UNIVERSITÀ DEGLI STUDI DI PADOVA
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Realizzazione su piattaforma ad agenti mobili Jade di un sistema di autenticazione biometrica

Relatore: *Ch.mo Michele Moro.....*

Laureando: *Marco Rosada 578129-IF*

Sessione Laurea Febbraio

Anno Accademico 2010 / 2011

Sommario

La diffusione su scala mondiale della rete Internet ha portato ad un aumento dei requisiti minimi riguardanti scalabilità, tolleranza ai guasti e sicurezza richiesti dalle moderne applicazioni operanti in rete. In questo scenario il classico modello client-server ha evidenziato i propri limiti.

Una soluzione a questo problema è rappresentata dal paradigma ad *agenti mobili*, entità software capaci di migrare, attraverso una rete, da un calcolatore all'altro e di eseguire su di essi svariati tipi di compiti.

Un altro fenomeno di massa registratosi negli ultimi anni è senza dubbio l'esplosione degli smartphone, dispositivi mobili con performance sempre maggiori e capaci di eseguire applicazioni molto simili a quelli per ambienti desktop.

Da queste osservazioni nasce l'idea di sviluppare un sistema di autenticazione biometrica che, da un lato, sfrutti le nuove tecnologie e, dall'altro, garantisca sicurezza ed efficienza.

A tale scopo, il sistema realizzato utilizza le impronte digitali come metodo per effettuare l'autenticazione dei propri utenti. Per incrementarne ulteriormente la sicurezza, gli agenti che lo popolano utilizzano algoritmi crittografici in modo da proteggere le informazioni da essi trasportate. Quest'ultimo aspetto risulta essere fondamentale in quanto esse sono costituite da dati strettamente personali e quindi soggetti alle leggi sulla *privacy*.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Nel corso dell'ultimo decennio la diffusione della rete Internet è stato uno dei maggiori fenomeni tecnologici, tanto da aver inciso profondamente sullo stile di vita e le abitudini delle persone. L'aumento del numero di servizi disponibili sulla rete ha avuto come diretta conseguenza uno sviluppo consistente dei sistemi distribuiti. Questo ha reso necessaria un'intensa attività di ricerca in ambito informatico per far fronte alle problematiche che tale espansione ha causato, come la crescente necessità di dinamicità, scalabilità, tolleranza ai guasti, adattabilità e sicurezza.

Per rispondere a questi quesiti è stato introdotto un nuovo paradigma di programmazione, detto ad agenti mobili, migliore rispetto ai classici modelli client-server per quanto riguarda il rispetto dei nuovi vincoli tecnologici imposti. Va comunque sottolineato che in questo campo il paradigma principale è ancora quello basato sull'architettura client-server.

Il paradigma ad agenti mobili è basato sulla nozione di mobilità e quella di agente. In sintesi, esso prevede di poter sviluppare un programma, detto agente, che sia in grado di spostarsi tra i vari nodi che compongono una rete di calcolatori attraverso vari domini di protezione. Una volta arrivato a destinazione esso è in grado di decidere quali azioni compiere e su quali nodi spostarsi successivamente, in modo sostanzialmente indipendente dal calcolatore su cui è eseguito.

I vantaggi di questo approccio sono molteplici: fra i più importanti citiamo la tolleranza ai guasti (l'agente prima di muoversi da un nodo all'altro può decidere di portare con sé dati prelevati dal nodo in cui si trova, in modo da averli disponibili nel calcolatore di destinazione, senza quindi chiedere una connessione sempre attiva tra i due nodi), l'efficienza (ripetute interazioni con un server remoto possono essere soppresse spedendo il processo direttamente al client in modo che vi interagisca localmente), semplicità e flessibilità (il mantenimento di una rete può essere molto più semplice quando le applicazioni sono su un server, ed i client, autonomamente, le installano solo quando necessario) ed infine l'occupazione di spazio (scaricare il

codice quando necessario, permette di ridurre la duplicazione del codice sui vari siti).

Risulta evidente come sia fondamentale modellare correttamente le interazioni cui tali agenti possono venire sottoposti, in modo da renderle affidabili e sicure. L'ottenimento e la gestione di uno specifico livello di sicurezza, tuttavia, sono un compito tutt'altro che banale, soprattutto se si considera l'alta dinamicità di interazione cui gli agenti sono sottoposti.

Un'altra possibilità offerta da questo paradigma di programmazione è quella di eseguire agenti portatili anche su dispositivi mobili, dotati in genere di risorse limitate, sia in termini di potenza di calcolo che di disponibilità di memoria. Tenendo poi in considerazione il grandissimo sviluppo tecnologico nel campo dei dispositivi mobili registratosi negli ultimi anni, appare evidente come l'integrazione tra l'utente finale (ad esempio il possessore del telefonino) ed il sistema centrale risulti estremamente forte e di facile attuazione.

Nasce quindi l'idea di sviluppare un sistema di autenticazione ad agenti mobili che integri le nuove tecnologie nell'ambito dei dispositivi mobili e le conoscenze disponibili nel campo del matching biometrico, ottenendo una struttura solida sulla quale potranno essere sviluppate in futuro numerose applicazioni. Nella realizzazione di questo modello è di fondamentale importanza garantire la sicurezza dei dati e delle comunicazioni, in modo da evitare che entità malevoli possano introdursi all'interno del sistema con lo scopo di comprometterne il corretto funzionamento.

1.1 Obiettivi della tesi e software sviluppato

L'obbiettivo di questa tesi è la progettazione e lo sviluppo di un sistema sicuro per l'autenticazione biometrica sfruttando la potenza offerta dal paradigma ad agenti mobili, ottenendo uno strumento completo dal quale partire per sviluppare applicazioni successive.

Le caratteristiche peculiari del sistema realizzato sono quindi l'utilizzo di una piattaforma ad agenti mobili in alternativa al classico paradigma client-server, l'autenticazione degli utenti tramite parametro biometrico, nel caso di questa tesi si è optato per l'utilizzo delle impronte digitali, e l'impiego di algoritmi crittografici per garantire la sicurezza delle informazioni trasmesse attraverso la rete in cui opera il sistema.

Di seguito una breve descrizione dei programmi proposti (per una trattazione più specifica si invita il lettore a prendere visione del capitolo 5):

- Enrollment Tool : software per la registrazione dei dati personali dell'utente che in futuro richiederà l'autenticazione.

- Sistema di autenticazione biometrico Android TemplateOnPhone, in cui il template dell'utente è salvato sullo smartphone ed il processo di matching biometrico avviene sul server .

In parallelo a tali programmi sono state implementate due librerie:

- Utility: funzioni di varia utilità comprendenti tra le altre cose la gestione dei file, la creazione di connessioni SSL e la gestione della cifratura simmetrica/asimmetrica utilizzata in modo massiccio all'interno degli altri programmi.
- Utility Android : funzioni comprendenti la gestione dei file e la crittografia specifiche per smartphone con sistema operativo Android.

1.2 Prerequisiti per la lettura del testo

Non sono necessarie particolari conoscenze preliminari, né per quanto riguarda il campo informatico, né per quanto riguarda il campo della biometria per una comprensione efficace di questa tesi.

Si presuppone, tuttavia, che il lettore abbia una certa dimestichezza con le nozioni di base dell'informatica. In particolare di:

- buona conoscenza del linguaggio di programmazione Java in quanto è il linguaggio con cui è stata scritta l'applicazione;
- alcune nozioni tipiche dei sistemi distribuiti: paradigma client-server, crittografia, ecc. . . ;

1.3 Prerequisiti per l'utilizzo del software

I requisiti necessari per l'esecuzione del software sono:

- sistema Linux (il software è stato sviluppato su Ubuntu 10.04 LTS 64-bit)[1]
- Java Development Kit 1.62 (o superiore)[2][3]
- libreria JAI - Java Advanced Imaging3[4]
- smartphone Android versione 2.1 (o superiore)[5]
- server MySQL 5.05 (o superiore)[6]
- scanner biometrico modello Biometrika Fx2000 comprensivo di driver versione 1.07 (o superiore)[7]

Nel caso in cui non si sia in possesso di un telefono basato su sistema operativo Android, è possibile testare il software sull'emulatore fornito da Android SDK(*software developement kit*)[8]. Per la descrizione dei passi necessari al funzionamento dei programmi si rimanda il lettore interessato al capitolo 6.

Capitolo 2

Piattaforme ad agenti mobili

2.1 Introduzione agli agenti software

Per la realizzazione del software oggetto di questa tesi si è utilizzato una piattaforma basata sugli agenti mobili: questi ultimi, come già anticipato, costituiscono un campo dell'informatica relativamente recente su cui è stata posta l'attenzione della comunità scientifica internazionale. Ne verranno presentate ora le nozioni e la terminologia di base, in modo da poter permettere al lettore la comprensione dei contenuti che verranno esposti successivamente.

Il concetto di *agente software* è nato all'inizio degli anni '80, come punto d'incontro tra il campo di ricerca dell'intelligenza artificiale e quello dell'informatica; il suo scopo era fornire una soluzione che permettesse di risolvere, in maniera automatizzata, i problemi che l'espansione delle reti di calcolatori (Internet in primis) iniziava a porre, di cui si accennato nel precedente capitolo.

Attualmente non esiste ancora una definizione universalmente accettata di cosa si intenda effettivamente con il termine "agente software". Volendone comunque dare una definizione intuitiva, esso può essere identificato come un software dotato di un certo grado di autonomia, inteso come capacità di prendere decisioni in modo indipendente e sulla base delle condizioni ambientali circostanti; queste ultime possono includere ad esempio: informazioni ricevute o inviate verso altri agenti, output ricevuti da altri processi in esecuzione concorrente, risultati di query effettuate su un database, etc...

Proprio perché una delle caratteristiche peculiari di un agente software è costituita dalla sua capacità di prendere autonomamente decisioni, per meglio caratterizzarne le abilità, le potenzialità e quindi i comportamenti che esso potrebbe esibire durante la sua esecuzione, la letteratura scientifica ha identificato alcune proprietà utili a fornirne una più accurata classificazione.

