that calculates <i>FAR</i> (False
 * Acceptance Rate) and <i>FRR</i> (False Rejection Rate) for the given
 * fingerprint database. Result is in the output text file. Test description can
 * be found at: <link>http://bias.csr.unibo.it/fvc2004/perfeval.asp</link>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class FinalTest {

    /**
     * Match threshold: below this value, two fingerprints are positively
     * matched.
     */
    private static final float maxMatchCost= 280;

    /**
     * Creates a new instance of FinalTest.
     */
    public FinalTest() {
```

```

}

/**
 * Test <i>main</i>.
 *
 * @param args the command line arguments (two parameters): directory path
 * of fingerprint text file database and path of the file with containing
 * match result.
 */
public static void main(String[] args) {

    if (args.length != 2) {

        System.err.println("Error on input parameters.");
        System.err.println("\tUsage: java FinalTest input_directory
                                outputfile");
        System.exit(-1);

    }

    String dirBase= new String(args[0]);
    String outfi= new String(args[1]);

    // Creating log file
    FileOutputStream out= null;
    PrintStream p= null;

    try {
        out = new FileOutputStream(outfi);
        p = new PrintStream( out );
    }
    catch (Exception e) {
        System.err.println("Error writing to file.\n\t" + e.getMessage());
        System.exit(-1);
    }

    int cont_test = 0;
    int frr= 0;
    int far= 0;
    //int count_match= 0;
    //int count_match_frr=0;
    //int count_match_far=0;
    boolean frrFlag= false;

    for (int i= 1; i <= 110; i++) {

        p.println("*****Matching phase for fingerprint " + i + "
                    *****" +"\n");

        for (int j= 1; j <= 8; j++) {

            String file1= new String(dirBase + Integer.toString(i) + "_" +
                                    Integer.toString(j) + "_min_out.txt");

            for (int k= 1; k <= 110; k++) {
                if (i > k) continue; // no symmetric

                if ((j != 1) && (i != k)) continue;
                //if ((i == k) && (j == 1)) continue;

                for (int z= 1; z <= 8; z++) {
                    if (j > z) continue; // no symmetric

```

```

if ((i != k ) && (z != 1)) continue;
if ((i == k) && (j == z)) continue;

System.out.println("test n. " + (++cont_test) + "\n");

String file2= new String(dirBase + Integer.toString(k) +
                        "_" + Integer.toString(z) +
                        "_min_out.txt");

FileReader firstFile= null;
FileReader secondFile= null;

try {
    firstFile= new FileReader(file1);
}
catch (Exception e) {
    System.err.println("Error opening first file (" +
                        file1 + "):");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}
try {
    secondFile= new FileReader(file2);
}
catch (Exception e) {
    System.err.println("Error opening second file (" +
                        file2 + "):");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}

// Creating the parser for file format
AlgParser ap= new AlgParser();

AlgMinutiaeList l1= null;
try {
    l1= new AlgMinutiaeList();
}
catch (MinutiaeListException mle) {
    System.err.println("Error creating first minutiae
                        list: \n\t" + mle.getMessage());
    System.exit(-1);
}

AlgMinutiaeList l2= null;
try {
    l2= new AlgMinutiaeList();
}
catch (MinutiaeListException mle) {
    System.err.println("Error creating second minutiae
                        list: \n\t" + mle.getMessage());
    System.exit(-1);
}

//System.out.println(file1 + " parsing phase.\n");

// Parsing first file
try {
    ap.parse(firstFile, l1);
}
catch (AlgParserException pe) {
    System.err.println("Error parsing first file (" +

```

```

        file1 + "):");
    System.err.println("\t" + pe.getMessage());
    try {
        firstFile.close();
    }
    catch (Exception e) {
        System.err.println("Error closing " + file1 + "
            file.");
        System.err.println("\t" + e.getMessage());
    }
    System.exit(-1);
}

// Closing first file
try {
    firstFile.close();
}
catch (Exception e) {
    System.err.println("Error closing " + file1 + "
        file.");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}

//System.out.println(file2 + " parsing phase.\n");

// Parsing second file
try {
    ap.parse(secondFile, l2);
}
catch (AlgParserException pe) {
    System.err.println("Error parsing second file (" +
        file2 + "):");
    System.err.println("\t" + pe.getMessage());
    try {
        secondFile.close();
    }
    catch (Exception e) {
        System.err.println("Error closing " + file2 + "
            file.");
        System.err.println("\t" + e.getMessage());
    }
    System.exit(-1);
}

// Closing second file
try {
    secondFile.close();
}
catch (Exception e) {
    System.err.println("Error closing " + file2 + "
        file.");
    System.err.println("\t" + e.getMessage());
    System.exit(-1);
}

// Creating the Matcher
Matcher minMatcher= new Matcher();

// Matching
MatchSet result= null;
try {

```

```

        result= minMatcher.match(l1, l2);
    }
    catch (MatcherException me) {
        System.err.println("Error matching fingerprint.");
        System.err.println("\t" + me.getMessage());
        System.exit(-1);
    }

    //p.println("File " + file1 + " " + file2 + "    match
    //result= " + Float.toString(result) + "\n");
    float resultCost= (float) result.returnMatchCost();
    int pairs= result.getPairsNumber();
    float mean= 0;
    if (result.getPairsNumber() != 0)
        mean= resultCost / result.getPairsNumber();
    if (i == k) {
        p.print("FRR ");
        frrFlag= true;
        //count_match_frr++;
    }
    else {
        frrFlag= false;
        //count_match_far++;
    }

    p.print("'" + i+"_"+j + " Vs " + k+"_"+z + "    match
        result= " + Float.toString(mean) + " - Pairs
        number= " + pairs);
    //count_match++;

    // Checking match
    if ((mean < maxMatchCost) /*&& (pairs >= 15)*/) {

        // Fingerprints are matched!
        p.println(" *****OK***** \n");

        // Checking FAR
        if (!frrFlag)
            far++;
    }
    else {
        p.println("\n");

        // Checking FRR
        if (frrFlag)
            frr++;
    }
}

}

}

float farPCent= ((float)far * (float)100) / (float)5995;
float frrPCent= ((float)frr * (float)100) / (float)3080;

p.println("\n\n FAR = " + Float.toString(farPCent) + "    FRR = " +
    Float.toString(frrPCent) + " NFAR = " + far + " NFRR = " + frr
    + " maxMatchCost= " + maxMatchCost);

try {
    p.close();

```



```
    }  
    catch (Exception e) {  
        System.err.println("Error closing output file.\n\t" +  
                            e.getMessage());  
    }  
}  
}
```

C.11 Classe Main Farfrr.

```
/*
 * farfrr.java
 *
 * Created on 16 luglio 2003, 17.01
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 221, Boston, MA 02222-2217 USA
 */

import java.io.*;
import java.lang.*;
import java.util.StringTokenizer;

/**
 * This class helps checking algorithm performance with threshold value
 * changing.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class Farfrr {

    /**
     * Minimum threshold to check performance.
     */
    private final static int mint = 3500;

    /**
     * Maximum threshold to check performance.
     */
    private final static int maxt = 6000;

    /**
     * Creates a new instance of farfrr.
     */
    public Farfrr() {
    }

    /**
```

```

* Tool to check performance changing threshold value from minimum to
* maximum. Output text file stores these performances.
*
* @param args the command line arguments: Input text file containing
*         <code>FinalTest</code> output and second parameter is new
*         text file path.
*/
public static void main(String[] args) {

    // Controlling input parameters
    if (args.length != 2) {

        System.err.println("Error on input parameters.");
        System.err.println("\tUsage: java farfrr input_match_file
                           outputfile");
        System.exit(-1);

    }

    // Changing threshold (to minimum to maximum parameters)
    float maxMatchCost= 0;

    String infi= new String(args[0]);
    String outfi= new String(args[1]);

    // Creating log file
    FileOutputStream out= null;
    PrintStream p= null;

    try {
        out = new FileOutputStream(outfi);
        p = new PrintStream( out );
    }
    catch (Exception e) {
        System.err.println("Error writing to file.\n\t" + e.getMessage());
        System.exit(-1);
    }

    int cont_test = 0;
    int frr= 0;
    int far= 0;
    int count_match= 0;
    boolean frrFlag= false;

    for (int w = mint; w <= maxt; w++) {
        maxMatchCost = w;
        cont_test = 0;
        frr= 0;
        far= 0;
        frrFlag= false;

        BufferedReader in=null;
        try {
            in = new BufferedReader(new FileReader(infi));
        }
        catch (Exception e) {
            System.err.println("Error opening file.\n\t" + e.getMessage());
            System.exit(-1);
        }

        for (int i= 1; i <= 110; i++) {
            //p.println("*****inizio match phase per impronta " +i+ "

```

```

//*****" + "\n");

for (int j= 1; j <= 8; j++) {

    //String file1= new String(dirBase + Integer.toString(i) +
    // "_" + Integer.toString(j) + "_min_out.txt");

    for (int k= 1; k <= 110; k++) {
        if (i > k) continue;

        if ((j != 1) && (i != k)) continue;
        //if ((i == k) && (j == 1)) continue;

        for (int z= 1; z <= 8; z++) {
            if (j > z) continue;

            if ((i != k ) && (z != 1)) continue;
            if ((i == k) && (j == z)) continue;

            //System.out.println("test n. " + (++cont_test)
            // + "\n");

            int pairs= 0;
            float mean= 0;
            boolean trovato = false;

            while (!trovato) {

                String aaa=null;
                try {
                    aaa = in.readLine();
                }
                catch (Exception me) {
                    System.err.println("Error readline");
                }

                StringTokenizer st=null;
                try {
                    st = new StringTokenizer(aaa);
                }
                catch (Exception e) {
                    System.err.println("Error StringTokenizer" +
                                        e.getMessage());
                }
                while (st.hasMoreTokens()) {
                    //System.out.println(st.nextToken());
                    String bada = st.nextToken();

                    if ( bada.equals("result=") ) {
                        bada = st.nextToken();
                        mean = (new Float(bada)).floatValue();
                        trovato = true;
                    }
                    else if ( bada.equals("number=") ) {
                        bada = st.nextToken();
                        pairs = (new Integer(bada)).intValue();
                    }
                }
            }
        }
    }
}

```

```

        if (i == k) {
            //p.print("FRR ");
            frrFlag= true;
            //count_match_frr++;
        }
        else {
            frrFlag= false;
            //count_match_far++;
        }

        //p.print(" " + i+"_"+j + " Vs " + k+"_" + z + "
        //match result= " + Float.toString(mean) + " - Pairs
        //number= " + pairs);
        //count_match++;
        //p.print(count_match);
        if ((mean < maxMatchCost) /*&& (pairs >= 5)*/) {
            //p.println(" *****OK*****");
            //\n");
            if (!frrFlag)
                far++;

        }

        else {
            //p.println("\n");
            if (frrFlag)
                frr++;

        }

    }

}

}

}

float farPCent= ((float)far * (float)100) / (float)5995;
float frrPCent= ((float)frr * (float)100) / (float)3080;

p.println("FAR = " + Float.toString(farPCent) + "    FRR = " +
    Float.toString(frrPCent) + " NFAR = " + far + " NFRR = "
    + frr + " maxMatchCost= " + maxMatchCost);

} // end for far frr

try {
    p.close();
}
catch (Exception e) {
    System.err.println("Error closing output file.\n\t" +
        e.getMessage());
}

} // end main

}

```


Appendice D: listato del codice sorgente dell'applicazione Java Card™.

D.1 Package converting.