Alcune tra le più importanti sono:

- *Autonomia*: caratterizza il grado di libertà che possiede l'agente nel prendere decisioni riguardo alle azioni che eseguire. Essa costituisce anche un sinonimo di *intelligenza*;
- *Reattività*: indica la capacità dell'agente di percepire l'ambiente ad esso circostante, e di conseguenza di reagire ai cambiamenti che si verificano in quest'ultimo;
- *Proattività*: un agente proattivo in grado non solo di reagire agli stimoli dell'ambiente circostante, ma anche di prendere autonomamente la decisione di intraprendere una determinata azione: esso esibisce cioè dei comportamenti *goal-oriented*;
- *Abilità sociali*: indicano la capacità dell'agente di interagire con altri agenti, ed eventualmente anche con esseri umani; ovviamente questo deve avvenire in base a dei linguaggi e delle semantiche concordate precedentemente;
- *Persistenza*: nel caso l'esecuzione dell'agente venisse sospesa, esso è in grado di salvare un insieme di informazioni nell'ambiente in cui è situato in modo da poter riprendere il proprio flusso computazionale in un momento successivo;
- *Mobilità*: nel caso il calcolatore su cui l'agente in esecuzione sia connesso ad una rete, quest'ultimo possiede la capacità di muoversi tra i nodi che la compongono;
- *Adattività*: un agente adattivo in grado di apprendere sia dalle proprie azioni che dall'ambiente circostante, migliorando conseguentemente l'efficienza delle proprie azioni nel tempo;
- *Fidatezza*: un agente si definisce fidato se fornisce la garanzia agli utenti con cui interagisce che non comunicherà mai informazioni riservate a terze parti senza esserne stato autorizzato precedentemente;
- *Benevolenza*: l'agente proverà sempre ad eseguire ciò che gli è stato richiesto, controllando comunque che gli obbiettivi ad esso assegnati non siano in contrasto tra di loro;
- *Cooperazione*: l'agente esibisce logiche di collaborazione con altri agenti eventualmente presenti nel proprio ambiente. In questo modo esso può portare a termine i propri compiti in meno tempo oppure utilizzando meno risorse, proprio grazie alla collaborazione con gli altri agenti. Esiste anche il caso opposto, in cui l'agente esibisce comportamenti in cui entra in competizione con gli altri agenti per l'utilizzo delle risorse presenti nell'ambiente: in questo caso si parla di *competitività*.

Con riferimento all'elenco appena esposto, un agente viene classificato come *debole* se esibisce proprietà che sono tra le prime sei; in caso contrario, esso viene definito *forte*. Si procederà ora con un approfondimento di alcune proprietà degli agenti risultate fondamentali per lo sviluppo del software oggetto di questa tesi.

2.2 Mobilità

2.2.1 Introduzione

Gli agenti utilizzati per la realizzazione del sistema oggetto di questa tesi sono mobili, ovvero, come si evince dal sottocapitolo precedente, agenti che possiedono la capacità di spostarsi da un nodo ad un altro di una rete di calcolatori.

È quindi possibile apprezzare immediatamente come, da un punto di vista architetturale, essi possano dar vita ad un paradigma totalmente diverso rispetto ad un sistema basato su una classica architettura client-server. Nel primo caso, infatti, si hanno delle entità di pari livello, in grado di spostarsi all'interno della rete in cui operano, e che possono prendere decisioni dinamicamente in base alle condizioni dell'ambiente in cui si trovano. Nel secondo caso, invece, le entità sono in relazione gerarchica tra di loro: generalmente, infatti, i client necessitano dei server per poter ultimare i loro compiti, mentre non è vero il contrario; essi inoltre occupano posizioni fisse all'interno della rete (salvo cambiamenti nella topologia della rete stessa) e le decisioni che essi prendono sono stabilite in base a logiche predefinite.

Poiché la mobilità è risultata di importanza fondamentale per la realizzazione del sistema che verrà presentato in seguito, si ritiene opportuno proporne qui un approfondimento, in modo da poter consentire al lettore una migliore comprensione delle nozioni contenute nei capitoli successivi.

Da un punto di vista strettamente operativo, un agente mobile è caratterizzato da due elementi fondamentali: il *programma*, inteso come serie di istruzioni interpretabili da un calcolatore elettronico, che viene generalmente memorizzato in un dispositivo secondario; quando viene eseguito dal sistema operativo, esso dà luogo nella memoria volatile del calcolatore ad un *processo*, caratterizzato da uno *stato*, inteso come l'insieme di informazioni su cui sta operando.

La capacità dell'agente di trasferire il proprio stato con sé, durante il movimento da un nodo ad un altro della rete, permette di operare una classificazione riguardo al tipo di mobilità di cui l'agente stesso dotato. Si distinguono in particolare tre tipi di mobilità:

- *Weak Mobility* (mobilità debole): un agente dotato di mobilità debole è in grado di portare con sé solo il codice del suo programma, e non anche il suo stato;

- *Not-so-weak Mobility* (mobilità semi-forte): caratterizza la capacità dell'agente di portare una parte del suo stato con sé;
- *Strong Mobility* (mobilità forte): a quest'ultimo caso appartiene ogni agente in grado di portare con sé tutte le informazioni riguardanti il suo stato;

La distinzione tra i vari tipi di mobilità è fondamentale in quanto, una volta giunto a destinazione, un agente si limita a eseguire di nuovo il suo codice dall'inizio: se invece è stato possibile portare, assieme ad esso, anche il suo stato (o una sua parte), allora l'agente potrà ricordarsi le azioni compiute in precedenza; chiaramente quest'ultima opzione permette all'agente, in generale, di svolgere operazioni più complesse e articolate, e quindi anche di poter implementare al suo interno una maggiore intelligenza.

2.2.2 Mobilità e linguaggio Java

Il software oggetto di questa tesi è stato sviluppato in linguaggio Java, che permette di implementare nativamente una mobilità di tipo *not-so-weak*. Al fine di permettere al lettore una migliore comprensione del contenuto dei capitoli successivi, si ritiene opportuno a questo punto fornire un breve richiamo su come la mobilità possa essere implementata negli agenti mobili tramite Java.

Per poter eseguire i programmi che gli vengono passati in input Java si basa su una macchina virtuale, la cosiddetta JVM (*Java Virtual Machine*). Essa non è tuttavia in grado di interpretare la sintassi del linguaggio Java; per poter eseguire un programma Java esso deve essere precedentemente compilato, in modo da ottenere un file con estensione `.class`, composto dal cosiddetto *bytecode*, ovvero un insieme di istruzioni comprensibili dalla JVM. A questo punto la macchina virtuale in grado di tradurle a loro volta in comandi elaborabili dalla CPU del calcolatore su cui essa è in esecuzione.

Ai fini di questa presentazione si vuole ricordare al lettore che, quando in esecuzione, la JVM è caratterizzata dai seguenti tre elementi fondamentali:

- il *program counter*: esso contiene la prossima istruzione del bytecode che verrà eseguita dalla JVM; ne esiste uno per ogni thread in esecuzione nella macchina virtuale.
- un *heap*: esso è costituito da un'area di memoria comune a tutti i thread, ed è soggetta a *garbage collection*. Essa contiene le istanze degli oggetti creati dai vari thread durante l'esecuzione della JVM stessa.
- una *method area*: anche quest'ultima è condivisa tra tutti i thread, ha come compito quello di memorizzare le varie strutture associate ad ogni classe, come ad esempio il codice dei metodi.

A questo punto è utile ricordare come uno dei paradigmi su cui si fonda il linguaggio Java sia quello della sicurezza: in particolare, sono permessi all'interno di un programma solo riferimenti simbolici alla memoria (al contrario di quello che avviene in altri linguaggi come ad esempio il C). Tali riferimenti verranno poi tradotti in indirizzi fisici in fase di esecuzione del programma stesso.

Questa premessa è fondamentale per comprendere come sia organizzato il meccanismo, chiamato *serializzazione* e fornito nativamente dal linguaggio Java[1], utilizzato per l'implementazione della mobilità negli agenti.

Esso permette la copia dell'heap in un generico flusso di uscita, come può essere un file, un *byte stream* oppure un socket TCP/IP; tale operazione è resa fattibile da un'apposita classe chiamata `ObjectOutputStream`. Il processo inverso, tramite cui si può ottenere l'heap di un oggetto dal precedente output, è possibile attraverso l'uso di una classe simmetrica alla precedente, chiamata `ObjectInputStream`.

La possibilità, per un oggetto, di essere o no serializzato dipende dalla definizione dell'oggetto stesso, in particolare dal fatto che esso implementi una tra le due interfacce `Serializable` oppure `Externalizable`. La differenza tra le due è che la prima non richiede la definizione aggiuntiva di alcun metodo, mentre la seconda permette di ottenere maggiori prestazioni in termini di tempo di calcolo, a fronte di una più alta complessità di implementazione.

Vi sono tuttavia alcuni oggetti che non possono essere serializzati: tipicamente, si tratta di oggetti che operano su un insieme di variabili fortemente dipendenti dall'ambiente all'interno del quale sono in esecuzione, come ad esempio quelli di tipo `Thread`.

Da quanto si evince dalla descrizione precedente, per dotare un agente realizzato in Java di mobilità *not-so-weak*, è sufficiente permettere, oltre che la copia del bytecode dell'agente stesso, anche del suo heap, tramite l'utilizzo delle classi descritte prima.

2.3 Sicurezza

Come già anticipato, poiché il software realizzato si basa su agenti mobili, uno degli aspetti chiave da tenere in considerazione è quello della sicurezza. Il paradigma ad agenti mobili possiede infatti un carattere fortemente distribuito, ed è quindi naturale adottare tutte le contromisure necessarie per evitare che utenti non autorizzati possano intercettare gli agenti per carpire informazioni riservate al loro interno oppure modificarne il codice per alterarne il comportamento. Inoltre essa risulta particolarmente importante per il tipo di informazioni strettamente personali su cui gli agenti operano nel sistema realizzato, come ad esempio dati anagrafici, password, ...

[1] In particolare, dalla versione 1.1 del JDK in poi.

È quindi facile comprendere come, per garantire la sicurezza e l'integrità del sistema, oltre che per soddisfare le normative esistenti (come ad esempio quelle relative alla *privacy*), sia necessario prestare la massima attenzione alla tematica della sicurezza.

Parlando a livello generale, le minacce si possono classificare in tre categorie:

- intercettazione di informazioni;
- alterazione di informazioni;
- interruzione di servizio (*Denial of Service*);

Nel primo caso, un individuo non autorizzato riesce ad ottenere l'accesso ad informazioni o ad un servizio riservati ad altri utenti. Nel secondo caso, una terza parte non autorizzata modifica alcuni dati o servizi in modo che non corrispondano più alle specifiche originali. Infine, nel terzo caso, si vuole rendere inutilizzabili dati o servizi in maniera dolosa, in modo che non siano più disponibili per gli utenti autorizzati che accedono al sistema.