```
/*
 * CardMinutia.java
 *
 * Created on 25 febbraio 2003, 11.57
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package converting;

/***** IMPORTS *****/
import javaMatchOnCard.*;

/**
 * This class represents minutia information to be sent to SmartCard.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class CardMinutia {
```

```

/**
 * Minutia x position.
 */
private short xPos= (short) 0;

/**
 * Minutia y position.
 */
private short yPos= (short) 0;

/**
 * Minutia reliability.
 */
private byte reliability= (byte) 0;

/**
 * Minutia type: value <i>0</i> stands for "ridge ending" and <i>1</i> for
 * bifurcation.
 */
private byte type= (byte) 0;

/**
 * Vector storing neighbors digest.
 */
private short[] neighborsVector= null;

/**
 * Creates a new instance of CardMinutia.
 */
public CardMinutia() {
    this.neighborsVector= new short[JMOCCardMinutia.neighborsNumber];
}

/**
 * Returns x position of this minutia.
 *
 * @return Minutia x position.
 */
public short getXPos() {
    return (this.xPos);
}

/**
 * Returns y position of this minutia.
 *
 * @return Minutia y position.
 */
public short getYPos() {
    return (this.yPos);
}

```



```

}

/**
 * Returns reliability of this minutia.
 *
 * @return Minutia reliability.
 */
public byte getReliability() {
    return (this.reliability);
}

/**
 * Returns type (ridge ending or bifurcation) of this minutia.
 *
 * @return Minutia minutia type.
 */
public byte getType() {
    return (this.type);
}

/**
 * Returns neighbor vector (feature digest values) of this minutia.
 *
 * @return Minutia neighbors vector.
 */
public short[] getNeighborsVector() {
    return (this.neighborsVector);
}

/**
 * Sets x position for this minutia.
 *
 * @param x New x position.
 */
public void setXPos(short xPos) {
    this.xPos= xPos;
}

/**
 * Sets y position for this minutia.
 *
 * @param y New y position.
 */
public void setYPos(short yPos) {
    this.yPos= yPos;
}

```

```

/**
 * Sets reliability for this minutia.
 *
 * @param reliability New minutia reliability.
 */
public void setReliability(byte reliability) {
    this.reliability= reliability;
}

/**
 * Sets type (ridge ending or bifurcation) for this minutia.
 *
 * @param type New minutia type.
 */
public void setType(byte type) {
    this.type= type;
}
}

```

```

/*
 * Convert.java
 *
 * Created on 24 febbraio 2003, 17.39
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 *
 */

```

```
package converting;
```

```
/****** IMPORTS *****/
```

```
import algMinInformation.*;
import minutiaeListConverting.*;
import minutiaeSuperClasses.*;
import javaMatchOnCard.*;
import java.util.*;
```

```
/**
```

```
 * Methods in this class convert minutiae from text file format to algorithm
 * input format.
 *
```

```
 * @author Francesco Santini
 * @version 1.0
 */
```

```
public class Convert {
```

```
    /**
```

```
     * Number of minutiae in returned list.
    */
```

```
    private static final int minNumber= MinutiaeListConverter.cutNumber;
```

```
    /**
```

```
     * Maximum short value, used neighbors digest.
    */
```

```
    private static final short maxShort= (short) 32767;
```

```
    /**
```

```
     * Euclidean distance weight.
    */
```

```
    private static final short edWeight= 400;
```

```

/**
 * Distance relative angle weight.
 */
private static final short draWeight= 350;

/**
 * Relative angle weight.
 */
private static final short raWeight= 300;

/**
 * ridge count weight.
 */
private static final short rcWeight= 300;

/**
 * Creates a new instance of Convert.
 */
public Convert() {
}

/**
 * Converts list of minutia <code>listToConvert</code> in a
 * <code>Vector</code> of <code>CardMinutia</code> compatible with matching
 * card algorithm.
 *
 * @param listToConvert Minutia list to convert.
 * @return Vector storing converted minutiae.
 *
 * @throws ConvertException If failing while converting: error extracting
 *                          minutia from list.
 */
public static Vector convert(AlgMinutiaeList listToConvert) throws
    ConvertException {

    if (listToConvert == null)
        throw new ConvertException("Null input parameter.");

    Vector returnList= new Vector(Convert.minNumber);

    int highBound= Convert.minNumber;
    if (listToConvert.size() < highBound)
        highBound= listToConvert.size();

    for (int i= 0; i < highBound; i++) {

        // Creating new Card minutia
        CardMinutia minToInsert= new CardMinutia();

        AlgMinutia tempMin= null;
        try {
            tempMin= listToConvert.get(i);
        }
        catch (MinutiaeListException mle) {
            throw new ConvertException("Error getting minutia " + i + " from
                input list: \n\t" + mle.getMessage());
        }

        // Converting minutia

```

```

        try {
            Convert.convMinutia(tempMin, minToInsert, listToConvert);
        }
        catch(ConvertException ce) {
            throw ce;
        }

        // Inserting new Card minutia in returned list.
        returnList.add(i, minToInsert);

    }

    // Returning minutiae list in new format
    return returnList;
}

/**
 * Convert one minutia in a format consistent with card matching algorithm.
 *
 * @param fromMin Original minutia.
 * @param toMin Converted minutia: this method fills its fields converting
 *         from <code>fromMin</code>
 *
 * @throws ConvertException If fails to convert minutia information
 */
private static void convMinutia(AlgMinutia fromMin, CardMinutia toMin,
                                AlgMinutiaeList listToConvert) throws
                                ConvertException {

    if ((fromMin == null) || (toMin == null))
        throw new ConvertException("Null input parameter");

    // Setting x position
    short minId= fromMin.getId();
    short xPos= fromMin.getXPos();
    toMin.setXPos(xPos);

    // Setting y position
    short yPos= fromMin.getYPos();
    toMin.setYPos(yPos);

    // Setting reliability
    float reliability= fromMin.getReliability();
    byte relCard= (byte)(reliability * 100);
    toMin.setReliability(relCard);

    // Setting type
    AlgMinutiaType tempType= null;
    tempType= fromMin.getType();
    if (tempType.isBifurcation())
        toMin.setType((byte) 1);
    else
        if (tempType.isRidgeEnding())
            toMin.setType((byte) 0);
        else
            throw new ConvertException("Unknown minutia type in minutia " +
                                       minId);

    AlgNeighborList neighborList= fromMin.getAlgNeighborList();

    // Controlling neighbors number

```

```

if (neighborList.size() != JMOCMinutia.neighborsNumber)
    throw new ConvertException("Minutia " + minId + " neighbors number
                                not allowed.");

short[] cardNeighborsVector= toMin.getNeighborsVector();

// Filling neighbors vector
for (int i= 0; i < JMOCMinutia.neighborsNumber; i++) {

    AlgNeighbor tempNeighbor= null;
    try {
        tempNeighbor= neighborList.get(i);
    }
    catch(NeighborListException nle) {
        throw new ConvertException("Error retrieving neighbor " + i);
    }

    float listToConvertMaxNeighDistance=
        listToConvert.getMaxNeighDistance();
    short listToConvertMaxRidgeCount= listToConvert.getMaxRidgeCount();

    // Retrieving information fields
    float distance=
        tempNeighbor.getDistance()/listToConvertMaxNeighDistance;
    //System.out.println("Distance : " + distance);
    if (distance > 1)
        distance= 1;
    float dRelativeAngle= tempNeighbor.getDistanceRelativeAngle();
    if (dRelativeAngle < 0)
        dRelativeAngle+= 360;
    dRelativeAngle= dRelativeAngle / 360;
    //System.out.println("dRelativeAngle : " + dRelativeAngle);
    float rAngle= tempNeighbor.getRelativeAngle();
    if (rAngle < 0)
        rAngle+= 360;
    rAngle= rAngle / 360;
    //System.out.println("rAngle : " + rAngle);
    short rc= (short)
        (tempNeighbor.getRidgeCount()/listToConvertMaxRidgeCount);
    //System.out.println("Ridge count : " + rc + "\n");

    if ((distance > 1) || (dRelativeAngle > 1) || (rAngle > 1) || (rc >
    1) ||(distance < 0) || (dRelativeAngle < 0) || (rAngle < 0) || (rc <
    0))
        throw new ConvertException ("Error converting minutia (" +
            distance + " " + dRelativeAngle + " "
            + rAngle + " " + rc);

    float sum= ((distance * edWeight) + (dRelativeAngle * draWeight) +
        (rAngle * raWeight) + (rc * rcWeight)) ;

    if ((sum > Convert.maxShort) || (sum < 0))
        throw new ConvertException("Unexpected error: neighbor digest
            value not valid for short type (" +
            sum + ")");

    short neighborDigest= (short) (sum);

    cardNeighborsVector[i]= neighborDigest;
}
}
}

```

```

/*
 * ConvertException.java
 *
 * Created on 24 febbraio 2003, 17.45
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package converting;

/**
 * Exception thrown by methods in <code>converting.Convert</code> class.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class ConvertException extends java.lang.Exception {

    /**
     * Creates a new instance of <code>ConvertException</code> without detail
     * message.
     */
    public ConvertException() {
    }

    /**
     * Constructs an instance of <code>ConvertException</code> with the
     * specified detail message.
     *
     * @param msg the detail message.
     */
    public ConvertException(String msg) {
        super(msg);
    }
}

```

D.2 Package javaMatchOnCard.

```
/*
 * BioBuilder.java
 *
 * Created on 29 gennaio 2003, 21.15
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package javaMatchOnCard;

/***** IMPORTS *****/
import javacard.framework.*;

/**
 * This class implements method JC Biometric API <code>buildBioTemplate</code>.
 *
 *
 * @author Francesco Santini
 * @version 1.0
 */
public final class BioBuilder {

    // ----- Constants -----
    public static final byte FACIAL_FEATURE = (byte)1;
    public static final byte VOICE_PRINT = (byte)2;
    public static final byte FINGERPRINT = (byte)3;
    public static final byte IRIS_SCAN = (byte)4;
    public static final byte RETINA_SCAN = (byte)5;
    public static final byte HAND_GEOMETRY = (byte)6;
    public static final byte SIGNATURE = (byte)7;
    public static final byte KEYSTROKES = (byte)8;
    public static final byte LIP_MOVEMENT = (byte)9;
    public static final byte THERMAL_FACE = (byte)10;
    public static final byte THERMAL_HAND = (byte)11;
    public static final byte GAIT_STYLE = (byte)12;
    public static final byte BODY_ODOR = (byte)13;
    public static final byte DNA_SCAN = (byte)14;
    public static final byte EAR_GEOMETRY = (byte)15;
    public static final byte FINGER_GEOMETRY = (byte)16;
    public static final byte PALM_GEOMETRY = (byte)17;
```



```

public static final byte VEIN_PATTERN    = (byte)18;
public static final byte PASSWORD        = (byte)19;

// Max try limit permitted
private static final byte maxTryLimit= 5;

// ----- Constructors -----

/**
 * Creates a new instance of BioBuilder.
 */
BioBuilder() {
}

// ----- API methods -----

/**
 * Creates new fingerprint template.
 *
 * @param bioType One template type (<i>"facial"</i>, <i>"fingerprint"</i>,
 *      etc).
 * @param tryLimit Number of times a match can fail before the card is
 *      blocked.
 * @return New fingerprint template.
 *
 * @throws BioException Thrown if there are errors in input parameters
 *      (illegal value for try limit).
 */
public static JMOCOwnerBioTemplate buildBioTemplate(byte bioType, byte
                                                    tryLimit) throws
                                                    BioException {

    // Controlling input parameters
    if (bioType != BioBuilder.FINGERPRINT)
        BioException.throwIt(BioException.NO_SUCH_BIO_TEMPLATE);

    if ((tryLimit < 1) || (tryLimit > maxTryLimit))
        BioException.throwIt(BioException.ILLEGAL_VALUE);

    // Return created template reference
    return (new JMOCOwnerBioTemplate(tryLimit));
}
}

```

```

/*
 * BioServer.java
 *
 * Created on 29 gennaio 2003, 21.15
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

/***** IMPORTS *****/
package javaMatchOnCard;

```

```

/**
 * This class implements method JC Biometric API class
 * <code>BioException</code>.
 *
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class BioException extends javacard.framework.CardRuntimeException {

    /*----- Constants -----*/
    public static final short ILLEGAL_VALUE= (short)1;
    public static final short INVALID_DATA= (short)2;
    public static final short NO_SUCH_BIO_TEMPLATE= (short)3;
    public static final short NO_TEMPLATES_ENROLLED= (short)4;
    public static final short ILLEGAL_USE= (short)5;

    // ----- Data -----
    private static BioException thisBioException= null;

    // ----- Constructors -----

    /**
     * Creates a new instance of BioException.
     *
     * @param reason Codification of exception reason.
     */
    private BioException(short reason) {

        super(reason);
    }

```

```

}

// ----- API methods -----
/**
 * Exception throwned for all for <code>javaMatchOnCard</code> class methods.
 *
 * @param reason Exception reason
 *
 * @throws BioException Throws static exception changing its reason.
 */
public static void throwIt(short reason) throws BioException {
    if (thisBioException == null)
        thisBioException= new BioException(reason);
    else
        thisBioException.setReason((short)reason);

    throw thisBioException;
}

}

```

```

/*
 * BioServer.java
 *
 * Created on 29 gennaio 2003, 21.15
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package javaMatchOnCard;
```

```

/***** IMPORTS *****/
import javacard.framework.*;
import org.javacardforum.javacard.biometry.*;

```

```

/**
 * The <b>BioServer</b> is a manager/owner of fingerprint template on a Java
 * Card.
 */

```

```

 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class BioServer extends javacard.framework.Applet {
```

```
    // ---- CONSTANTS ---- //
```

```
    final static byte TRY_LIMIT= (byte)0x05;
```

```
    // Class code in command APDU
```

```
    final static byte BIO_SERVER_CLA= (byte) 0x03;
```

```
    // Instruction codes for command APDU (APDU header)
```

```

/**
 * Initializing match phase.
 */

```

```
    final static byte BIO_VERIFY_INIT_MATCH= (byte) 0x01;
```

```

/**
 * Going on with matching phase.
 */

```

```
    final static byte BIO_VERIFY_MATCH= (byte) 0x02;
```

```

/**
 * Initializing enrollment phase.

```

```

    */
final static byte BIO_INS_TEMPLATE_INIT= (byte) 0x03;