Entrando più nello specifico nell'ambito degli agenti mobili, si possono identificare le possibili minacce come:

- Agent-to-Platform, quando un agente si avvantaggia delle debolezze di una piattaforma per poter lanciare degli attacchi;
- Agent-to-Agent, nel caso un agente sfrutti le falle presenti in altri agenti nel sistema;
- Platform-to-Agent, nell'eventualità che sia la piattaforma stessa ad attaccare gli agenti;

Per quanto concerne il primo caso, tra le minacce che è opportuno prendere in considerazione si citano:

- quella comunemente chiamata *masquerading*, nella quale un agente non autorizzato dichiara l'identità di un altro agente, solitamente autorizzato, al fine di poter accedere ad informazioni riservate. Oltre alla fuoriuscita di informazioni riservate, in questo caso si può ledere anche la fidatezza dell'agente autorizzato di cui si ruba l'identità;
- il *denial of service*, che possono essere causati da errori di programmazione degli agenti stessi oppure, volutamente, da questi ultimi sfruttando eventuali falle note nella sicurezza del sistema. In ogni caso, il risultato è l'occupazione di risorse della piattaforma, al fine di renderne inutilizzabili i servizi da essa offerti;

- eventuali *accessi non autorizzati*, tramite cui un agente può accedere a risorse e servizi per i quali non possiede le autorizzazioni. Questo tipo di minaccia può essere prevenuta adottando adeguate politiche per la sicurezza, che consentano di verificare in maniera sicura le credenziali dell'agente;

Nel secondo caso, è importante tenere conto dei seguenti tipi di attacchi:

- *masquerading*, il quale è possibile che si verifichi nel momento in cui due agenti vogliano comunicare; uno dei due potrebbe essere infatti un'entità non autorizzata che, fingendosi un agente autorizzato, ha interessi a carpire informazioni riservate;

- *denial of service*, tramite cui agenti non autorizzati possono intasare le code dei messaggi di altri agenti (*flooding*), togliendo a questi ultimi la capacità di ricevere messaggi dall'esterno e occupandone tutte le risorse computazionali disponibili. È quindi necessario prevedere meccanismi di comunicazione tra agenti che impediscano tale fenomeno;

- *ripudio*, che può avvenire quando un agente, impegnato in una transazione, al termine di quest'ultima neghi che essa sia effettivamente avvenuta. Questo tipo di comportamento può portare alla generazione di dispute che, in generale, potrebbero non essere facilmente risolvibili;

- *accesso non autorizzato*: in questo caso un agente privo di autorizzazione potrebbe modificare i dati in possesso di un altro agente, oppure invocarne i metodi pubblici senza averne un'effettiva necessità. Questo tipo di attacco può avere come conseguenza la modifica del comportamento predefinito di un agente, e può quindi provocare nel sistema effetti imprevedibili;

Infine, nel terzo caso, è necessario prestare attenzione a minacce come:

- *masquerading*: in questo caso una piattaforma può ingannare un agente, facendogli credere di essere un'altra piattaforma che sarebbe invece autorizzata. Questo tipo di attacco può indurre l'agente a compiere determinate operazioni tramite la piattaforma non autorizzata, che diventa quindi capace di carpire informazioni riservate;

- *eavesdropping*, termine con il quale si indica, genericamente, il tentativo di accedere ad informazioni riservate. Nel caso di un attacco Platform-to-Agent essa diventa però particolarmente insidiosa: questo perché, quando un agente utilizza una determinata piattaforma, espone ad essa anche i dati che possiede, oltre al suo codice e al suo stato. Risulta quindi fondamentale prevedere meccanismi per oscurare tali dati alla piattaforma;

- *denial of service*, tramite cui una piattaforma non autorizzata potrebbe negare ad un agente i servizi che esso si aspetterebbe di ricevere da essa. Ad esempio, la piattaforma potrebbe rifiutarsi di eseguire determinate operazioni (che invece dovrebbe eseguire), oppure introdurre ritardi inaccettabili, con la conseguenza di alterare il normale funzionamento dell'agente e quindi compromettere la solidità dell'intero sistema.

Come si può quindi dedurre dall'elenco sopra riportato, è necessario analizzare e progettare le contromisure per prevenire tali tipi di attacchi con la massima attenzione. Tra i metodi più utilizzati, in particolare, si possono citare:

- *l'autenticazione*: essa permette di stabilire non solo i soggetti autorizzati ad accedere a determinate risorse, ma anche delle politiche per la sicurezza, in modo da poter definire delle procedure e delle metodologie per la definizione dei diritti di accesso ai dati e servizi disponibili nel sistema;

- *la cifratura*: consente, tramite algoritmi crittografici [2], di rendere determinati dati non comprensibili a soggetti privi della necessaria autorizzazione;

- *l'hash*: permette di verificare in modo immediato, e con una ragionevole certezza, se un messaggio è stato modificato da una terza parte oppure no. Esso consiste nel calcolo di una particolare funzione che ha come input proprio il messaggio stesso;

- *la firma digitale*: il suo scopo è quello di certificare l'identità del mittente e quindi di evitare il problema del ripudio;

Tali tecniche possono ovviamente essere utilizzate singolarmente o in combinazione, in modo da contrastare più minacce contemporaneamente.

Quale sia la strategia da adottare rimane ovviamente una scelta finale del progettista, che dovrà prendere tale decisione in funzione dei requisiti del sistema da realizzare.

2.4 Lo standard FIPA

Le piattaforme ad agenti mobili hanno conosciuto, negli ultimi anni, un crescente consenso ed utilizzo nei più svariati ambiti applicativi: sono quindi disponibili, attualmente, numerose piattaforme, sia commerciali che gratuite e open-source.

Parallelamente alla loro crescita, ha acquisito importanza il problema dell'interoperabilità tra di esse e tra gli agenti che le popolano. Nei primi anni di sviluppo

[2] Come si vedrà in seguito, essi possono essere sia a chiave simmetrica che asimmetrica.

degli agenti software, infatti, ogni piattaforma implementava dei linguaggi e delle semantiche per la comunicazione e la collaborazione tra i propri agenti specifici ad essa solamente, e quindi incompatibili con le altre piattaforme.

Sono state quindi elaborate, dagli sviluppatori del settore, alcune proposte di standard per la comunicazione tra agenti mobili di diverse piattaforme: tra le prime proposte, si segnalano in questa sede KQML[10] e MASIF[11]. Entrambi non ebbero però il successo sperato: il primo fu il frutto di un lavoro unicamente accademico e rimase confinato in quell'ambito, mentre il secondo aveva il solo scopo di stabilire un livello di interoperabilità di base tra agenti, che si rivelò in seguito insufficiente per le applicazioni sempre più complesse che venivano sviluppate.

Lo standard *de facto* attuale per l'interoperabilità tra agenti è invece FIPA (*Foundation for Intelligent and Physical Agents*)[12]. Esso è stato elaborato dall'omonimo consorzio, con sede in Svizzera, di cui fanno parte numerose multinazionali che operano nel campo dell'informatica e dell'elettronica [3], e dal 2005 è entrato a far parte dell'IEEE come *Standards Committee*. Come FIPA stessa dichiara, lo scopo di tale standard è solo quello di definire un'interfaccia comune per i diversi oggetti che popolano l'ambiente in cui gli agenti software si troveranno operare, in modo da permettere la comunicazione tra agenti (mobili e non) di diverse piattaforme, ed anche tra agenti software e altri tipi di tecnologie non basate sugli agenti.

In particolare, lo standard FIPA coinvolge i seguenti cinque ambiti:

- *Applications*: esso è costituito da un insieme di esempi di applicazioni, per ognuna delle quali FIPA propone un insieme minimo di servizi che l'agente dovrebbe essere in grado di fornire;

- *Abstract Architecture*: definisce, a livello astratto, gli elementi essenziali per la realizzazione di una piattaforma per agenti software, le relazioni tra di essi ed indica alcune linee guida per la loro implementazione;

- *Agent Communication*: riguarda un insieme di specifiche per la comunicazione tra agenti. Riguardo a questa tematica FIPA ha elaborato un linguaggio, chiamato ACL (*Agent Communication Language*), costituito da un formato comune per lo scambio di informazioni tra agenti. Esso inoltre prevede numerosi protocolli e schemi di interazioni;

- *Agent Management*: è formato da una serie di linee guida per la gestione e il controllo degli agenti e del loro comportamento, sia quando essi si trovano all'interno di una piattaforma che durante il transito tra una piattaforma e l'altra;

[3] Come, ad esempio, Siemens e Toshiba.

- *Agent Message Transport*: definisce delle modalità standard per il trasferimento e la rappresentazione delle informazioni attraverso reti disomogenee per quanto riguarda i protocolli di trasporto;

Per poter aderire allo standard FIPA una piattaforma ad agenti software deve quindi implementare le specifiche contenute nei cinque ambiti sopra elencati.

2.5 Piattaforme disponibili

In seguito all'aumento di interesse nei confronti del paradigma di programmazione ad agenti mobili, sono state sviluppate diverse piattaforme, open-source e non. Le principali sono:

- *Aglets*: è una piattaforma per agenti mobili sviluppata in linguaggio Java dai laboratori di ricerca IBM, resa open-souce al momento dell'abbandono del progetto da parte di IBM stessa. È importante sottolineare che questa piattaforma non soddisfa le specifiche dello standard FIPA (vedi [13]).

- *Cougaar*: sviluppata dalla DARPA (Defence Advanced Research Projects Agency), è una piattaforma ad agenti mobili dotata di un'ottima documentazione, numerosi add-on e di una comunità open-souce molto attiva. Purtroppo, anche Cougaar non aderisce allo standard FIPA (vedi [14]).

- *Agent Factory*: piattaforma per agenti mobili realizzata interamente in Java: è stata sviluppata da alcuni ricercatori dell'Università di Dublino nell'ambito di ricerca sulle possibili applicazioni degli agenti mobili in settori come il mobile computing, la robotica e reti intelligenti di sensori. Anche se in grado di realizzare applicazioni ad agenti mobili complesse, secondo le personali esigenze di un utente, la caratteristica principale della piattaforma è quella di poterli creare rapidamente: essa infatti è fornita di numerosi modelli predefiniti di reti ad agenti mobili che possono poi essere composti per crearne di nuovi e di maggiore complessità. È resa accessibile al pubblico tramite licenza LGPL e possiede numerose altre caratteristiche, tra cui l'adesione allo standard FIPA2000, la possibilità di essere eseguita anche su piccoli terminali wireless e la presenza di numerosi progetti per lo sviluppo di add-on da poter implementare nella piattaforma (vedi [15]).

- *Semoa*: è una piattaforma per agenti mobili realizzata completamente in Java da un'azienda tedesca (la Fraunhofer-IGD): è attualmente disponibile per il download pubblico sotto licenza LGPL. Ha fatto della sicurezza degli agenti e delle loro comunicazioni il suo punto di forza: sono infatti presenti meccanismi avanzati di crittografia e di gestione di certificati per la rilevazione

di malicious host, così come per la difesa di attacchi di tipo denial-of-service, ed anche per garantire l'integrità delle comunicazioni tra agenti. Un'altra caratteristica molto interessante di questa piattaforma è quella della sua interoperabilità con gli agenti mobili delle piattaforme Jade e Aglets. Semoa supporta infine anche il linguaggio di comunicazione per agenti definito da FIPA, ovvero ACL (vedi [16]).

- *Jade*:è la piattaforma utilizzata per lo sviluppo del software oggetto di questa tesi, aderisce pienamente allo standard FIPA: la sua architettura verrà presentata nel proseguo del capitolo (vedi [17]).

2.5.1 Confronto tra le piattaforme disponibili

Come è già stato sottolineato in precedenza, si è ritenuto di fondamentale importanza che la piattaforma scelta per sviluppare il progetto dovesse attenersi allo standard FIPA.Inoltre, quando si è dovuto decidere quale piattaforma adottare, si è preferito privilegiare le piattaforme attorno alle quali vi è una forte comunità di sviluppatori. Seguendo queste linee guida, la scelta è ricaduta sulla piattaforma Jade, scelta rinforzata dal supporto che Jade fornisce per i dispositivi mobili. Altre opzioni sono state scartate per motivi non strettamente legati alla loro qualità, ma per fattori quali la mancanza di sviluppo e aggiornamento (Semoa).

2.6 JADE

2.6.1 Introduzione

JADE (*Java Agent DEvelopment Framework*) è una piattaforma ad agenti mobili sviluppata dai laboratori di ricerca Telecom situati a Torino.

Il progetto, che ha avuto inizio alla fine del 1998, rese disponibile la prima versione per il download al pubblico all'inizio del 2000. Poco dopo, il team di sviluppo decise di aderire allo standard FIPA e rese il software open-source, rilasciandolo sotto licenza LGPL (*Lesser General Public License*). Da quel momento si è aggregata attorno al progetto una sempre più numerosa comunità di sviluppatori, che ha contribuito allo sviluppo di JADE e creato numerose estensioni, allo scopo di renderlo adatto all'utilizzo nelle più svariate applicazioni. Il rilascio sotto licenza LGPL non ha comunque comportato un abbandono del progetto da parte dei laboratori Telecom, che anzi ancora oggi contribuiscono attivamente al suo sviluppo.

Come risultato dell'interesse di alcune aziende nel progetto, nel 2003 è stata fondata la *JADE Board*, un'organizzazione no-profit che si pone come scopo la promozione dell'utilizzo di JADE come middleware per lo sviluppo di applicazioni

basate su agenti software. È da segnalare che tali aziende hanno contribuito attivamente allo sviluppo della piattaforma, in particolare attraverso la creazione di nuovi add-on, che in virtù della licenza LPGL sotto cui è rilasciato JADE sono stati resi disponibili al pubblico.

Lo sviluppo di questo software è poi proseguito fino ad oggi: la versione più aggiornata, nel momento in cui stato scritto questo testo, è la 4.0.1. Per capire come è organizzata la piattaforma su cui è stato sviluppato il software per questa tesi è quindi opportuno analizzarne ora l'architettura.

2.6.2 Elementi di base dell'architettura

JADE aderisce allo standard FIPA, e implementa quindi la struttura e gli elementi fondamentali che tale standard definisce: è opportuno notare come i componenti descritti siano di natura logica, ovvero sono da intendersi come insieme di servizi; non è quindi necessario che ad ognuno di essi corrisponda un componente software distinto.

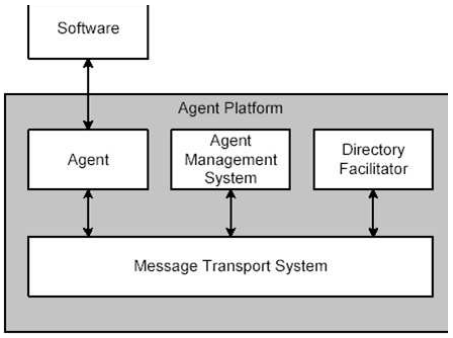

Figura 2.1: Elementi fondamentali di una piattaforma FIPA

Con riferimento alla figura 2.1, si può vedere come gli elementi fondamentali di una piattaforma secondo lo standard FIPA siano:

- *l'agente*: esso rappresenta l'entità principale che opera all'interno della piattaforma, ed è quindi in grado di svolgere un determinato insieme di azioni all'interno di essa, tra cui l'interazione con altri agenti, con la piattaforma stessa o con software esterno ad essa. Ad ogni agente deve essere associato un proprietario (*owner*), che può essere un utente umano oppure un'entità software. Ad ogni agente è inoltre assegnato un AID (*Agent IDentifier*), un identificatore univoco che permette di individuarlo con certezza all'interno della piattaforma in cui opera: esso è in genere costituito da una stringa, da un numero o da una combinazione di questi due elementi. Inoltre, a causa del fatto che un agente può utilizzare vari servizi di trasporto per spostarsi da

un nodo ad un altro, ad esso viene anche associato un indirizzo di trasporto, per poterlo identificare univocamente, questa volta all'interno della rete in cui opera;

- *l'Agent Management System (AMS)*: è un componente che ha come compito quello di supervisionare gli accessi alla piattaforma da parte di entità esterne, oltre che di monitorare le azioni che essi compiono; è fondamentale, secondo le specifiche FIPA, che all'interno di una piattaforma ve ne sia *uno e uno solo*. Quindi, se ad esempio un agente (esterno o appena creato) vuole operare all'interno della piattaforma è obbligato a effettuare una *registrazione presso l'AMS*: deve cioè segnalare la propria presenza e attendere che quest'ultimo gli conceda l'autorizzazione ad operare all'interno della piattaforma: in questo caso l'AMS provvederà poi ad assegnargli un AID opportuno. Per poter svolgere questo tipo di compito, l'AMS mantiene al suo interno un elenco di tutti gli AID assegnati e, per ognuno di essi, anche altre informazioni accessorie utili alla gestione degli agenti (come ad esempio il loro indirizzo). Un'altra funzione accessoria svolta da questo componente, che risulta molto utile, è quella del servizio di pagine bianche (*white page service*): ogni agente può utilizzarlo per conoscere quali agenti sono presenti all'interno della piattaforma in un determinato istante e in quale luogo si trovino;

- *il directory facilitator (DF)*: come per l'AMS, anche la presenza di questo componente è obbligatoria all'interno della piattaforma, anche se in questo caso, ve ne possono essere anche più di uno. Il suo compito principale è quello di fornire il cosiddetto servizio di pagine gialle (*yellow page service*): mantiene cioè al suo interno un elenco dei servizi offerti da ogni agente presente all'interno della piattaforma. In questo modo, un agente che offre un determinato servizio può comunicarlo al DF, che provvederà a memorizzare tale informazione. Simmetricamente, un altro agente che necessita di tale servizio può effettuare una query al DF per conoscere il nome e il luogo dell'agente che lo rende disponibile;

- *il message transport system (MTS)*: ha il compito di gestire e coordinare lo scambio di informazioni tra agenti; in particolare questi ultimi possono essere situati nella stessa piattaforma o anche in piattaforme diverse;

- *l'agent platform*: è la piattaforma per gli agenti mobili vera e propria, ovvero un componente che fornisce un ambiente uniforme in cui gli agenti possono svolgere i propri compiti, comunicare e spostarsi. È da notare come, secondo lo standard FIPA, ne facciano parte anche il sistema operativo e il calcolatore fisico su cui essa è in esecuzione;

- *il software esterno*: viene utilizzato come riferimento per indicare tutto il software non basato sulla tecnologia ad agenti software, ma che offrono servizi a cui gli agenti della piattaforma possono accedere;

Introdotti gli elementi fondamentali definiti dallo standard FIPA, risulta ora interessante analizzare sinteticamente come essi siano stati implementati in pratica in JADE.

Una piattaforma per agenti mobili JADE è costituita da uno o più contenitori (*container*), chiamati così proprio perché si tratta di entità in grado di ospitare agenti al proprio interno. Tra di essi, ve ne è presente uno (e uno solo) "speciale", chiamato contenitore principale (*main container*); esso ha la funzione di controllare e coordinare il funzionamento degli altri contenitori, detti anche *contenitori periferici*, oltre che da fungere da punto di riferimento per essi. È importante sottolineare come esista una corrispondenza univoca tra un contenitore principale e una piattaforma JADE: se infatti ne venisse avviato un altro, esso darebbe origine ad un'altra piattaforma JADE a sé stante, indipendente ed esterna quindi alla prima.

Ciò che rende il contenitore principale fondamentale per il funzionamento della piattaforma è la presenza obbligatoria al suo interno dell'AMS e di (almeno) un DF: essi sono stati implementati in JADE come agenti software, e vengono istanziati automaticamente all'avvio della piattaforma stessa; in questo modo si garantisce la conformità con le specifiche FIPA.

Sempre all'interno del contenitore principale vengono mantenute alcune informazioni essenziali per il funzionamento della piattaforma stessa: tra di esse si citano l'*Agent Container Table* (ACT), una tabella che mantiene al suo interno una sorta di mappa di tutti i contenitori della piattaforma, e l'*Agent Global Descriptor Table* (AGDT), che mette in correlazione le informazioni presenti nell'AMS riguardo a un agente con il suo AID e con altri dati utili alla sua gestione.