/**
 * Going on with enrollment phase.
 */
final static byte BIO_INS_TEMPLATE_UPDATE= (byte) 0x04;

/**
 * Ending matching phase.
 */
final static byte BIO_INS_TEMPLATE_DOFINAL= (byte) 0x05;

/**
 * Verify previous match result.
 */
final static byte BIO_CHECK_VERIFY_RESULT= (byte) 0x06;

/**
 * Verify if a template is enrolled on card.
 */
final static byte BIO_CHECK_ENROLLED= (byte) 0x07;

/**
 * Get public template data.
 */
final static byte BIO_GET_PUBLIC_TEMPLATE_DATA= (byte) 0x08;

/**
 * Unblocking blocked template.
 */
final static byte BIO_UNBLOCK_REFERENCE= (byte) 0x09;

// Applet specific status word

/**
 * Exception base value for <code>BioException</code> value.
 */
private static final short SW_BIOEXCEPTION_BASE= (short) 0x9000;

/**
 * Exception code for failed match.
 */
private static final short SW_NOT_VALIDATED= (short) 0x9010;

/**
 * Exception code if no reference template is enrolled on card.
 */
private static final short SW_NOT_INITIALIZED= (short) 0x9011;

// ---- DATA MEMBERS ---- //

/**
 * Full access for server.
 */
public JMOCOwnerBioTemplate fullBio= null;

/**
 * Limited access for client.
 */
public ProxyBioTemplate proxyBioForClient= null;

```

```

/**
 * Method <code>init</code> can be called only once. This flag is
 * <i>true</i> when has been called, <i>false</i> otherwise.
 */
private static boolean initCalled= false;

/**
 * Creates a new instance of BioServer.
 *
 * @param buffer
 * @param offset
 * @param length
 *
 * @throws ISOEXCEPTION
 */
public BioServer(byte[] buffer, short offset, byte length) throws
    IOException {

    // Creating bio template and its proxy for clients
    this.proxyBioForClient= new ProxyBioTemplate();
    try {
        this.fullBio= BioBuilder.buildBioTemplate(BioBuilder.FINGERPRINT,
            this.TRY_LIMIT);
    }
    catch (BioException be) {
        IOException.throwIt((short) (BioServer.SW_BIOEXCEPTION_BASE +
            be.getReason()));
    }
    proxyBioForClient.fullBio= this.fullBio;
    if (buffer[offset] == 0)
        register();
    else
        register (buffer, (short) (offset+1), (byte) (buffer[offset]));
}

/**
 * To create an instance of the Applet subclass, the JCRE will call this
 * static method first. All the parameters below are ignored by this method.
 *
 * @param bArray the array containing installation parameters.
 * @param bOffset the starting offset in bArray.
 * @param blength the length in bytes of the parameter data in bArray. The
 *                maximum value of blength is 32.
 */
public static void install(byte[] bArray, short bOffset, byte blength) {

    // Instantiating the applet and register it with the JCRE
    new BioServer(bArray, bOffset, blength);
}

/**
 * Called by the JCRE to process an incoming APDU command. An applet is
 * expected to perform the action requested and return response data if any
 * to the terminal.

```

```

*
* @param aPDU the incoming APDU object
*
* @throws ISOException If fails to process APDU
*/
public void process(javacard.framework.APDU aPDU) throws
    javacard.framework.ISOException {

    // Catching BioException
    try {

        // Getting first five APDU header bytes
        byte[] apduBuffer= aPDU.getBuffer();

        // The JCRE also passes the SELECT APDU command to the applet.
        if (selectingApplet())
            return;

        // Controlling instruction class
        if (apduBuffer[ISO7816.OFFSET_CLA] != BIO_SERVER_CLA)
            ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

        short numTemplateBytes= aPDU.setIncomingAndReceive();

        byte ins= apduBuffer[ISO7816.OFFSET_INS];

        if (ins == BIO_INS_TEMPLATE_INIT) {
            if (initCalled)
                ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);
            else {

                // If method init throws an exception, flag initCalled is
                // not set to true
                fullBio.init(apduBuffer, ISO7816.OFFSET_CDATA,
                    numTemplateBytes);
                this.initCalled= true;

            }// End else
        }// End first if

        else if (ins == BIO_INS_TEMPLATE_UPDATE)
            fullBio.update(apduBuffer, ISO7816.OFFSET_CDATA,
                numTemplateBytes);

        else if (ins == BIO_INS_TEMPLATE_DOFINAL)
            fullBio.doFinal();

        else if (ins == BIO_VERIFY_INIT_MATCH)
            fullBio.initMatch(apduBuffer, ISO7816.OFFSET_CDATA,
                numTemplateBytes);

        else if (ins == BIO_VERIFY_MATCH) {

            short le= aPDU.setOutgoing();
            short matchCost= fullBio.match(apduBuffer, ISO7816.OFFSET_CDATA,
                numTemplateBytes);
            aPDU.setOutgoingLength((short) (2));
            //javacard.framework.Util.setShort(apduBuffer, (short)0,
            //matchCost);
            apduBuffer[0]= (byte)(matchCost >> 8);
            apduBuffer[1]= (byte)(matchCost & 0xFF);
            aPDU.sendBytes ((short)0 , (short)2 );
        }
    }
}

```

```

    }

    else if (ins == BIO_CHECK_VERIFY_RESULT) {
        if (!fullBio.isValidated())
            ISOException.throwIt(BioServer.SW_NOT_VALIDATED);
    }

    else if (ins == BIO_GET_PUBLIC_TEMPLATE_DATA) {

        short le= apdu.setOutgoing();

        apdu.setOutgoingLength((short) (JMOCMinutia.minNBytes + 1));

        // Writing last minutia in buffer
        short writtenBytes= fullBio.getPublicTemplateData((short) 0,
            apduBuffer, (short) 0, le);

        apdu.sendBytes ( (short)0 , writtenBytes );

    }

    // Only for test
    else if (ins == BIO_UNBLOCK_REFERENCE)
        fullBio.resetUnblockAndSetTryLimit(TRY_LIMIT);

    else if (ins == BIO_CHECK_ENROLLED) {

        if (!fullBio.isInitialized())
            ISOException.throwIt(BioServer.SW_NOT_INITIALIZED);

    }

    else
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);

}
catch (BioException be) {
    ISOException.throwIt((short)(BioServer.SW_BIOEXCEPTION_BASE +
        (short)(be.getReason())));
}
}

/**
 * Implements the shareable lookup by client via JCRE
 *
 * @param clientAid the AID object of the client applet: here is ignored.
 * @param parameter optional parameter byte. The parameter byte may be used
 * by the client to specify which shareable interface
 * object is being requested. Here there is only one
 * shareable interface so it's not checked.
 *
 * @return the shareable interface object: proxy object to biometric
 * template.
 */
public Shareable getShareableInterfaceObject(AID clientAid, byte parameter){

    return proxyBioForClient;

}

```



```

/**
 * Called by the JCRE to inform this applet that it has been selected. If
 * return false, it indicates to the JCRE that this Applet declines to be
 * selected. This happens when the template is blocked.
 *
 * @return <i>true</i> to indicate success, <i>false</i> if the template is
 *         blocked.
 */
public boolean select() {

    // Algorithm initialization can be inserted here
    // The applet declines to be selected if the template is blocked
    if (fullBio.getTriesRemaining() == 0)
        return false;
    else {
        fullBio.reset();
        return true;
    }
}

/**
 * Called by the JCRE to inform this currently selected applet that another
 * (or the same) applet will be selected.
 */
public void deselect() {

    // Resetting template state before the applet is deselected
    fullBio.reset();

}

} // End of class BioServer

```

```

/*
 * JMOCEdge.java
 *
 * Created on 15 febbraio 2003, 10.47
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package javaMatchOnCard;
```

```

/***** IMPORTS *****/
import javacard.framework.*;

```

```

/**
 * This class represents an edge in minutiae or neighbors graph.
 *
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class JMOCEdge {
```

```

    /**
     * Costant indicating no destination node for edges.
     */
    public static final byte noDest= -1;

```

```

    /**
     * Upper bound to edge cost.
     */
    public static final short M= (short) 32767;

```

```

    /**
     * Edge cost.
     */
    public short edgeCost= 0;

```

```

    /**
     * Destination node for this edge.

```

```
    */  
    public byte destNode= noDest;  
  
    /**  
     * Creates a new instance of JMOCEdge  
     */  
    public JMOCEdge() {  
    }  
}
```

```

/*
 * JMOCForwardStar.java
 *
 * Created on 11 febbraio 2003, 15.08
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package javaMatchOnCard;

```

```

/**
 * This class represents a forward star of a node in minutiae and neighbors
 * graph. Practically it's a vector of <code>JMOCEdge</code>.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class JMOCForwardStar {

    /**
     * Number of edges in this forward star.
     */
    public byte insPointEdge= 0;

    /**
     * Edge vector representing this forward star.
     */
    public JMOCEdge[] edges= null;

    /**
     * Number of edges in forward star if it's a minutiae graph forward star.
     * Must be different from <code>this.neighEdgeNumber</code>.
     */
    public static final byte minEdgeNumber= 6;

    /**
     * Number of edges in forward star if it's a neighbors graph forward star.
     */
    public static final byte neighEdgeNumber= 5;

    /**

```

```

    * Creates a new instance of JMOCForwardStar
    */
public JMOCForwardStar(byte nEdge) {

    byte index;

    this.edges= new JMOCEdge[nEdge];

    // Filling this forward star with JMOCEdges
    for (index= 0; index < nEdge; index++)
        this.edges[index]= new JMOCEdge();

    // Setting insPointEdge to this.neighEdgeNumber if it' a neighbors
    // forward star
    if (nEdge == JMOCForwardStar.neighEdgeNumber)
        this.insPointEdge= JMOCForwardStar.neighEdgeNumber;

}

/**
 * Inserts one edge in position <code>insPointEdge</code> of this forward
 * star and increments pointer <code>insPointEdge</code>.
 *
 * @param cost Cost of new edge.
 * @param dest Destination node of new edge.
 */
public void insertEdge(short cost, byte dest) {

    this.edges[this.insPointEdge].edgeCost= cost;
    this.edges[this.insPointEdge].destNode= dest;
    this.insPointEdge++;

}

/**
 * Return edge in this forward star with <code>dest</code> destination node.
 *
 * @param dest Destination node of the edge to be found.
 *
 * @return Edge in this forward star with <code>dest</code> destination
 *         node.
 */
public JMOCEdge getEdge(byte dest) {

    byte index;

    JMOCEdge returnEdge= null;

    for (index= 0; index < this.insPointEdge; index++) {

        if (this.edges[index].destNode == dest)
            returnEdge= this.edges[index];

    }

    return returnEdge;

}

```

```

/**
 * <i>Reinsert</i> an edge in this forward star. Edges are removed from
 * forward star during matching phase (in <code>changeMatch</code>): while
 * searching A-A path edges are inverted.
 *
 * @param edgeTemp Edge to reinsert.
 */
public void reinsert(JMOCEdge edgeTemp) {
    byte index;
    for (index= 0; index < this.insPointEdge; index++) {
        if (this.edges[index].destNode == JMOCEdge.noDest) {
            this.edges[index].destNode= edgeTemp.destNode;
            this.edges[index].edgeCost= (short) (-edgeTemp.edgeCost);
        }
    }
}
}
}

```

```

/*
 * JMOCInternalState.java
 *
 * Created on 9 febbraio 2003, 19.47
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```

package javaMatchOnCard;

```

```

/**
 * This class contains reference template and candidate template.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class JMOCInternalState {

    /**
     * List of minutiae in reference template.
     */
    public JMOCMinutia[] refTemplate= null;

    /**
     * List of minutiae in candidate template.
     */
    public JMOCMinutia[] candTemplate= null;

    /**
     * Number of minutiae a reference or a candidate template <i>MUST</i> have.
     */
    public static final byte minNumber= 25;

    /**
     * The flag in position <i>i</i> is <i>true</i> if reference minutia in
     * position <i>i</i> is inserted, <i>false</i> otherwise.
     */
    public boolean isInsertedMin[]= null;

    /**
     * The value of this field is a counter of reference minutiae inserted.

```

```

    */
    public byte refMinInserted= 0;

    /**
     * Max distance^2 between two minutiae while building minutiae graph.
     */
    public static final short maxDistance= 2500;

    /**
     * The value of thi field is used as a counter and as an index to minutiae
     * in candidate template: points to next free position. Its range of value
     * is <i>[0..this.minNumber]</i>.
     */
    public static byte candInsertedMinutiaeIndex= 0;

    /**
     * Creates a new instance of JMOCInternalState.
     */
    public JMOCInternalState() {

        byte index;