È infine da segnalare come, per facilitare la gestione della piattaforma, è stata sviluppata una GUI (*Graphical User Interface*), ovvero un'interfaccia grafica che permette di monitorare lo stato degli agenti e dei container in esecuzione in un dato istante. Quest'ultima prende il nome di RMA (*Remote Monitoring Agent*) e, come si può dedurre dal nome, è stata implementata anch'essa come un agente; esso viene avviato all'avvio della piattaforma (insieme quindi con l'AMS e il primo DF), ed ovviamente può essere chiuso e riavviato in qualsiasi momento, dato che il suo funzionamento non interferisce con quello della piattaforma stessa. Nella figura 2.2 si può osservare come si presenta in esecuzione, e di come esso permetta di eseguire un insieme di operazioni utili alla gestione di una piattaforma JADE, come ad esempio avviare, terminare, sospendere, trasferire un agente, etc...

Oltre all'RMA, JADE è fornito nativamente di alcuni agenti che forniscono altre funzioni accessorie; essi sono:

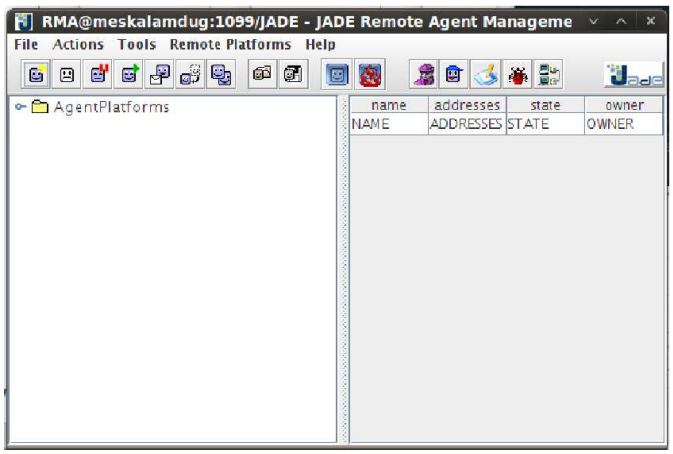

Figura 2.2: L'interfaccia grafica dell'RMA.

- il *Sniffer Agent*: si tratta di un agente che permette di monitorare, ed eventualmente ispezionare, i messaggi scambiati tra gli agenti che popolano la piattaforma;
- il *Dummy Agent*: è un agente di test, il cui scopo è quello di testare il funzionamento dello scambio di messaggi tra di esso ed un altro agente: in questo caso può memorizzare i messaggi scambiati con gli altri agenti e comporre messaggi ad-hoc per testare la risposta di altri agenti;
- il *Introspector Agent*: esso consente di visualizzare lo stato di ogni agente, le azioni di cui esso è capace e visualizzare quale esso stia attualmente eseguendo;
- il *Log Manager Agent*: è un agente che, selezionato un dato container, permette di eseguirne un logging approfondito; memorizza quindi tutti gli eventi che accadono in tale container e permette di analizzarne i dettagli successivamente;

Dopo aver fornito al lettore una breve introduzione alla piattaforma, verrà analizzato come sono state implementate le sue funzionalità principali.

2.6.3 Scambio di messaggi

Un'abilità fondamentale di cui sono dotati gli agenti è quella di poter comunicare tra di loro: in JADE questo avviene tramite unità informative discrete chiamate *messaggi*; in particolare, JADE ha implementato l'Agent Communication Language (ACL) definito dalle specifiche FIPA, e supporta quindi tutti i protocolli di trasporto da esso definiti: quale venga utilizzato tra questi ultimi per la trasmissione dei

messaggi dipende da dove sono situati i due agenti che vogliono comunicare. Si hanno infatti tre casi distinti, a seconda del fatto che i due agenti si trovino nello stesso contenitore, in due contenitori diversi all'interno della stessa piattaforma, o in due piattaforme diverse.

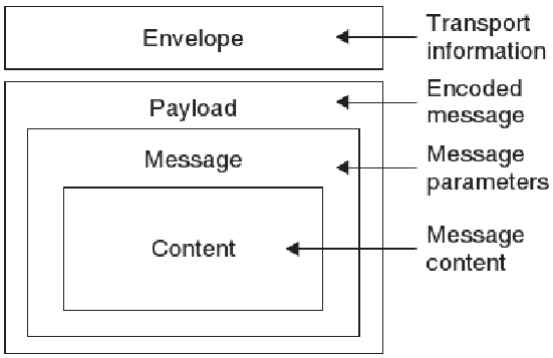

Figura 2.3: Struttura generale di un messaggio ACL.

Con riferimento alla figura 2.3, si può notare come un messaggio ACL sia essenzialmente diviso in due parti: la prima, chiamata *envelope*, contiene informazioni utili per la consegna del messaggio, come ad esempio mittente, destinatario, codifica utilizzata, etc... La seconda, chiamata *payload*, contiene invece il contenuto vero e proprio del messaggio che si vuole inviare al destinatario.

Nel caso di comunicazioni interne ad un contenitore si utilizza un protocollo denominato IMTP (*Internal Message Transport Protocol*), che non deve essere necessariamente standard dato che non coinvolge altri contenitori; esso viene infatti utilizzato anche per il trasporto di comandi interni utili alla gestione della piattaforma e per il monitoraggio dello stato dei container periferici.

Nel caso invece di comunicazioni tra due agenti situati in due contenitori diversi, ma sempre all'interno della stessa piattaforma, viene utilizzato il meccanismo di invocazione remota di metodi fornito da Java (RMI). Grazie ad esso, infatti, è possibile ottenere un riferimento ad un oggetto in modo trasparente rispetto al fatto che si trovi in un altro contenitore: in questo caso, il contenitore che contiene l'agente destinatario del messaggio agisce da server, mentre quello che contiene il mittente svolge il ruolo di client.

2.6.4 I Behaviour

Per permettere l'esecuzione di compiti (o più propriamente, di *task*) da parte dei propri agenti, e quindi il raggiungimento degli obiettivi ad essi assegnati, JADE implementa un particolare paradigma, basato sul concetto di *behaviour* (comportamento). Un behaviour è implementato concretamente dalla classe `Behaviour`

contenuta nel package `jade.core.behaviour`; esso corrisponde ad un particolare task che l'agente dovrà eseguire durante il suo ciclo di vita all'interno della piattaforma. Le specifiche azioni che l'agente dovrà compiere per eseguire tale task andranno poi inserite al suo interno grazie ad appositi metodi che tale classe fornisce.

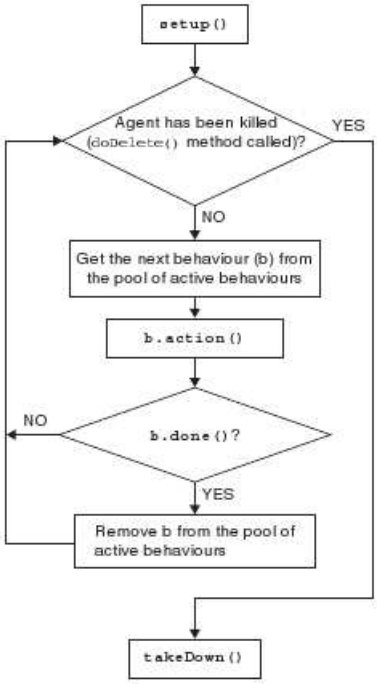

Figura 2.4: Schema generale di esecuzione di un behaviour

Lo sviluppatore di un agente può notificargli la presenza di un behaviour da eseguire grazie al metodo `addBehaviour()` che la classe `jade.core.Agent`, che implementa l'agente stesso, mette a disposizione. Ogni agente mantiene al proprio interno una lista dei behaviour da eseguire: l'ordine con cui questo avverrà sarà lo stesso con cui essi saranno stati notificati all'agente tramite il metodo prima citato. Una volta iniziata l'esecuzione di un behaviour (che avviene invocando il suo metodo `action()`), essa verrà portata avanti fino alla sua terminazione. A questo punto l'agente controllerà, tramite il metodo `done()` [4], se il task associato al behaviour è stato portato a termine con successo. In caso affermativo, l'agente inizierà ad eseguire il prossimo behaviour presente nell'elenco; in caso contrario, ricomincerà l'esecuzione del behaviour non terminato. Questo modello di esecuzione è schematizzato nella figura 2.4.

[4]Che infatti ritorna una variabile booleana, indicante appunto se il behaviour da considerarsi terminato oppure no

Le API di JADE forniscono nativamente, oltre alla classe base `Behaviour`, anche numerose altre classi che la estendono e permettono di creare logiche di comportamento anche complesse, rivelatesi poi di importanza fondamentale per la realizzazione del software oggetto di questa tesi; questo ovviamente non preclude la possibilità, qualora ve ne fosse la necessità, di creare da zero dei behaviour *ad-hoc* per scopi specifici.

Si citano in questa sede alcuni tra i più importanti, che vengono implementati dalle seguenti classi:

- `OneShotBehaviour`: si tratta di una sottoclasse della classe `Behaviour` il cui metodo `done()` restituisce sempre il valore `true`. Essa permette quindi di eseguire behaviour di tipo *one-shot*: il loro metodo `action()` verrà cioè eseguito una e una sola volta;

- `CyclicBehaviour`: è una classe simmetrica alla precedente, il cui metodo `done()` ritorna cioè sempre il valore *false*. Questo tipo di behaviour verrà quindi eseguito ciclicamente fino alla terminazione dell'agente stesso;

- `SequentialBehaviour`: questa classe permette di creare un behaviour composto da uno o più sotto-behaviour, che possono essere aggiunti tramite il metodo `addSubBehaviour()`. Questi ultimi verranno eseguiti in sequenza nell'ordine specificato, e il behaviour terminerà solo quando saranno terminati tutti i sotto-behaviour;

- `FSMBehaviour`: costituisce un'ulteriore estensione della classe precedente. Tramite questa classe è possibile specificare dei veri e propri automi a stati finiti deterministici, in cui ogni stato corrisponde a un sotto-behaviour. Ad ogni sotto-behaviour viene inoltre data la possibilità di restituire un valore intero, a seconda dell'esito che ha avuto la sua esecuzione: tale valore deciderà quale sarà il prossimo sotto-behaviour che verrà eseguito; in questo modo si emulano le transizioni presenti in un automa a stati finiti. L'esecuzione del behaviour inizia sempre dal sotto-behaviour che corrisponde allo stato iniziale dell'automa (ve ne deve essere uno e uno solo), e termina quando è terminata l'esecuzione del metodo `action()` di uno dei sotto-behaviour marcati come stati finali dell'automa (gli stati finali di un'automa possono essere uno o più di uno);

- `ParallelBehaviour`: è una classe che permette di eseguire i sotto-behaviour specificati in modo concorrente. In questo caso è possibile specificare se il behaviour è da considerarsi terminato quando tutti i suoi sotto-behaviour sono terminati, oppure quando il primo sotto-behaviour ha ultimato la sua esecuzione (in quest'ultimo caso, l'esecuzione degli altri sotto-behaviour verrà interrotta);

2.6.5 Mobilità

Essendo JADE una piattaforma ad agenti mobili, la mobilità risulta una delle funzionalità chiave per il suo corretto funzionamento: essa verrà qui descritta, seppur brevemente, al fine di permettere al lettore una migliore comprensione di come possa effettivamente venire programmato il trasferimento di un agente da un contenitore all'altro.

Come già anticipato, JADE fornisce una mobilità di tipo *not-so-weak*, implementata tramite il meccanismo della serializzazione fornito nativamente da Java. Al programmatore che intende permettere il trasferimento di un agente da un contenitore all'altro, vengono in particolare forniti tre metodi, chiamati `beforeMove()`, `doMove()` e `afterMove()`, facenti parte della classe `Agent`. Viene qui ora brevemente riassunta la sequenza degli eventi che accadono quando un agente si trasferisce da un contenitore all'altro:

1. Durante l'esecuzione del behaviour corrente, l'agente esegue il metodo `doMove()`, la cui funzione è quella di segnalare all'agente di dover iniziare il trasferimento verso un altro contenitore, che viene passato come parametro al metodo. È da notare come quest'ultimo debba essere necessariamente richiesto all'AMS, poiché mantiene al suo interno una lista dei contenitori con sufficienti risorse per poter ospitare l'agente.

2. L'agente, una volta scelto il contenitore di destinazione, procede con la copia al suo interno del proprio codice e heap; subito dopo verrà creata all'interno del contenitore di destinazione una nuova istanza dell'agente stesso.

3. Nel caso la copia effettuata dall'agente sia andata a buon fine, un attimo prima di creare una nuova istanza di sé stesso nel contenitore di destinazione, l'agente esegue il metodo `beforeMove()`: in tale metodo dovranno quindi essere inserite le eventuali ultime azioni che l'agente dovrà compiere prima della terminazione del suo thread originale, come ad esempio il rilascio delle risorse da esso utilizzate al momento della partenza. È interessante notare come tale metodo venga eseguito *solo* dopo che il trasferimento del codice e dell'heap dell'agente ha avuto successo: nel caso contrario, infatti, l'agente sarebbe costretto a tornare in competizione con gli altri agenti per riappropriarsi delle stesse risorse.

4. Giunto a questo punto, l'AMS provvede a terminare definitivamente il thread dell'agente nel contenitore di partenza e ad avviarne uno nuovo in quello di destinazione. Prima per di effettuare quest'ultima azione, viene eseguito il metodo `afterMove()`, che dovrà quindi contenere eventuali azioni da compiere non appena attivato, come ad esempio il ripristino del suo heap.

Questa appena presentata è la sequenza di operazioni che avviene quando un agente si trasferisce da un contenitore ad un altro della stessa piattaforma. Nel caso il due contenitori non appartengano alla stessa piattaforma, la procedura rimane la stessa, tranne che per due dettagli:

- al metodo `doMove()` non viene più passato come parametro l'identificativo del contenitore di destinazione, ma quello della piattaforma di arrivo, che corrisponde al suo contenitore principale. Come conseguenza di ciò, l'agente viene in realtà trasferito prima al contenitore principale della piattaforma di arrivo, e da qui successivamente al contenitore di destinazione vero e proprio: l'intera operazione sarà coordinata dall'MTS;

- l'heap e il codice dell'agente non vengono trasportati tramite serializzazione, ma all'interno di speciali messaggi ACL, che possiedono in questo caso un proprio linguaggio e una propria semantica.

Per completare questa introduzione, si segnala al lettore che esistono tre metodi simmetrici a quelli trattati in questa sezione, chiamati `beforeClone()`, `doClone()` e `afterClone()` che possiedono come finalità quella di clonare un agente in un altro contenitore. A differenza dei metodi prima esposti, essi si limitano a copiare solo il codice dell'agente e non anche il suo heap; inoltre, la copia di un agente in un nuovo contenitore non comporta la terminazione del thread dell'agente originario.

2.6.6 Jade-Leap

Fra i numerosi add-on disponibili che rendono questa piattaforma ad agenti mobili estremamente versatile, ce ne è uno in particolare che merita attenzione in quanto permette lo sviluppo di sistemi che possono funzionare non solo su hardware complessi ma, parte di essi, anche su dispositivi a risorse limitate quali palmari o cellulari. Per venire incontro a queste esigenze è stato necessario adattare il framework in quanto, per svariate ragioni, il progetto originale non era in grado di rispondere a tali esigenze:

- l'occupazione di memoria del runtime di JADE è troppo elevata.

- JADE richiede una JVM10 1.4 o superiore mentre nella maggioranza dei dispositivi portatili è presente solo KVM11.

- i collegamenti wireless hanno caratteristiche diverse rispetto alla loro controparte cablata come ad esempio alta latenza, scarsa banda, connettività ad intermittenza e assegnamento di IP dinamico che devono essere prese seriamente in considerazione.

Su questi presupposti è stato sviluppato Jade-Leap che, quando combinato con Jade, rimpiazza alcune parti del kernel JADE formando un ambiente runtime modificato (che prende il nome di "JADE powered by LEAP") e che può essere impiegato in un ampia gamma di dispositivi che vanno dai server fino ai cellulari dotati di ambiente J2ME. Dal punto di vista del programmatore un ambiente Jade-Leap è perfettamente coincidente con uno Jade, in quanto si utilizzano le stesse API e gli stessi metodi di amministrazione; è da sottolineare però che non è possibile mescolare nella stessa piattaforma container appartenenti ai due framework (possono ovviamente comunicare attraverso MTP se sono su piattaforme separate).

In generale un ambiente Jade-Leap può essere eseguito in due modalità distinte:

- *stand-alone*: quando viene avviato un container completo su un dispositivo dove è presente un runtime JADE;

- *split*: quando il container viene diviso in due parti chiamate rispettivamente "FrontEnd", eseguito sul dispositivo dove è presente un runtime Jade, e "BackEnd", eseguito su un server remoto, collegate tra loro da un collegamento permanente (in senso logico).

La modalità split è utilizzata quando si ha a che fare con dispositivi mobili in quanto il FrontEnd è molto più leggero di un container completo, la fase di bootstrap è molto più veloce dal momento che tutte le comunicazioni per accedere alla piattaforma vengono eseguite dal BackEnd, ed infine anche il collegamento wireless è ottimizzato. È comunque da sottolineare che tutti questi processi sono del tutto trasparenti allo sviluppatore in quanto il set di API cui può accedere è esattamente lo stesso.

2.6.7 Jade-Leap for Android

Android è un sistema operativo per dispositivi mobili sviluppato da Google in collaborazione con la Open Handset Alliance. L'obbiettivo principale di Android è quello di fornire tutto ciò di cui un operatore, un vendor di dispositivi o uno sviluppatore hanno bisogno. Inoltre esso ha come principale caratteristica essere *open*, dove per *open* si intende:

- Android è open in quanto utilizza tecnologie open, prima fra tutte il kernel di Linux nella versione 2.6.

- Android è open in quanto le librerie API che sono state usate per la sua realizzazione sono esattamente le stesse disponibili ad un programmatore. Questo comporta che non ci sono quasi limiti alla personalizzazione dell'ambiente.

- Android è open in quanto il suo codice è open source, consultabile da chiunque possa contribuire a migliorarlo, lo voglia documentare o semplicemente voglia scoprirne il funzionamento.

Android inoltre, è completamente sviluppato in linguaggio di programmazione Java, permette di interfacciarsi con il sistema operativo, controllare le risorse hardware del dispositivo mobile e di sviluppare delle Android GUI (*Graphic User Interface*).

Le grandi potenzialità di Android hanno spinto il team di Jade a fornire il supporto necessario per utilizzare una piattaforma Jade-Leap su una piattaforma Android. Attualmente, l'ultima versione sviluppata di Jade-Android è la 1.2, il cui funzionamento è garantito solo sulla versione di Jade-Leap 4.0 o successive, e su piattaforma Android 1.5, 1.6, 2.1, 2.2. Essa fa uso della modalità di esecuzione split di Jade-Leap.

Purtroppo vi sono ancora delle limitazioni legate alle prestazioni delle risorse disponibili su un dispositivo mobile. È bene ricordare infatti che, ad oggi, per quanto un telefono cellulare o un palmare possa essere non si avvicinerà nemmeno alle prestazioni di un pc di con caratteristiche anche medio-basse.

Le limitazioni principali sono:

- L'add-on Jade-Android non supporta l'esecuzione di più di un agente sullo split container lanciato su Android.

- L'add-on Jade-Android non consente la mobilità degli agenti.

In ogni caso, anche Jade-Android add-on fornisce lo stesso set di API della piattaforma JADE.

Capitolo 3

Parametri biometrici

Le metodologie per l'identificazione del singolo individuo al fine di avere un controllo sulle informazioni cui possa accedere, sui luoghi cui sia autorizzato a recarsi, o ai sistemi cui abbia effettivamente il diritto di utilizzo, hanno subito nel corso del tempo numerosi sviluppi passando da sistemi di verifica tradizionali basati sul possesso di un oggetto (ad esempio un badge) o sulla conoscenza di una informazione (ad esempio un PIN), fino ad arrivare a sistemi più complessi, ove si cerca di collegare alla persona che richiede l'autenticazione una informazione estremamente personale e non facilmente riproducibile di cui lei stessa è portatrice: in questo caso un parametro biometrico.

L'adozione delle conoscenze ottenute nel campo della biometria in particolare, hanno comportato un deciso passo in avanti nella affidabilità dei sistemi di identificazione. I metodi precedentemente utilizzati infatti non garantivano che una persona in grado di essere identificato fosse in realtà il legittimo proprietario dell'informazione o dell'oggetto necessari per il processo di autenticazione. Non è il caso dei sistemi basati su parametri biometrici, in cui l'informazione è inseparabile dall'utente e non può essere per così dire ceduta a terzi.