        // Creating minutiae list
        refTemplate= new JMOCMinutia[this.minNumber];
        candTemplate= new JMOCMinutia[this.minNumber];
        isInsertedMin= new boolean[this.minNumber];

        // Initializing minutiae list
        for (index= 0; index < JMOCInternalState.minNumber; index++) {
            refTemplate[index]= new JMOCMinutia();
            candTemplate[index]= new JMOCMinutia();
            isInsertedMin[index]= false;
        }
    }
}

```



```

/*
 * JMOCGraph.java
 *
 * Created on 11 febbraio 2003, 10.24
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package javaMatchOnCard;
```

```

/**
 * This class contains all data needed by matching algorithm like the graph,
 * <i>SPT</i> parent and cost label vectors, <i>SPT</i> node set and data for
 * storing matched node pairs.
 *
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class JMOCMatchData {
```

```
    // ----- GRAPH DATA -----
```

```

    /**
     * Number of nodes in one node set: on the whole there are <i>2*NumNodes +
     * 1</i> nodes in this bipartite graph.
     */

```

```
    public byte numNodes= 0;
```

```

    /**
     * Reference Template node forward stars.
     */

```

```
    public JMOCForwardStar[] firstSet= null;
```

```

    /**
     * Candidate Template node forward stars.
     */

```

```
    public JMOCForwardStar[] secondSet= null;
```

```

    /**
     * Forward star of root node.
     */

```

```
    public JMOCForwardStar rootFS= null;
```

```

// ----- SPT DATA -----
/**
 * This vector contains node parent during SPT visit.
 */
public byte[] parentVector= null;

/**
 * Flag indicating "no parent" for a node in <code>parentVector</code>.
 */
public static final byte noParent= -1;

/**
 * This vector contains cost labels during SPT.
 */
public short[] costVector= null;

// ----- SPT NODE QUEUE DATA -----
/**
 * Node set used by SPT procedure implemented as a FIFO list.
 */
public byte[] fifoNodeSet= null;

/**
 * Boolean in position <i>i</i> is <i>>true</i> if node <i>i</i> is inside
 * <code>fifoNodeSet</code> data structure.
 */
public boolean[] nodePresence= null;

/**
 * Pointer to node insertion in node set <code>fifoNodeSet</code>.
 */
public byte insNodeSet= 0;

/**
 * Pointer to node extraction in node set <code>fifoNodeSet</code>.
 */
public byte extNodeSet= 0;

// ----- MATCH SET DATA -----
/**
 * Value in position <i>i</i> is node id in second set matched with node
 * <i>i</i> in first set.
 */
public byte[] firstSetMatch= null;

/**
 * Flag in position <i>i</i> is <i>>true</i> if node <i>i</i> in second set
 * is matched with one node in first set of bipartite graph, <i>>false</i>
 * otherwise.
 */
public boolean[] secondSetIsMatched= null;

```

```

/**
 * Creates a new instance of JMOCGraph.
 */
public JMOCMatchData(byte nNode, byte edgeNumberInFS) {

    byte index;

    this.firstSet= new JMOCForwardStar[nNode];
    this.secondSet= new JMOCForwardStar[nNode];
    this.rootFS= new JMOCForwardStar(nNode);

    this.parentVector= new byte[nNode * 2];
    //this.parentVector= javacard.framework.JCSystem.makeTransientByteArray
    //(((short)(nNode * 2) ,javacard.framework.JCSystem.CLEAR_ON_RESET));
    this.costVector= new short[nNode * 2];
    //this.costVector= javacard.framework.JCSystem.makeTransientShortArray
    //(((short)(nNode * 2) ,javacard.framework.JCSystem.CLEAR_ON_RESET));
    this.fifoNodeSet= new byte[nNode * 2];
    //this.fifoNodeSet= javacard.framework.JCSystem.makeTransientByteArray
    //(((short)(nNode * 2) ,javacard.framework.JCSystem.CLEAR_ON_RESET));
    this.nodePresence= new boolean[nNode * 2];
    //this.nodePresence=
    //javacard.framework.JCSystem.makeTransientBooleanArray
    //(((short)(nNode * 2) ,javacard.framework.JCSystem.CLEAR_ON_RESET));
    this.firstSetMatch= new byte[nNode];
    //this.firstSetMatch= javacard.framework.JCSystem.makeTransientByteArray
    //(((short)(nNode * 2) ,javacard.framework.JCSystem.CLEAR_ON_RESET));
    this.secondSetIsMatched= new boolean[nNode];
    //this.secondSetIsMatched=
    //javacard.framework.JCSystem.makeTransientBooleanArray
    //(((short)(nNode * 2) ,javacard.framework.JCSystem.CLEAR_ON_RESET));

    this.numNodes= nNode;

    // Creating forward stars.
    for (index= 0; index < nNode; index++) {

        this.firstSet[index]= new JMOCForwardStar(edgeNumberInFS);
        this.secondSet[index]= new JMOCForwardStar((byte) 1);

    }
}

/**
 * Fill this neighbor graph using neighbors from two input minutiae.
 * Edge costs are differences between neighbors digest value.
 *
 * @param min1 Calculating graph edge costs from this minutia.
 * @param min2 Calculating graph edge costs from this minutia.
 */
public void fillWithNeighbors(JMOCMinutia min1, JMOCMinutia min2) {

    byte index;
    byte index2;

    short tempCost= 0;

    for (index= 0; index < JMOCMinutia.neighborsNumber; index++) {

```

```

        for (index2= 0; index2 < JMOCForwardStar.neighEdgeNumber; index2++)
    {

        // Taking feature difference absolute value
        tempCost= (short) (min1.neighborsVector[index] -
                           min2.neighborsVector[index2]);
        if (tempCost < 0)
            tempCost= (short) (-tempCost);

        // Inserting new edge
        this.firstSet[index].edges[index2].edgeCost= tempCost;
        this.firstSet[index].edges[index2].destNode= (byte) (index2 +
                                                             this.numNodes);

    }
}

// Resetting graph data structure
for (index= 0; index < this.numNodes; index++) {

    // Resetting second set forward star
    this.secondSet[index].edges[0].edgeCost= 0;
    this.secondSet[index].edges[0].destNode= JMOCEdge.noDest;

    // Resetting root node forward star destination nodes
    this.rootFS.edges[index].destNode= index;

    // Resetting match data
    this.firstSetMatch[index]= JMOCEdge.noDest;
    this.secondSetIsMatched[index]= false;

}

}

/**
 * Called before match on minutiae graph: resets data structure.
 */
public void resetMinutiaeData() {

    byte index;

    for (index= 0; index < this.numNodes; index++) {

        // Resetting first set forward star
        this.firstSet[index].insPointEdge= 0;

        // Resetting second set forward star
        this.secondSet[index].edges[0].edgeCost= 0;
        this.secondSet[index].edges[0].destNode= JMOCEdge.noDest;

        // Resetting root node forward star destination nodes
        this.rootFS.edges[index].destNode= index;

        // Resetting match data
        this.firstSetMatch[index]= JMOCEdge.noDest;
        this.secondSetIsMatched[index]= false;

    }

}
}

```

```

/**
 * Inserts a node in SPT node collection managed as a FIFO list.
 *
 * @param node Node label to insert.
 */
public void insNodeSet(byte node, short[] d) {

    this.fifoNodeSet[this.insNodeSet]= node;
    this.insNodeSet= (byte) (((this.insNodeSet + 1)) %
                             this.fifoNodeSet.length);
    this.nodePresence[node]= true;

}

/**
 * Extracts a node in SPT node collection managed as a FIFO list.
 *
 * @return Node label extracted.
 */
public byte extNodeSet() {

    byte returnByte;
    returnByte= this.fifoNodeSet[this.extNodeSet];
    this.extNodeSet= (byte) (((this.extNodeSet + 1)) %
                             this.fifoNodeSet.length);
    this.nodePresence[returnByte]= false;
    return returnByte;

}

/**
 * Returns <i>true</i> if parameter node is not in node collection,
 * <i>false</i> otherwise.
 */
public boolean isNotInNodeSet(byte node) {

    return !(this.nodePresence[node]);

}

}

```

```

/*
 * JMOCMatcher.java
 *
 * Created on 11 febbraio 2003, 17.58
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package javaMatchOnCard;
```

```

/***** IMPORTS *****/
import javacard.framework.*;

```

```

/**
 * This class contains matching algorithm.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class JMOCMatcher {
```

```

    /**
     * Flag indicating spt method has not found A-A path in the graph
     */
    public static final byte noEnd= -1;

```

```

    /**
     * Upper bound to Alternating-Augmenting path found by SPT procedure.
     */
    public static final short maxAAPathCost= (short) 32767;

```

```

    /**
     * Creates a new instance of JMOCMatcher.
     */
    public JMOCMatcher() {

```

```

    }

```

```

/**
 * Method used for matching both minutiae and neighbors bipartite graph.
 *
 * @param data Graph (and correlated data structures) to match.
 *
 * @return Assignment cost.
 */
public short assign(JMOCMatchData data) {

    byte counter= 0;
    byte endNode;

    // Assignment cost
    short assignmentCost= 0;

    while (true) {

        endNode= spt(data);
        if (endNode == JMOCMatcher.noEnd)
            break;

        assignmentCost+= changeAssignment(data, endNode);

        // Counting number of nodes matched
        counter++;

    }

    // Calculating match mean.
    if (data.numNodes != JMOCMinutia.neighborsNumber) {
        if (counter > 0)
            assignmentCost= (short) (assignmentCost / counter);
        else
            assignmentCost= (short) 32767;
    }

    //System.out.println("Assignment cost " + this.assignmentCost);

    // Return assignment cost
    return assignmentCost;
}

/**
 * Finds Shortest Path Tree on input graph. Returns last node of A-A path
 * found by SPT or <code>JMOCMatcher.noEnd</code> if no A-A path has been
 * found.
 *
 * @param data SPT runs over this graph.
 *
 * @return node in second set with the lowest cost label and not yet
 *         matched, which is also the last A-A path node.
 */
private byte spt(JMOCMatchData data) {

    byte index;

    byte choosenNode= JMOCMatcher.noEnd;
    short choosenNodeCost= JMOCMatcher.maxAAPathCost;

    // Initializing data strucuters

```

```

for (index= 0; index < data.costVector.length; index++) {

    // Resetting SPT vectors (vector second half)
    data.parentVector[index]= JMOCMatchData.noParent;

    // Seeting labels to upper bound value (vector second half)
    data.costVector[index]= JMOCEdge.M;

    // Resetting node presence vector (vector second half)
    data.nodePresence[index]= false;

}

// Resetting node set pointer
data.insNodeSet= 0;
data.extNodeSet= 0;

// Visiting first set of node in bipartite set starting from root node
for (index= 0; index < data.rootFS.edges.length; index++) {

    if (data.rootFS.edges[index].destNode != JMOCEdge.noDest) {

        data.insNodeSet(data.rootFS.edges[index].destNode,
                        data.costVector);
        data.costVector[index]= 0;

    }

}

// SPT Loop
byte extNode= JMOCMatcher.noEnd;
JMOCEdge tempEdge= null;
short newCost= 0;
while (data.insNodeSet != data.extNodeSet) {

    extNode= data.extNodeSet();

    // Node in first set of bipartite graph
    if (extNode < data.numNodes) {

        for (index= 0; index < data.firstSet[extNode].insPointEdge;
            index++) {

            tempEdge= data.firstSet[extNode].edges[index];

            // Controlling if this edge has a destination node
            if (tempEdge.destNode != JMOCEdge.noDest) {

                newCost= (short) (data.costVector[extNode] +
                                tempEdge.edgeCost);

                if (newCost < data.costVector[tempEdge.destNode]) {

                    data.parentVector[tempEdge.destNode]= extNode;
                    data.costVector[tempEdge.destNode]= newCost;

                    // Controlling if destination node is a destination
                    // node of A-A path
                    if (!data.secondSetIsMatched[tempEdge.destNode -
                        data.numNodes]) {
                        if (tempEdge.edgeCost < choosenNodeCost) {

```



```

        choosenNode= tempEdge.destNode;
        choosenNodeCost= tempEdge.edgeCost;
    }
}

if (data.isNotInNodeSet(tempEdge.destNode))
    data.insNodeSet(tempEdge.destNode,
        data.costVector);
}

}

}

// Node in second set of bipatite graph
else {

    // Nodes in second set have only one edge
    tempEdge= data.secondSet[extNode - data.numNodes].edges[0];

    if (tempEdge.destNode != JMOCEdge.noDest) {

        newCost= (short) (data.costVector[extNode] +
            tempEdge.edgeCost);

        if (newCost < data.costVector[tempEdge.destNode]) {

            data.parentVector[tempEdge.destNode]= extNode;
            data.costVector[tempEdge.destNode]= newCost;

            if (data.isNotInNodeSet(tempEdge.destNode))
                data.insNodeSet(tempEdge.destNode, data.costVector);

        }

    }

}

} // End of while loop

// Returning found node
return (choosenNode);