3.1 Definizioni generali

L'idea base di un sistema biometrico è, come detto in precedenza, quella di spostare l'attenzione sull'accertamento della presenza fisica dell'individuo che richiede l'accesso a qualcosa di protetto e di verificarne quindi l'autorizzazione. Quello che si ricerca è sviluppare un sistema del tutto automatico che esegua una rilevazione di alcune delle caratteristiche fisiche del soggetto e le possa confrontare con altre registrate in precedenza e opportunamente convalidate, al fine di trovare quello che in gergo si chiama match biometrico.

Le caratteristiche biometriche si possono dividere principalmente in due categorie in base alla loro natura: fisiologiche e comportamentali.

Alla prima categoria appartengono attualmente:

- impronte digitali (caratteristica adottata per lo svolgimento della tesi);
- retina/iride;
- mano;
- viso;

mentre alla seconda appartengono:

- voce;
- grafia;
- stile di battitura.

In linea generale il funzionamento di un sistema biometrico per l'identificazione è il seguente: tramite uno scanner (ovviamente specifico per il tipo di dato che si vuole acquisire) si ricava una impronta biometrica dalla quale viene estratto un modello che viene fatto combaciare (fase di matching) con un altro preventivamente salvato (la cui autenticità è fidata) in un database; se i modelli "coincidono" per un valore percentuale superiore ad una soglia fissata l'utente risulta autenticato. Questa strategia di tipo probabilistico è dettata dal fatto che, a causa di fattori sia fisici che tecnologici, è impossibile ottenere due scansioni perfettamente identiche dello stesso dato biometrico. Al fine di decidere quali siano le caratteristiche migliori che un individuo possa presentare e che ben si prestano ad essere utilizzate in tale procedimento è necessario prendere in considerazione le seguenti proprietà:

- universalità: la caratteristica biometrica da valutare deve essere posseduta da tutti i membri della popolazione;
- unicità: ogni firma biometrica deve differire da quella di ogni altro membro della popolazione;
- invarianza: la firma biometrica non deve variare nelle diverse condizioni di rilevamento e nel tempo;
- misurabilità: le caratteristiche devono poter essere misurabili quantitativamente e facilmente ottenibili;
- performance: si riferisce all'accuratezza di riconoscimento raggiungibile e alle risorse necessarie per il conseguimento;

- accettazione: in riferimento al grado di accettazione del sistema di rilevazione biometrica da parte dei membri della popolazione;

- robustezza: quanto il sistema biometrico si presta a tentativi di accesso fraudolento.

È comunque da sottolineare che anche tale sistema non è esente da limiti, infatti queste metodiche non identificano persone bensì "corpi". Falsificare le informazioni biometriche è un'operazione molto complessa ma comunque fattibile.

Misurazione delle performance

Ci sono essenzialmente 3 parametri che permettono di valutare un sistema biometrico e sono: dimensione, velocità e accuratezza.

Le dimensioni del modello che si estrae dall'impronta biometrica, intese come dimensioni in byte, hanno significato in relazione al supporto ove verrà memorizzato; tale dato assume importanza soprattutto se si prevede l'utilizzo di dispositivi a memoria limitata come ad esempio le smartcard.

La velocità del sistema è ovviamente una discriminante molto importante in quanto, se il processo di autenticazione impiegasse troppo tempo per ottenere un risultato, diventerebbe inutilizzabile per scopi pratici (si pensi ad esempio all'identificazione dei dipendenti all'ingresso di un'azienda).

L'accuratezza risulta un parametro critico da determinare a causa dell'approccio probabilistico adottato dal sistema e va commisurata al rischio cui si incorre in caso di falsa accettazione.

I tipi di errore cui un sistema biometrico può incorrere sono essenzialmente due:

- *falsa accettazione* (falso positivo): un utente non autorizzato viene autenticato dal sistema perché la sua impronta risulta abbastanza simile ad un modello precedentemente archiviato. L'indice associato a tale tipo di errore prende il nome di FAR (False Acceptance Rate) ed é misurato come rapporto tra il numero di accessi effettuato da utenti non autorizzati ed il numero di accessi totali;

- *falso rigetto* (falso negativo) : un utente autorizzato viene rifiutato dal sistema perché la sua impronta non risulta abbastanza simile al modello con cui è stata comparata. L'indice associato a tale tipo di errore prende il nome di FRR (False Reject Rate) ed é misurato come rapporto tra il numero di accessi rifiutati, da parte di utenti che ne detengono invece il relativo diritto di accesso, ed il numero di accessi totali;

I due parametri variano in modo inverso ed il loro punto di intersezione viene definito indice di uguaglianza degli errori (EER), ed è in qualche modo un punto

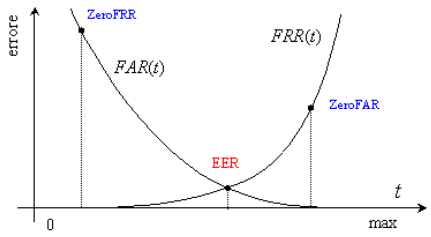

Figura 3.1: Relazione tra FAR e FRR al variare della soglia di discriminazione

di ottimo caratteristico di ogni implementazione biometrica. Teoricamente questo indice dovrebbe esprimere l'accuratezza per comparare sistemi differenti.

3.2 Impronte digitali

La caratteristica biometrica maggiormente utilizzata oggigiorno per i metodi di autenticazione è senza ombra di dubbio l'impronta digitale[18], che si può considerare come l'insieme di creste e valli rilevate sui polpastrelli generalmente sulla prima falange.

Figura 3.2: Creste e valli in un'impronta digitale

È comunque da tenere in considerazione una problematica nell'utilizzo di questo parametro biometrico: circa il 4% della popolazione non può fornire impronte di buona qualità. Si pensi ai lavoratori manuali che possono presentare ferite frequenti sui polpastrelli oppure pieghe cutanee dovute all'immersione in acqua per un periodo prolungato, si pensi agli anziani affetti da un naturale invecchiamento della pelle, si pensi ai portatori di particolari malattie del derma che inducono allo spellamento ecc. Tale problema deve essere messo a confronto ovviamente con le specifiche del sistema che si vuole realizzare: se per esempio si vuole utilizzare l'impronta digitale come parametro biometrico per l'identificazione dei dipendenti all'interno di una banca non ci sono problemi, ma se si prevedesse di usarlo come

firma digitale per tutti i cittadini di uno Stato sarebbe necessario ponderare sui rischi ai quali si andrebbe incontro.

3.2.1 Persistenza ed individualità

L'autenticazione attraverso l'uso delle impronte digitali si basa su due premesse fondamentali:

- persistenza: le caratteristiche dell'impronta non cambiano attraverso il tempo;
- individualità: l'impronta è unica da individuo ad individuo.

Mentre la persistenza è stata provata scientificamente, per l'individualità bisogna accontentarsi a dati statistici; fino ad oggi, nonostante le numerose ricerche effettuate, non sono ancora state trovate due impronti digitali identiche.

3.2.2 Parametri per la classificazione delle impronte

Come abbiamo già accennato precedentemente l'impronta digitale è costituita da un insieme di linee di rilievo e spazi posizionati sul polpastrello: le creste e le valli. L'analisi delle sue proprietà ha portato, nel corso degli anni, a due approcci differenti tra loro per quanto riguarda ciò che deve essere rilevato al fine di poter avviare un processo di confronto: l'individuazione di caratteristiche globali l'uno, l'individuazione di caratteristiche locali l'altro.

Nell'estrazione delle caratteristiche globali l'attenzione è rivolta alla struttura ad alto livello; infatti si considerano le linee di flusso disegnate dall'andamento delle creste per individuare le seguenti caratteristiche:

- modello delle creste: il loro andamento traccia delle figure in cui si possono riconoscere forme particolari. Nel corso degli anni i gruppi di ricerca ne hanno identificate molte, le più comuni sono comunque le seguenti:
  1. loop: struttura più comunemente rilevabile (si trova circa nel 65% delle acquisizioni). Sono identificabili dalla curva molto stretta tracciata dalla cresta;
  2. arco: la sua struttura è composta da una cresta che traccia una curva più aperta di quella formata dal loop;
  3. bidelta concentrico: consiste in una cresta che disegna un cerchio completo.
- area del modello: corrisponde alla parte dell'impronta che racchiude tutte le caratteristiche globali ed è delimitata dalle due line più esterne;

- core: è un punto posizionato approssimativamente al centro dell'impronta e viene usato come riferimento per la lettura e la classificazione;
- delta: è il punto della prima biforcazione posizionato di fronte al punto di divergenza delle due type lines ovvero le linee che delimitano l'area del modello;
- numero delle creste: è il numero di creste che si contano tra il delta e il core.

Nelle caratteristiche locali si ricercano invece alcuni comportamenti anomali delle creste, nella fattispecie i principali sono le terminazioni e le biforcazioni. I punti identificati da queste particolarità vengono definite minuzie.

Le caratteristiche utilizzate per identificare le minuzie sono le seguenti

- tipologia: a partire da terminazioni e biforcazioni vengono definite una serie di elementi caratteristici:
  1. terminazione: si dice di una cresta che termina bruscamente;
  2. biforcazione: il punto in cui una cresta si divide in due;
  3. punto o isola: una cresta tanto corta da sembrare un punto;
  4. lago: una cresta che si divide in due e poi si riunisce, creando una zona chiusa al cui interno non ci sono altre creste;
  5. cresta corta: una cresta di piccole dimensioni, ma non tanto da essere considerata un punto;
  6. sperone: combinazione di due creste indipendenti e una biforcazione;
  7. incrocio: cresta indipendente che attraversa due creste parallele.
- orientamento: si riferisce alla particolare direzione posseduta dalla minuzia in oggetto;
- frequenza spaziale: si riferisce a quanto lontano si trovano le creste nelle vicinanze della minuzia;
- curvatura: si riferisce al tasso di cambiamento dell'orientazione della cresta nel punto in questione;
- posizione: indica le coordinate piane della minuzia in senso assoluto o relativamente al delta e al core.

Un'impronta è formata da circa 100 minuzie e l'FBI ha stabilito che " non possono esistere " due individui le cui impronte digitali coincidano per più di 8 minuzie: ne segue che, impostando una soglia minima di 20 minuzie, si raggiunge un adeguato livello di sicurezza (nello svolgimento della tesi è stato scelto di aumentare tale soglia a 30 vista l'elevata qualità di acquisizione dello sscanner in uso).