}

/**
 * Changes matching according with SPT visit (stored in <code>data</code>):
 * <code>endNode</code> is the last node in found A-A path. Increments
 * current match set cardinality by one.
 * Uses A-A path found by SPT.
 *
 * @param data Changes graph edges and uses SPT parent and cost vectors.
 * @param endNode Last A-A path node found by SPT
 */
private short changeAssignment(JMOCMatchData data, byte endNode) {

    JMOCEdge tempEdge= null;
    short returnCost= 0;
    byte fromNode= data.parentVector[endNode];

```

```

//JMOCEdge tempEdge;
byte oddEven= 1;

while (fromNode != JMOCMatchData.noParent) {

    // Edge from first set node to second set node case
    if ((oddEven % 2) != 0) {

        data.secondSetIsMatched[endNode - data.numNodes]= true;
        data.firstSetMatch[fromNode]= endNode;

        // Changing edge direction in graph
        // Removing edge from first set node forward star
        tempEdge= data.firstSet[fromNode].getEdge(endNode);

        tempEdge.destNode= JMOCEdge.noDest;

        // Increment in assignment cost
        //System.out.println(" - Cost : " + tempEdge.edgeCost);
        returnCost+= tempEdge.edgeCost;

        // Inserting edge in second set node forward star
        data.secondSet[endNode - data.numNodes].edges[0].destNode=
            fromNode;
        data.secondSet[endNode - data.numNodes].edges[0].edgeCost=
            (short) (- tempEdge.edgeCost);

        endNode= fromNode;
        fromNode= data.parentVector[fromNode];

    }

    // Edge from second set node to first set node case
    else {

        data.firstSet[endNode].reinsert(data.secondSet[fromNode -
            data.numNodes].edges[0]);

        data.secondSet[fromNode - data.numNodes].edges[0].destNode=
            JMOCEdge.noDest;

        // Decrementing assignment cost
        returnCost+= data.secondSet[fromNode -
            data.numNodes].edges[0].edgeCost;
        endNode= fromNode;
        fromNode= data.parentVector[fromNode];

    }

    oddEven++;

} // End while loop

// Removing from root fs
data.rootFS.edges[endNode].destNode= JMOCEdge.noDest;

return returnCost;

}

}

```

```

/*
 * Minutia.java
 *
 * Created on 8 febbraio 2003, 10.05
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

package javaMatchOnCard;

/**
 * This class represents a minutia stored in reference and candidate template.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class JMOCMinutia {

    /**
     * Minutia byte occupation.
     */
    public static final byte minNBytes= 16;

    /**
     * Minutia x position.
     */
    public short xPos= 0;

    /**
     * Minutia y position
     */
    public short yPos= 0;

    /**
     * Minutia reliability.
     */
    public byte reliability= 0;

    /**
     * Minutia type: type value is <i>0</i> for "ridge ending" and <i>1</i> for

```

```

    * "bufurcation".
    */
    public byte type= 0;

    /**
     * Stores this minutia features digest.
     */
    public short[] neighborsVector= null;

    /**
     * Number of neighbors per minutia.
     */
    public static final byte neighborsNumber= 5;

    /**
     * Creates a new instance of Minutia.
     */
    public JMOCMinutia() {
        this.neighborsVector= new short [neighborsNumber];
    }

}

```

```

/*
 * JMOCOwnerBioTemplate.java
 *
 * Created on 29 gennaio 2003, 21.29
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package javaMatchOnCard;
```

```

/***** IMPORTS *****/
//import org.javacardforum.javacard.biometry.*;
import javacard.framework.*;

```

```

/**
 * This class implements
 * <i>org.javacardforum.javacard.biometry.OwnerBioTemplate</i>
 * interface created by <i>JavaCardForum</i>.
 * Defines methods for fingerprint enrollment and matching.
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class JMOCOwnerBioTemplate extends OwnerBioTemplate {
```

```

    /**
     * This field value is true when the template is enrolled and false
    otherwise.
    */

```

```
    private boolean isEnrolled= false;
```

```

    /**
     * This field value is true if the template has been successfully checked
     * since the last card reset or last call to methos <code>reset()</code>.
    */

```

```
    private boolean isValidated= false;
```

```

    /**
     * Maximum match try limit for this template.
    */

```

```
    private byte tryLimit= 0;
```

```

/**
 * This field value represents number of times remaining that an incorrect
 * candidate template can be presented before the reference template is
 * blocked.
 */
private byte triesRemaining= 0;

/**
 * This field is <i>true</i> if a matching session has been started,
 * <i>false</i> otherwise.
 */
private boolean matchInProgress= false;

/**
 * This field is <i>true</i> if an enrollment session has been started,
 * <i>false</i> otherwise.
 */
private boolean enrollmentInProgress= false;

/**
 * Matching threshold: below this value the match is positive.
 */
private static final short matchingThreshold= 1300;

/**
 * Internal state: contains reference template, candidate template and some
 * counters and flags.
 */
private JMOCInternalState state= null;

/**
 * Data used to match minutiae from reference and candidate template.
 */
private JMOCMatchData minutiaeData= null;

/**
 * Bipartite data used to match neighbors from two different minutiae.
 */
private JMOCMatchData neighborsData= null;

/**
 * Matcher class reference.
 */
private JMOCMatcher matcher= null;

/**
 * Creates a new instance of JMOCOwnerBioTemplate.
 *
 * @param tryLimit maximum match try limit value for this template.
 */
public JMOCOwnerBioTemplate(byte tryLimit) {
    // Setting try limit

```

```

        this.tryLimit= tryLimit;

        // Setting remaining tries
        this.triesRemaining= tryLimit;

        // Creating minutiae vector and correlated internal state
        this.state= new JMOCInternalState();

        // Creating minutiae data
        this.minutiaeData= new JMOCMatchData(JMOCInternalState.minNumber,
                                              JMOCForwardStar.minEdgeNumber);

        // Creating neighbors data
        this.neighborsData= new JMOCMatchData(JMOCMinutia.neighborsNumber,
                                              JMOCForwardStar.neighEdgeNumber);

        // Creating matcher class
        this.matcher= new JMOCMatcher();
    }

    /**
     * Finalizes the enrollment of a reference template. Final action of
     * enrollment is to designate a reference template as being complete and
     * ready for use (marks the reference as initialized, resets the try counter
     * and unblocks the reference). This routine may also include some error
     * checking prior to the validation of reference template as ready for use.
     *
     * @throws BioException
     */
    public void doFinal() throws BioException {

        // Controlling if the template is already enrolled
        if (this.isEnrolled == true)
            BioException.throwIt(BioException.ILLEGAL_USE);

        // Controlling if enrollment session has been started
        if (this.enrollmentInProgress == false)
            BioException.throwIt(BioException.ILLEGAL_USE);

        // Controlling if inserted minutiae reach the maximum number
        if (state.refMinInserted != state.minNumber)
            BioException.throwIt(BioException.ILLEGAL_USE);

        // Resetting try counter to maximum
        this.triesRemaining= this.tryLimit;

        // Ending enrollment session
        this.enrollmentInProgress= false;

        // Setting enrolled flag to true: template enrolled
        this.isEnrolled= true;
    }

    /**
     * Get the biometric type. In this case is returned
     * <code>BioBuilder.FINGER_GEOMETRY</code> since this algorithm regards
     * fingerprint matching.

```

```

*
* @return biometric general type: in this case
*         code>BioBuilder.FINGER_GEOMETRY</code>
*/
public byte getBioType() {
    return BioBuilder.FINGERPRINT;
}

/**
 * Get public part of the reference template. Returns in the array,
 * information about last minutia inserted: it can be used during enrollment
 * phase to check data correctness.
 *
 * @param publicOffset starting offset within the public data (ignored in
 *                    this API implementation.
 * @param dest destination byte array.
 * @param destOffset starting offset within the destination byte array.
 * @param length maximum bytelength of the requested data.
 *
 * @return number of bytes really written in the destination byte array. If
 *         there's an error in input parameters, returns <i>0</i>, else
 *         returns <code>JMOCMinutia.minNBytes + 1</code> wich is the number
 *         of byte occupied by minutia information plus minutia index
 *
 * @throws BioException if the reference template is uninitialized.
 */
public short getPublicTemplateData(short publicOffset, byte[] dest, short
                                   destOffset, short length) throws
                                   BioException {

    byte index= JMOCInternalState.minNumber - 1;
    byte index2;
    short offset;
    JMOCMinutia tempMin= null;

    // Controlling if the template is enrolled
    if ((!this.isEnrolled) && (!this.enrollmentInProgress))
        BioException.throwIt(BioException.NO_TEMPLATES_ENROLLED);

    // Controlling if at least one minutia is in reference template
    if (state.refMinInserted == 0)
        return 0;

    // Controlling input parameters
    if ((dest == null) || (destOffset < 0))
        return 0;

    // Controlling array dest length
    if (((short)(destOffset + JMOCMinutia.minNBytes)) >= dest.length)
        return 0;

    while (index >= 0) {
        if (state.isInsertedMin[index]) {
            // Writing minutia information in dest array
            tempMin= state.refTemplate[index];
            dest[destOffset]= (byte)(index);
            //javacard.framework.Util.setShort(dest, (short) (destOffset +

```



```

        //1), tempMin.xPos);
        dest[(short) (destOffset + 1)] = (byte) (tempMin.xPos >> 8);
        dest[(short) (destOffset + 2)] = (byte) (tempMin.xPos & 0xFF);
        //javacard.framework.Util.setShort(dest, (short) (destOffset +
        //3), tempMin.yPos);
        dest[(short) (destOffset + 3)] = (byte) (tempMin.yPos >> 8);
        dest[(short) (destOffset + 4)] = (byte) (tempMin.yPos & 0xFF);
        dest[(short) (destOffset + 5)] = tempMin.reliability;
        dest[(short) (destOffset + 6)] = tempMin.type;
        offset = (short) (destOffset + 7);

        // Writing minutia neighbors
        for (index2 = 0; index2 < JMOCMinutia.neighborsNumber; index2++) {
            //javacard.framework.Util.setShort(dest, offset,
            //tempMin.neighborsVector[index]);
            dest[offset] = (byte) (tempMin.neighborsVector[index2] >> 8);
            dest[(short) (offset + 1)] = (byte)
                (tempMin.neighborsVector[index2]
                 & 0xFF);
            offset = (short) (offset + 2);
        }

        // Jumping outside loop
        break;
    } // End If

    index--;

} // End while

// Returning bytes written in dest array
if (index < 0)
    return 0;
else
    return ((short) (JMOCMinutia.minNBytes + 1));

}

/**
 * Returns the number of times remaining that an incorrect candidate
 * template can be presented before the reference template is blocked.
 *
 * @return the number of tries remaining.
 */
public byte getTriesRemaining() {
    return this.triesRemaining;
}

/**
 * Get the matching algorithm version and ID.
 *
 * @param dest destination byte array.
 * @param destOffset starting offset within the destination byte array.
 *
 * @return number of bytes written in the destination byte array.
 */

```

```

public short getVersion(byte[] dest, short destOffset) {

    if ((dest != null) && (destOffset >= 0) && (dest.length > ((short)
        (destOffset + 1)))) {

        // Inserting algorithm version
        dest[destOffset]= (byte) (1);

        // Inserting algorithm ID
        dest[(short)(destOffset + 1)]= (byte) (1);
        return 2;
    }
    else

        // Otherwise write nothing
        return 0;
}

/**
 * Initializes the enrolment of a reference template. It is also used to
 * update a reference template. It resets the validated flag and, in the
 * update case, uninitializes the previous reference.
 *
 * @param bArray byte array containing the data of the template.
 * @param offset starting offset in the bArray.
 * @param length byte length of the template data in the bArray.
 *
 * @throws BioException If fails to start enrollment phase: in this
 *                       implementation is never thrown.
 */
public void init(byte[] bArray, short offset, short length) throws
    BioException {

    byte index;

    // Exit from match session in progress
    if (this.matchInProgress) {
        this.matchInProgress= false;
        state.candInsertedMinutiaeIndex= 0;
    }

    // Starting enrollment session
    this.enrollmentInProgress= true;

    // Resetting validated flag
    this.isValidated= false;

    // Unenrolling previous template
    this.isEnrolled= false;

    // Setting minutiae inserted in reference template to 0
    state.refMinInserted= 0;

    // Resetting insertion flag to false for every minutia
    for (index= 0; index < state.isInsertedMin.length; index++)
        state.isInsertedMin[index]= false;
}

```

```

/**
 * Initialize or reinitialize a biometric matching session.
 *
 * @param candidateTemplate byte array containing the data of the template.
 *                          (not used).
 * @param offset starting offset in the candidateTemplate (not used).
 * @param length byte length of the template data in the candidateTemplate
 *              (not used).
 *
 * @return <code>BioTemplate.MATCH_NEEDS_MORE_DATA</code> because the match
 *         needs candidate template minutiae.
 *
 * @throws BioException If fails to start new matching phase (e.g. template
 *                       is blocked).
 */
public short initMatch(byte[] candidateTemplate, short offset, short length)
    throws BioException {

    this.isValidated= false;

    // Controlling if the card is blocked
    if (this.triesRemaining == 0)
        BioException.throwIt(BioException.ILLEGAL_USE);

    // Controlling if the template is enrolled
    if (this.isEnrolled == false)
        BioException.throwIt(BioException.NO_TEMPLATES_ENROLLED);

    // Controlling if a matching session is already started
    if (this.matchInProgress == true) {

        // Setting to zero candidate template minutiae counter
        state.candInsertedMinutiaeIndex= 0;

    }

    // Starting a new matching session
    this.matchInProgress= true;

    // Setting to zero candidate template minutiae counter
    state.candInsertedMinutiaeIndex= 0;

    // Returning MATCH_NEEDS_MORE_DATA
    return BioTemplate.MATCH_NEEDS_MORE_DATA;
}

/**
 * Returns true if the reference template is completely loaded and ready for
 * matching functions.
 *
 * @return Value of field <code>isEnrolled</code>: <i>true</i> if
 *         initialize <i>false</i> otherwise.
 */
public boolean isInitialized() {

    return this.isEnrolled;
}

```

```

/**
 * Returns true if the template has been successfully checked since the last
 * card reset or last call to reset().
 *
 * @return Value of field <code>isValidated</code>: <i>true</i> if validated
 *         <i>false</i> otherwise.
 */
public boolean isValidated() {

    return this.isValidated;

}

/**
 * Continues a biometric matching session and automatically ends when all
 * of candidate template minutiae are sent.
 *
 * @param candidate The data or part of the data of the candidate template.
 * @param offset starting offset into the candidate array where the
 *               candidate data is to be found.
 * @param length number of bytes to be taken from the candidate array.
 *
 * @return The matching score.
 *
 * @throws BioException If fails to match candidate end reference template
 *                       (e.g. if reference template is not enrolled)
 */
public short match(byte[] candidate, short offset, short length) throws
    BioException {

    byte index;
    byte index2;
    short scanningOffset= offset;
    short xPosTemp;
    short yPosTemp;
    byte reliabilityTemp;
    byte typeTemp;

    // Controlling if the template is enrolled
    if (this.isEnrolled != true)
        BioException.throwIt(BioException.NO_TEMPLATES_ENROLLED);

    // Controlling if a matching session is started: if the method initMatch
    // is been called before
    if (this.matchInProgress == false)
        BioException.throwIt(BioException.ILLEGAL_USE);

    if (((length % JMOCMinutia.minNBytes) != 0) || (length == 0))
        BioException.throwIt(BioException.INVALID_DATA);

    // Inserting new minutiae phase
    while (length >= ((short) (scanningOffset + JMOCMinutia.minNBytes -
        ISO7816.OFFSET_CDATA))) {

        // Retrieving minutia data from byte buffer
        //xPosTemp= javacard.framework.Util.getShort(candidate,
        //scanningOffset);
        xPosTemp= (short) (((short) (candidate[scanningOffset]) << 8) +
            (short) ((candidate[(short) (scanningOffset + 1)]) &
                0xFF));
        //yPosTemp= javacard.framework.Util.getShort(candidate, (short)

```

```

        //(scanningOffset + 2));
        yPosTemp= (short) (((short) (candidate[(short) (scanningOffset +
            2)]) < 8) + (short) ((candidate[(short) (scanningOffset
            + 3)]) & 0xFF));
        reliabilityTemp= candidate[(short) (scanningOffset + 4)];
        typeTemp= candidate[(short) (scanningOffset + 5)];

        // Controlling retrieved minutia data
        if ((typeTemp != 0) && (typeTemp != 1))
            BioException.throwIt(BioException.INVALID_DATA);

        // Inserting minutia data in candidate template
        state.candTemplate[state.candInsertedMinutiaeIndex].xPos= xPosTemp;
        state.candTemplate[state.candInsertedMinutiaeIndex].yPos= yPosTemp;
        state.candTemplate[state.candInsertedMinutiaeIndex].reliability=
            reliabilityTemp;
        state.candTemplate[state.candInsertedMinutiaeIndex].type= typeTemp;

        // Updating scanning offset
        scanningOffset+= 6;

        // Inserting minutia neighbors
        for (index= 0; index < JMOCMinutia.neighborsNumber; index++) {
            //state.refTemplate[state.candInsertedMinutiaeIndex].
            //neighborsVector[index]=
            //javacard.framework.Util.getShort(candidate, scanningOffset);

            state.candTemplate[state.candInsertedMinutiaeIndex].
                neighborsVector[index]=(short) (((short)
                (candidate[scanningOffset]) < 8) +
                (short) ((candidate[(short)
                (scanningOffset + 1)]) & 0xFF)));
            scanningOffset+= 2;
        }

        // Updating internal state counters and flags
        state.candInsertedMinutiaeIndex++;

        if (state.candInsertedMinutiaeIndex == state.minNumber)
            break;
    } // End while

    if (state.candInsertedMinutiaeIndex < state.minNumber)
        return (BioTemplate.MATCH_NEEDS_MORE_DATA);

    else {

        // Cost of the match
        short matchCost;

        // Closing matching phase
        this.matchInProgress= false;

        // Resetting minutiae graph data structures before matching
        minutiaeData.resetMinutiaeData();

        // Starting building graph
        for (index= 0; index < JMOCInternalState.minNumber; index++) {

            for (index2= 0; index2 < JMOCInternalState.minNumber; index2++)

```

```

        // Controlling forward star capacity
        if (minutiaeData.firstSet[index].insPointEdge >=
            (JMOCForwardStar.minEdgeNumber))
            break;

        // Controlling if minutiae type are the same
        /*if (state.refTemplate[index].type !=
            state.candTemplate[index2].type)
            continue;*/

        xPosTemp= (short) (state.refTemplate[index].xPos -
                           state.candTemplate[index2].xPos);
        yPosTemp= (short) (state.refTemplate[index].yPos -
                           state.candTemplate[index2].yPos);

        // xPoTemp stores now euclidean distance^2 between minutiae
        xPosTemp= (short) ((xPosTemp*xPosTemp) +
                           (yPosTemp*yPosTemp));

        // Controlling distance between minutiae
        if (xPosTemp > JMOCInternalState.maxDistance)
            continue;

        // Filling neighbor graph with these two minutiae neighbors
        // feature differences
        neighborsData.fillWithNeighbors(state.refTemplate[index],
                                       state.candTemplate[index2]);

        // Inserting assignment value of neighbor graph in minutiae
        // graph
        minutiaeData.firstSet[index].insertEdge
            (matcher.assign(neighborsData), (byte) (index2
            + JMOCInternalState.minNumber));

    }

} // Ending buildin graph

// Matching minutiae graph
matchCost= matcher.assign(minutiaeData);

if (matchCost < matchingThreshold) {

    // Positive match
    this.isValidated= true;
    matchCost= BioTemplate.MINIMUM_SUCCESSFUL_MATCH_SCORE;
}
else {

    // Negative match
    this.triesRemaining--;
    this.isValidated= false;
    matchCost= 0;
}

return matchCost;
}
}

```

```

/**
 * Resets the reference validated flag. This could be appropriate as a last
 * action after an access is completed.
 */
public void reset() {

    this.isValidated= false;

    // Aborting matching phase
    if (this.matchInProgress) {

        this.matchInProgress= false;

        // Setting to zero candidate template minutiae counter
        state.candInsertedMinutiaeIndex= 0;

    }

}

/**
 * Resets the validated flag, unblocks the reference, updates the try limit
 * value and resets the try counter to the try limit value.
 *
 * @param newTryLimit the number of tries allowed before the reference is
 * blocked. <i>newTryLimit</i> must be >= 1.
 *
 * @throws BioException If fails to unblock template (e.g. if newTryLimit
 * value is less than 1)
 */
public void resetUnblockAndSetTryLimit(byte newTryLimit)throws BioException{

    // Controlling new try limit
    if (newTryLimit < 1)
        BioException.throwIt(BioException.ILLEGAL_VALUE);

    // Resetting the validated flag
    this.isValidated= false;

    // Updating try limit with the new value
    this.tryLimit= newTryLimit;

    // Resetting try counter to new maximum value
    this.triesRemaining= newTryLimit;

}

/**
 * Continues the enrolment of a reference template. Must be called several
 * times to send minutiae to build reference template.
 *
 * @param bArray byte array containing the data of the template.
 * @param offset starting offset in the bArray.
 * @param length byte length of the template data in the bArray.
 *
 * @throws BioException If fails to save minutie in template (e.g. if
 * enrollment phase is not started.
 */
public void update(byte[] bArray, short offset, short length) throws

```

```

        BioException {

byte index;

// Controlling if the template is already enrolled
if (this.isEnrolled == true)
    BioException.throwIt(BioException.ILLEGAL_USE);

// Controlling if enrollment session has been started
if (this.enrollmentInProgress == false)
    BioException.throwIt(BioException.ILLEGAL_USE);

// Controlling if number of input bytes represents one or more minutiae
if (((length % (JMOCMinutia.minNBytes + 1)) != 0) || (length == 0))
    BioException.throwIt(BioException.INVALID_DATA);

// Inserting new minutiae phase
short scanningOffset= offset;
byte position;
short xPosTemp;
short yPosTemp;
byte reliabilityTemp;
byte typeTemp;

while (length >= ((short) (scanningOffset + JMOCMinutia.minNBytes -
ISO7816.OFFSET_CDATA))) {

    position= bArray[scanningOffset];
    scanningOffset++;

    if ((position >= state.minNumber) || (position < 0))
        BioException.throwIt(BioException.INVALID_DATA);
    else {

        // Retrieving minutia data from byte buffer
        //xPosTemp= javacard.framework.Util.getShort(bArray,
        //scanningOffset);
        xPosTemp= (short) (((short) (bArray[scanningOffset]) << 8) +
            ((short) (bArray[(short) (scanningOffset + 1)]) &
            0xFF));

        //yPosTemp= javacard.framework.Util.getShort(bArray, (short)
        //(scanningOffset + 2));
        yPosTemp= (short) (((short) (bArray[(short) (scanningOffset +
            2)]) << 8) + ((short) (bArray[(short) (scanningOffset +
            3)]) & 0xFF));

        reliabilityTemp= bArray[(short) (scanningOffset + 4)];
        typeTemp= bArray[(short) (scanningOffset + 5)];

        // Controlling retrieved minutia data
        if ((typeTemp != 0) && (typeTemp != 1))
            BioException.throwIt(BioException.INVALID_DATA);

        // Inserting minutia data in reference template
        state.refTemplate[position].xPos= xPosTemp;
        state.refTemplate[position].yPos= yPosTemp;
        state.refTemplate[position].reliability= reliabilityTemp;
        state.refTemplate[position].type= typeTemp;

        // Updating scanning offset
        scanningOffset+= 6;
    }
}

```



```

// Inserting minutia neighbors
for (index= 0; index < JMOCMinutia.neighborsNumber; index ++) {
    //state.refTemplate[position].neighborsVector[index]=
    //javacard.framework.Util.getShort(bArray, scanningOffset);
    state.refTemplate[position].neighborsVector[index]= (short)
        (((short) (bArray[scanningOffset]) << 8) + ((short)
            (bArray[(short) (scanningOffset + 1)]) & 0xFF));
    scanningOffset+= 2;
}

// Updating internal state counters and flags
if (! state.isInsertedMin[position]) {
    state.isInsertedMin[position] = true;
    state.refMinInserted++;
}

} // End else
} // End while

} // End method

```

```

}

```

```

/*
 * ProxyBioTemplate.java
 *
 * Created on 29 gennaio 2003, 19.27
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

```

```
package javaMatchOnCard;
```

```

/***** IMPORTS *****/
import javacard.framework.*;
//import org.javacardforum.javacard.biometry.*;

```

```

/**
 * The <code>ProxyBioTemplate</code> contains the biometric template and
 * provides access to matching funciotns to other applets.
 *
 *
 * @author Francesco Santini
 * @version 1.0
 */

```

```
public class ProxyBioTemplate extends SharedBioTemplate {
```

```

    /**
     * This template is not directly shared outside the package.
     */
    public JMOCOwnerBioTemplate fullBio;

```

```

    /**
     * Creates a new instance of ProxyBioTemplate
     */
    public ProxyBioTemplate() {
    }

```

```

    /**
     * Calls <code>JMOCOwnerBioTemplate.getBioType</code> on bio template.
     */
    public byte getBioType() {

        return fullBio.getBioType();
    }

```

```

}

/**
 * Calls <code>JMOCOwnerBioTemplate.getPublicTemplateData</code> on bio
 * template.
 */
public short getPublicTemplateData(short publicOffset, byte[] dest, short
                                destOffset, short length) throws
                                BioException {

    return fullBio.getPublicTemplateData(publicOffset, dest, destOffset,
                                        length);

}

/**
 * Calls <code>JMOCOwnerBioTemplate.getTriesRemaining</code> on bio
 * template.
 */
public byte getTriesRemaining() {

    return fullBio.getTriesRemaining();

}

/**
 * Calls <code>JMOCOwnerBioTemplate.getVersion</code> on bio template.
 */
public short getVersion(byte[] dest, short destOffset) {

    return fullBio.getVersion(dest, destOffset);

}

/**
 * Calls <code>JMOCOwnerBioTemplate.initMatch</code> on bio template.
 */
public short initMatch(byte[] candidateTemplate, short offset, short length)
                                throws BioException {

    return fullBio.initMatch(candidateTemplate, offset, length);

}

/**
 * Calls <code>JMOCOwnerBioTemplate.isInitialized</code> on bio template.
 */
public boolean isInitialized() {

    return fullBio.isInitialized();

}

/**
 * Calls <code>JMOCOwnerBioTemplate.isValidated</code> on bio template.
 */
public boolean isValidated() {

```

```

        return (fullBio.isValidated());
    }

    /**
     * Calls <code>JMOCOwnerBioTemplate.match</code> on bio template.
     */
    public short match(byte[] candidate, short offset, short length) throws
        BioException {

        return fullBio.match(candidate, offset, length);
    }

    /**
     * Calls <code>JMOCOwnerBioTemplate.reset</code> on bio template.
     */
    public void reset() {

        fullBio.reset();
    }
}

```

D.3 Classe Main EnrollMatch.

```
/*
 * EnrollMatch.java
 *
 * Created on 24 febbraio 2003, 16.24
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/***** IMPORTS *****/
import javaMatchOnCard.*;
//import org.javacardforum.javacard.biometry.*;
import algParsing.*;
import converting.*;
import minutiaeInformation.*;
import minutiaeSuperClasses.*;
import algMinInformation.*;
import minutiaeListConverting.*;
import java.io.*;
import java.util.*;

/**
 * This class tests <i>Match On Card</i> (MOC) on PC: tries to match two
 * fingerprints.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class EnrollMatch {

    /**
     * Creates a new instance of EnrollMatch.
     */
    public EnrollMatch() {
    }

    /**
     * Tries to match two fingerprints.
     */
}
```

```

*
* @param args the command line arguments: two text file with fingerprint
* data for matching algorithm.
*/
public static void main(String[] args) {

    System.out.println("Card simulator matching program started.\n");

    MinutiaeListConverter mlc= new MinutiaeListConverter();
    short matchResult= 0;

    if ((args.length > 3) || (args.length < 2)) {
        System.err.println("Error on input parameter number.");
        System.err.println("\tUsage: java EnrollMatch \"filename2\"
            \"filename2\"");
        System.exit(-1);
    }

    System.out.println("File opening phase.\n");

    FileReader fileReferenceInput= null;
    FileReader fileCandidateInput= null;

    // Opening first file
    try {
        fileReferenceInput= new FileReader(args[0]);
    }
    catch (Exception e) {
        System.err.println("Error opening first file (" + args[0] + "):");
        System.err.println("\t" + e.getMessage());
        System.exit(-2);
    }

    try {
        fileCandidateInput= new FileReader(args[1]);
    }
    catch (Exception e) {
        System.err.println("Error opening second file (" + args[1] + "):");
        System.err.println("\t" + e.getMessage());
        System.exit(-3);
    }

    // Creating biometric template
    JMOCOwnerBioTemplate ob= null;
    try {
        ob= BioBuilder.buildBioTemplate(BioBuilder.FINGERPRINT, (byte)3);
    }
    catch (BioException be) {
        System.err.println("Error in buildBioTemplate(): " +
            be.getReason());
        System.exit(-12);
    }

    // Creating Parser
    AlgParser minParser= new AlgParser();

    System.out.println(args[0] + " parsing phase.\n");

    // Creating minutiae list for the first file
    AlgMinutiaeList minutiaeReferenceList= null;
    try {
        minutiaeReferenceList= new AlgMinutiaeList();
    }

```

```

catch (MinutiaeListException mle) {
    System.err.println("Error creating first minutiae list: " +
        mle.getMessage());
    System.exit(-4);
}

// Parsing first file
try {
    minParser.parse(fileReferenceInput, minutiaeReferenceList);
}
catch (AlgParserException ape) {
    System.err.println("Error parsing first file (" + args[0] + "):");
    System.err.println("\t" + ape.getMessage());
    try {
        fileReferenceInput.close();
    }
    catch (Exception e) {
        System.err.println("Error closing " + args[0] + " file.");
    }
    System.exit(-5);
}

// Closing first file
try {
    fileReferenceInput.close();
}
catch (Exception e) {
    System.err.println ("Error closing " + args[0] + " file.");
    System.exit(-6);
}

// Converting first minutia list
Vector referenceCardList= null;
try {
    referenceCardList= Convert.convert(minutiaeReferenceList);
}
catch (ConvertException ce) {
    System.err.println("Error converting enrollment list: \n\t" +
        ce.getMessage());
    System.exit(-7);
}

System.out.println(args[1] + " parsing phase.\n");

// Creating minutiae list for the second file
AlgMinutiaeList minutiaeCandidateList= null;
try {
    minutiaeCandidateList= new AlgMinutiaeList();
}
catch (MinutiaeListException mle) {
    System.err.println("Error creating second minutiae list: " +
        mle.getMessage());
    System.exit(-8);
}

// Parsing second file
try {
    minParser.parse(fileCandidateInput, minutiaeCandidateList);
}
catch (AlgParserException ape) {
    System.err.println("Error parsing second file (" + args[1] + "):");
    System.err.println("\t" + ape.getMessage());
}

```

```

        try {
            fileCandidateInput.close();
        }
        catch (Exception e) {
            System.err.println("Error closing " + args[1] + " file.");
        }
        System.exit(-9);
    }

    // Closing second file
    try {
        fileCandidateInput.close();
    }
    catch (Exception e) {
        System.err.println ("Error closing " + args[1] + " file.");
        System.exit(-10);
    }

    // Converting second list
    Vector candidateCardList= null;
    try {
        candidateCardList= Convert.convert(minutiaeCandidateList);
    }
    catch (ConvertException ce) {
        System.err.println("Error converting enrollmt list: \n\t" +
                           ce.getMessage());
        System.exit(-11);
    }

    // Enrollment phase: INIT-UPDATE-DOFINAL

    try {
        ob.init(null, (short)(0), (short)(0));
    }
    catch (BioException be) {
        System.err.println("Error in init(): " + be.getReason());
        System.exit(-13);
    }

    for (int i= 0; i < referenceCardList.size(); i++) {

        byte[] uByte= new byte[17];
        CardMinutia tempMin= (CardMinutia) (referenceCardList.get(i));

        // Filling byte array
        short xPos= tempMin.getXPos();
        short yPos= tempMin.getYPos();
        uByte[0]= (byte) (i);
        uByte[1]= (byte) (xPos >> 8);
        uByte[2]= (byte) (xPos & 0xFF);
        uByte[3]= (byte) (yPos >> 8);
        uByte[4]= (byte) (yPos & 0xFF);
        uByte[5]= tempMin.getReliability();
        uByte[6]= tempMin.getType();
        short[] nVector= tempMin.getNeighborsVector();

        for (int j= 0; j < JMOCMinutia.neighborsNumber; j++) {
            uByte[7 + (j*2)]= (byte) (nVector[j] >> 8);
            uByte[8 + (j*2)]= (byte) (nVector[j] & 0xFF);
        }

        try {
            ob.update(uByte, (short) (0), (short) (uByte.length));

```



```

    }
    catch (BioException be) {
        System.err.println("Error in update(): " + be.getReason());
        System.exit(-14);
    }
}

try {
    ob.doFinal();
}
catch(BioException be) {
    System.err.println("Error in doFinal(): " + be.getReason());
    System.exit(-15);
}

// Matching phase: INITMATCH-MATCH

try {
    ob.initMatch(null, (short) (0), (short) (0));
}
catch(BioException be) {
    System.err.println("Error in initMatch(): " + be.getReason());
    System.exit(-5);
}

for (int j= 0; j < candidateCardList.size(); j++) {

    byte[] uByte= new byte[16];
    CardMinutia tempMin= (CardMinutia) (candidateCardList.get(j));

    // Filling byte array
    short xPos= tempMin.getXPos();
    short yPos= tempMin.getYPos();
    uByte[0]= (byte) (xPos >> 8);
    uByte[1]= (byte) (xPos & 0xFF);
    uByte[2]= (byte) (yPos >> 8);
    uByte[3]= (byte) (yPos & 0xFF);
    uByte[4]= tempMin.getReliability();
    uByte[5]= tempMin.getType();
    short[] nVector= tempMin.getNeighborsVector();
    //orderVector(nVector);
    for (int k= 0; k < JMOCMinutia.neighborsNumber; k++) {
        uByte[6 + (k*2)]= (byte) (nVector[k] >> 8);
        uByte[7 + (k*2)]= (byte) (nVector[k] & 0xFF);
    }

    // MATCHING!
    try {
        matchResult= ob.match(uByte, (short) (0), (short)
            (uByte.length));
    }
    catch (BioException be) {
        System.err.println("Error in match(): " + be.getReason());
        System.exit(-6);
    }
}

System.out.println("\nMatch Result " + matchResult + "\n");
System.out.println("End Test program.");
}
}

```

D.4 Classe Main CardFinalTest.

```
/*
 * CardFinalTest.java
 *
 * Created on 11 ottobre 2003, 17.04
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/***** IMPORTS *****/
import java.io.*;
import algParsing.AlgParser;
import algParsing.AlgParserException;
import algMinInformation.AlgMinutiaeList;
import minutiaeSuperClasses.MinutiaeListException;
import java.util.Vector;
import converting.*;
import javaMatchOnCard.*;

/**
 * This class has a <i>main</i> method that calculates <i>FAR</i> (False
 * Acceptance Rate) and <i>FRR</i> (False Rejection Rate) for the given
 * fingerprint database. Result is in the output text file. Test description can
 * be found at: <link>http://bias.csr.unibo.it/fvc2004/perfeval.asp</link>.
 * This test is writed for <i>"match on card"</i> algorithm version.
 *
 * @author Francesco Santini
 * @version 1.0
 */

public class CardFinalTest {

    /**
     * Match threshold: below this value, two fingerprints are positively
     * matched.
     */
    private static final float maxMatchCost= 200;

    /**
     * Creates a new instance of CardFinalTest.
     */
}
```

```

public CardFinalTest() {
}

/**
 * Java Card test <i>main</i>.
 *
 * @param args the command line arguments (two parameters): directory path
 *             of fingerprint text file database and path of the file with
 *             containing match result.
 */
public static void main(String[] args) {

    if (args.length != 2) {

        System.err.println("Error on input parameters.");
        System.err.println("\tUsage: java FinalTest input_directory
                           outputfile");
        System.exit(-1);

    }

    String dirBase= new String(args[0]);
    String outfi= new String(args[1]);

    // Creating log file
    FileOutputStream out= null;
    PrintStream p= null;

    try {
        out = new FileOutputStream(outfi);
        p = new PrintStream( out );
    }
    catch (Exception e) {
        System.err.println("Error writing to file.\n\t" + e.getMessage());
        System.exit(-1);
    }

    int cont_test = 0;
    int frr= 0;
    int far= 0;
    short resultCost= 0;
    int count_match= 1;
    int count_match_frr=0;
    int count_match_far=0;
    boolean frrFlag= false;

    for (int i= 1; i <= 110; i++) {
        p.println("*****Matching phase for fingerprint " + i + "
                  *****" +"\n");

        for (int j= 1; j <= 8; j++) {

            String file1= new String(dirBase + Integer.toString(i) + "_" +
                                     Integer.toString(j) + "_min_out.txt");

            for (int k= 1; k <= 110; k++) {
                if (i > k) continue; // no symmetrics

                if ((j != 1) && (i != k)) continue;
                //if ((i == k) && (j == 1)) continue;

                for (int z= 1; z <= 8; z++) {

```

```

if (j > z) continue; // no symmetric

if ((i != k ) && (z != 1)) continue;
if ((i == k) && (j == z)) continue;

System.out.println("test n. " + (count_match) + "\n");

String file2= new String(dirBase + Integer.toString(k) +
                        "_" + Integer.toString(z) +
                        "_min_out.txt");

FileReader fileReferenceInput= null;
FileReader fileCandidateInput= null;

// Opening first file
try {
    fileReferenceInput= new FileReader(file1);
}
catch (Exception e) {
    System.err.println("Error opening first file (" +
                        args[0] + "):");
    System.err.println("\t" + e.getMessage());
    System.exit(-2);
}

// Opening second file
try {
    fileCandidateInput= new FileReader(file2);
}
catch (Exception e) {
    System.err.println("Error opening second file (" +
                        args[1] + "):");
    System.err.println("\t" + e.getMessage());
    System.exit(-3);
}

// Creating biometric template
JMOCownerBioTemplate ob= null;
try {
    ob= BioBuilder.buildBioTemplate
        (BioBuilder.FINGERPRINT, (byte)3);
}
catch (BioException be) {
    System.err.println("Error in buildBioTemplate(): " +
                        be.getReason());
    System.exit(-12);
}

// Creating Parser
AlgParser minParser= new AlgParser();

//System.out.println(args[0] + " parsing phase.\n");

// Creating minutiae list for the first file
AlgMinutiaeList minutiaeReferenceList= null;
try {
    minutiaeReferenceList= new AlgMinutiaeList();
}
catch (MinutiaeListException mle) {
    System.err.println("Error creating first minutiae
                        list: " + mle.getMessage());
    System.exit(-4);
}

```

```

}

// Parsing first file
try {
    minParser.parse(fileReferenceInput,
                    minutiaeReferenceList);
}
catch (AlgParserException ape) {
    System.err.println("Error parsing first file (" +
                      args[0] + "):");
    System.err.println("\t" + ape.getMessage());
    try {
        fileReferenceInput.close();
    }
    catch (Exception e) {
        System.err.println("Error closing " + args[0] + "
                          " file.");
    }
    System.exit(-5);
}

// Closing first file
try {
    fileReferenceInput.close();
}
catch (Exception e) {
    System.err.println ("Error closing " + args[0] + "
                      file.");
    System.exit(-6);
}

// Converting first minutia list
Vector referenceCardList= null;
try {
    referenceCardList= Convert.convert
                      (minutiaeReferenceList);
}
catch (ConvertException ce) {
    System.err.println("Error converting enrollment
                      list: \n\t" + ce.getMessage());
    System.exit(-7);
}

//System.out.println(args[1] + " parsing phase.\n");

// Creating minutiae list for the second file
AlgMinutiaeList minutiaeCandidateList= null;
try {
    minutiaeCandidateList= new AlgMinutiaeList();
}
catch (MinutiaeListException mle) {
    System.err.println("Error creating second minutiae
                      list: " + mle.getMessage());
    System.exit(-8);
}

// Parsing second file
try {
    minParser.parse(fileCandidateInput,
                    minutiaeCandidateList);
}
catch (AlgParserException ape) {

```

```

        System.err.println("Error parsing second file (" +
                           args[1] + "):");
        System.err.println("\t" + ape.getMessage());
        try {
            fileCandidateInput.close();
        }
        catch (Exception e) {
            System.err.println("Error closing " + args[1] +
                               " file.");
        }
        System.exit(-9);
    }

    // Closing second file
    try {
        fileCandidateInput.close();
    }
    catch (Exception e) {
        System.err.println ("Error closing " + args[1] + "
                             file.");
        System.exit(-10);
    }

    // Converting second list
    Vector candidateCardList= null;
    try {
        candidateCardList= Convert.convert
                           (minutiaeCandidateList);
    }
    catch (ConvertException ce) {
        System.err.println("Error converting enrollment
                           list: \n\t" + ce.getMessage());
        System.exit(-11);
    }

    if ((candidateCardList.size() != 25) ||
        (referenceCardList.size() != 25)) {
        p.println("--");
        continue;
    }

    // Enrollment phase: INIT-UPDATE-DOFINAL

    try {
        ob.init(null, (short)(0), (short)(0));
    }
    catch (BioException be) {
        System.err.println("Error in init(): " +
                           be.getReason());
        System.exit(-13);
    }

    for (int a= 0; a < referenceCardList.size(); a++) {

        byte[] uByte= new byte[17];
        CardMinutia tempMin= (CardMinutia)
                             (referenceCardList.get(a));

        // Filling byte array
        short xPos= tempMin.getXPos();
        short yPos= tempMin.getYPos();
        uByte[0]= (byte) (a);
        uByte[1]= (byte) (xPos >> 8);
    }

```

```

        uByte[2]= (byte) (xPos & 0xFF);
        uByte[3]= (byte) (yPos >> 8);
        uByte[4]= (byte) (yPos & 0xFF);
        uByte[5]= tempMin.getReliability();
        uByte[6]= tempMin.getType();
        short[] nVector= tempMin.getNeighborsVector();
        //orderVector(nVector);
        for (int b= 0; b < JMOCMinutia.neighborsNumber;b++){
            uByte[7 + (b*2)]= (byte) (nVector[b] >> 8);
            uByte[8 + (b*2)]= (byte) (nVector[b] & 0xFF);
        }

        try {
            ob.update(uByte, (short) (0), (short)
                    (uByte.length));
        }
        catch (BioException be) {
            System.err.println("Error in update(): " +
                    be.getReason());
            System.exit(-14);
        }
    }

    try {
        ob.doFinal();
    }
    catch(BioException be) {
        System.err.println("Error in doFinal(): " +
                be.getReason());
        System.exit(-15);
    }
}

// Matching phase: INITMATCH-MATCH

try {
    ob.initMatch(null, (short) (0), (short) (0));
}
catch(BioException be) {
    System.err.println("Error in initMatch(): " +
            be.getReason());
    System.exit(-5);
}

// matchValue
short matchValue= 0;

count_match++;

for (int c= 0; c < candidateCardList.size(); c++) {

    byte[] uByte= new byte[16];
    CardMinutia tempMin= (CardMinutia)
            (candidateCardList.get(c));

    // Filling byte array
    short xPos= tempMin.getXPos();
    short yPos= tempMin.getYPos();
    uByte[0]= (byte) (xPos >> 8);
    uByte[1]= (byte) (xPos & 0xFF);
    uByte[2]= (byte) (yPos >> 8);
    uByte[3]= (byte) (yPos & 0xFF);

```

```

uByte[4]= tempMin.getReliability();
uByte[5]= tempMin.getType();
short[] nVector= tempMin.getNeighborsVector();
//orderVector(nVector);

for (int d= 0; d < JMOCMinutia.neighborsNumber;d++){
    uByte[6 + (d*2)]= (byte) (nVector[d] >> 8);
    uByte[7 + (d*2)]= (byte) (nVector[d] & 0xFF);
}

try {
    matchValue= ob.match(uByte, (short) (0), (short)
                        (uByte.length));
}
catch (BioException be) {
    System.err.println("Error in match(): " +
                      be.getReason());
    System.exit(-6);
}

}

if (i == k) {
    p.print("FRR ");
    frrFlag= true;
    count_match_frr++;
}
else {
    frrFlag= false;
    count_match_far++;
}

p.print("'" + i+"_"+j + " Vs " + k+"_"+z + "    match
        result= " + matchValue);

// Checking match
if ((matchValue < maxMatchCost)) {
    p.println(" *****OK***** \n");
    // Fingerprints are matched!

    // Checking FAR
    if (!frrFlag)
        far++;
}
else {
    p.println("\n");

    // Checking FRR
    if (frrFlag)
        frr++;
}
}

}

}

float farPCent= ((float)far * (float)100) / count_match_far;
float frrPCent= ((float)frr * (float)100) / count_match_frr

```



```

p.println("\n\n FAR = " + Float.toString(farPCent) + "    FRR = " +
        Float.toString(frrPCent) + " NFAR = " + far + " NFRR = " + frr
        + " maxMatchCost= " + maxMatchCost + " count_match_far= " +
        count_match_far + " count_match_frr= " + count_match_frr);

try {
    p.close();
}
catch (Exception e) {
    System.err.println("Error closing output file.\n\t" +
        e.getMessage());
}

}

}

```

D.5 Classe Main FarfrrCard.

```
/*
 * farfrr.java
 *
 * Created on 16 luglio 2003, 17.01
 *
 * Copyright 2003 Francesco Santini
 *
 * This file is part of FinMatch.
 *
 * FinMatch is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * FinMatch is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with FinMatch; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 221, Boston, MA 02222-2217 USA
 */

import java.io.*;
import java.lang.*;
import java.util.StringTokenizer;

/**
 * This class helps checking algorithm performance with threshold value
 * changing.
 *
 * @author Francesco Santini
 * @version 1.0
 */
public class FarfrrCard {

    /**
     * Minimum threshold to check performance.
     */
    private final static int mint = 0;

    /**
     * Maximum threshold to check performance.
     */
    private final static int maxt = 1000;

    /**
     * Creates a new instance of farfrr.
     */
    public FarfrrCard() {
    }

    /**
```

```

* Tool to check performance changing threshold value from minimum to
* maximum.
* Output text file stores these performances.
*
* @param args the command line arguments: Input text file containing
*         <code>FinalTest</code> output and second parameter is new
*         text file path.
*/
public static void main(String[] args) {

    // Controlling input parameters
    if (args.length != 2) {

        System.err.println("Error on input parameters.");
        System.err.println("\tUsage: java farfrr input_match_file
                           outputfile");
        System.exit(-1);

    }

    // Changing threshold (to minimum to maximum parameters)
    float maxMatchCost= 0;

    String infi= new String(args[0]);
    String outfi= new String(args[1]);

    // Creating log file
    FileOutputStream out= null;
    PrintStream p= null;

    try {
        out = new FileOutputStream(outfi);
        p = new PrintStream( out );
    }
    catch (Exception e) {
        System.err.println("Error writing to file.\n\t" + e.getMessage());
        System.exit(-1);
    }

    int cont_test = 0;
    int frr= 0;
    int far= 0;
    int count_match= 0;
    boolean frrFlag= false;

    for (int w = mint; w <= maxt; w++) {
        maxMatchCost = w;
        cont_test = 0;
        frr= 0;
        far= 0;
        frrFlag= false;

        BufferedReader in=null;
        try {
            in = new BufferedReader(new FileReader(infi));
        }
        catch (Exception e) {
            System.err.println("Error opening file.\n\t" + e.getMessage());
            System.exit(-1);
        }

        boolean exit= false;

```

```

for (int i= 1; i <= 110; i++) {
    //p.println("*****inizio match phase per impronta " +i+ "
    //*****" +"\n");

    if (exit)
        break;

    for (int j= 1; j <= 8; j++) {

        if (exit)
            break;
        //String file1= new String(dirBase + Integer.toString(i) +
        // "_" + Integer.toString(j) + "_min_out.txt");

        for (int k= 1; k <= 110; k++) {

            if (exit)
                break;

            if (i > k) continue; // salto i simmetrici

            if ((j != 1) && (i != k)) continue;
            //if ((i == k) && (j == 1)) continue;


            for (int z= 1; z <= 8; z++) {

                if (exit)
                    break;

                if (j > z) continue; // salto i simmetrici

                if ((i != k ) && (z != 1)) continue;
                if ((i == k) && (j == z)) continue;


                //System.out.println("test n. " + (++cont_test)
                //+" \n");

                int pairs= 0;
                float mean= 0;
                boolean trovato = false;
                boolean jump= false;

                while (!trovato) {

                    String aaa=null;
                    try {
                        aaa = in.readLine();
                    }
                    catch (Exception me) {
                        System.err.println("Error readline");
                    }

                    if (aaa == null) {
                        exit= true;
                        break;
                    }
                }
            }
        }
    }
}

```

```

StringTokenizer st=null;
try {
    st = new StringTokenizer(aaa);
}
catch (Exception e) {
    System.err.println("Error StringTokenizer" +
        e.getMessage());
    System.exit(-1);
}
while (st.hasMoreTokens()) {

    String bada = st.nextToken();

    if ( bada.equals("result=") ) {

        bada = st.nextToken();

        mean = (new Float(bada)).floatValue();
        trovato = true;
    }
    else
        if (bada.equals("--"))
            jump= true;

}

}

if (jump)
    continue;

if (i == k) {
    //p.print("FRR ");
    frrFlag= true;
    //count_match_frr++;
}
else {
    frrFlag= false;
    //count_match_far++;
}

//p.print("" + i+"_"+j + " Vs " + k+"_"+z + "
//match result= " + Float.toString(mean) + " - Pairs
//number= " + pairs);
//count_match++;
//p.print(count_match);
if ((mean < maxMatchCost)) {
    //p.println(" *****OK*****");
    //println);
    if (!frrFlag)
        far++;

}

else {
    //p.println("\n");
    if (frrFlag)
        frr++;
}

```

```

        }
    }

    }

    }
    float farPCent= ((float)far * (float)100) / (float)5995;
    float frrPCent= ((float)frr * (float)100) / (float)2924;
    p.println("FAR = " + Float.toString(farPCent) + "    FRR = " +
        Float.toString(frrPCent)
    + " NFAR = " + far + " NFRR = " + frr
    + " maxMatchCost= " + maxMatchCost);

} // end for far frr

try {
    p.close();
}
catch (Exception e) {
    System.err.println("Error closing output file.\n\t" +
        e.getMessage());
}

} // end main

}

```