Figura 3.3: Tipologie minuzie

3.2.3 Metodologie e strumenti per la rilevazione delle impronte digitali

Le immagini riguardanti le impronte digitali possono essere rilevate utilizzando principalmente due metodi detti inked, se si riferiscono a processi eseguiti off-line, e live-scan se si riferiscono invece a processi eseguiti on-line.

Nei primi metodi solitamente un addetto è incaricato della rilevazione dell'impronta tramite la pressione del dito dell'utente bagnato di inchiostro su un foglio di carta, successivamente acquisito digitalmente attraverso un normale scanner per documenti.

Per quanto riguarda invece i sistemi live-scan 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. La risoluzione di acquisizione è stata standardizzata dall'FBI in 500 dot per inch (dpi).

In commercio esistono molte tipologie di scanner; una delle tecnologie sicuramente più diffuse nel mercato è quella ottica 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. 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 prestazioni migliori rispetto agli scanner ottici, vista la sua tolleranza

ai disturbi causati dallo sporco. Il sensore funziona infatti come una piastra di 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 disturbate da scariche elettrostatiche.

Infine esistono anche scanner che utilizzano onde sonore sulle frequenze degli ultrasuoni (da 20 KHz fino ad alcuni GHz) 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 e si adatta benissimo anche a condizioni di luce molto intensa, dove i sensori ottici presentano notevoli problemi. Il lato negativo di tale tecnologia è che ad oggi risulta essere ancora molto costosa e quindi, nonostante l'estrema precisione delle misurazioni, non è ipotizzabile un suo utilizzo su larga scala.

Biometrika Fx2000

Lo scanner che è stato utilizzato durante lo svolgimento di questa tesi è uno di quelli a tecnologia ottica e viene prodotto in Italia dalla ditta Biometrika[19]. Il modello in questione è l'Fx2000, dotato di un'area sensibile di 13.2 mm x 25 mm e di una risoluzione di 569 dpi (*dot per inch*).

Figura 3.4: Lo scanner Fx2000

3.2.4 Processo di matching delle impronte

Fin dagli anni 60 i più importanti istituti di polizia del mondo hanno investito una grande quantità di sforzi al fine di sviluppare quello che prende il nome di Automated Fingerprint Identification System, più brevemente AFIS, ovvero una metodologia di identificazione biometrica basata sulla tecnologia digitale per il

trattamento delle immagini dalle quali ottenere, memorizzare ed analizzare i dati relativi all'impronta.

Tali sistemi, o più in generale un qualsiasi sistema automatico di verifica di identità, si basa su quattro componenti fondamentali:

- acquisizione: fase nella quale si utilizzano particolari metodologie, esposte nei paragrafi precedenti, per il rilevamento dell'impronta;
- rappresentazione: passo fortemente legato alla fase di verifica, dato che la rappresentazione dell'impronta deve contenere tutte le informazioni necessarie all'algoritmo di matching. I template possono essere basati o 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: fase nella quale si effettua la ricerca nell'immagine dell'impronta delle minuzie base ovvero biforcazioni e terminazioni. Tale fase è strettamente legata al risultato finale in quanto maggiore sarà la precisione dell'estrazione delle minuzie, maggiore sarà la qualità del template finale. Dopo un'applicazione di algoritmi necessari a ridurre il rumore o eventuali degradazioni dell'immagine, viene selezionata la parte centrale di quest'ultima e viene binarizzata, ovvero partendo dall'immagine in scala di grigio si decide se un punto appartiene oppure no ad una cresta e si trasforma questa decisione in una mappa in bianco e nero. A questo punto viene avviata la fase di estrazione vera e propria dove viene processata l'intera mappa (ovvero l'immagine binarizzata) identificando i pattern di pixel che indicano la fine o la divisione delle creste (ovvero la presenza delle minuzie base); come coordinate della minuzia lungo l'asse x e y viene preso in considerazione il pixel più estremo della cresta binarizzata ovvero quello appartenente alla sua parte finale;
- matching: fase che finalizza tutto il lavoro eseguito nelle precedenti parti. Il modulo di matching deve determinare se due template di input corrispondono alla stessa impronta e questo avviene attraverso una ricerca di una possibile sovrapposizione delle minuzie. Risulta chiaro che, accanto alla definizione dell'algoritmo, deve anche essere stabilita una soglia che definisce il numero di sovrapposizioni minimo tra minuzie che si deve ottenere per dichiarare due impronte appartenenti allo stesso soggetto. Ci sono poi da tenere in considerazione alcune problematiche:
  - il dito, in scansioni differenti, può essere posizionato sia in posizioni diverse sul vetro dello scanner sia con orientazioni diverse: si necessita quindi di inserire nell'algoritmo di matching una fase di controllo e normalizzazione della posizione mediante traslazioni e rotazioni;

- il dito può esercitare una differente pressione sul vetro causando una diversa scala rispetto al template di riferimento: può succedere quindi che le minuzie risultino tutte più ravvicinate;
- minuzie definite "false" possono essere presenti sia sul template originale sia in quello live: questo problema può essere causato o da fattori inerenti l'hardware, come ad esempio sporcizia sul lettore, oppure a causa di un errore del software che, in fase di selezione della parte significativa dell'impronta, la taglia in modo scorretto generando in questo modo delle terminazioni inesistenti sul bordo. Può verificarsi ovviamente anche il problema duale ovvero che le minuzie vere non vengano rilevate correttamente;
- possono essere presenti delle deformazioni non-lineari dovute all'elasticità intrinseca della pelle che sono molto difficili da eliminare. Fortunatamente all'interno di una piccola porzione dell'immagine tali perturbazioni risultano identiche e quindi possono essere non considerate.

3.3 Informazioni biometriche e Giurisprudenza

Le caratteristiche biometriche di un individuo, intese come caratteristica fisica o comportamentale, non trovano una definizione all'interno del mondo giuridico (allo stesso modo non si trovano regolamentazioni precise per le tecnologie che ne fanno uso) ma si fa riferimento a ciò che è proposto dalla sfera scientifica. Parlando a livello generale, l'elemento biometrico, non trova quindi una normativa che lo tuteli in senso stretto, ma si fa riferimento ai parametri costituzionali della salute ed integrità fisica (art. 32 Cost.).

In tale prospettiva si possono citare i seguenti articoli:

- art. 13 della Carta Costituzionale che garantisce l'inviolabilità della libertà personale;
- art. 33 della Legge n.833 del 23 dicembre 1978 che esclude la possibilità di accertamenti e di trattamenti sanitari contro la volontà del paziente e non ricorrono i presupposti dello stato di necessità;
- art. 5 del Codice Civile che impedisce gli atti dispositivi del proprio corpo che incidano sull'integrità fisica.

In mancanza di una legale definizione del dato biometrico si ritiene di adottare il concetto espresso dal Gruppo dei Garanti Europei (istituito sulla base dell'art. 29 della direttiva 95/46/CE sulla protezione dei dati personali, e pertanto noto come "Working Party Art. 29") nel documento adottato il 1 agosto 2003, laddove

si dice che [. . .] i dati biometrici possono sempre essere considerati come informazioni concernenti una persona fisica in quanto sono dati che, per la loro stessa natura,forniscono indicazioni su una determinata persona.

Le problematiche relative all'uso di tecnologie basate su sistemi biometrici, sono state dibattute in diversi sedi i cui riferimenti principali sono:

- Unione Europea: il cui Gruppo dei Garanti Europei ha prodotto un documento diventato punto di riferimento in materia[20];
- OECD (Organisation for Economic Co-operation and Development): che ha analizzato le caratteristiche dei sistemi biometrici e le loro relazioni rispetto alla sicurezza e alla privacy[21];
- ICAO (Internation Civil Aviation Organization): che ha proposto l'utilizzo di caratteristiche biometriche nei passaporti e in altri documenti di viaggio leggibili automaticamente al fine di velocizzare il transito dei passeggeri attraverso i controlli nelle aree aeroportuali a fini di sicurezza.

Sono poi da tenere in considerazione i principi sanciti dal codice della privacy dove all'art. 4 lettera b) del D.Lgs. n. 196/2003 definisce il dato personale come " qualunque informazione relativa a persona fisica, persona giuridica, ente od associazione, identificati o identificabili, anche indirettamente, mediante riferimento a qualsiasi altra informazione, ivi compreso un numero di identificazione personale ". In funzione di tale definizione, è evidente come il trattamento di dati biometrici sia, nella fattispecie, un trattamento di dati personali anche in forma di template, in quanto è sempre possibile considerarlo come un'informazione relativa ad una persona fisica " identificata o identificabile" anche attraverso " uno o più elementi specifici caratteristici della sua identità fisica ". Ai dati biometrici dunque si applicano integralmente i principi del D. Lgs. n. 196/2003 in materia di protezione dei dati personali, fin dalla fase di enrollment. Il trattamento dei dati biometrici può dunque essere considerato lecito solo se tutte le procedure utilizzate, a partire dall'iscrizione, vengono effettuate conformemente alle disposizioni di legge. Queste osservazioni sono importanti poiché giustificano da un lato la convenienza di avere i template su un server sicuro e dall'altro su un supporto rimovibile in possesso dell'utente. Per maggiori informazioni si veda [22]

Capitolo 4

Sistema sviluppato

Lo scopo principale di questo lavoro è quello di sviluppare un sistema di autenticazione biometrico bastato su agenti mobili, dove per autenticazione si intende un processo di verifica dell'identità di un soggetto, e per biometrico s'intende l'utilizzo delle tecniche tipiche della rilevazione di caratteristiche fisiologiche e comportamentali di cui si è già parlato nel capitolo 3.

Come già sottolineato, nonostante il movimento legato agli agenti mobili sia in grande fermento, in letteratura non si trovano molti esempi di un suo utilizzo legato a scopi di autenticazione anche se, all'interno del nostro gruppo dipartimentale, sono state svolte diverse ricerche in questo campo.

Partendo da questi presupposti si è pensato alla possibilità di allargare tale metodologia anche al mondo dei dispositivi mobili di ultima generazione, ovvero gli smartphone. Questi, oggigiorno, si stanno diffondendo in modo massiccio, e la creazione di nuove applicazioni da far girare al loro interno sta aprendo un nuovo orizzonte per gli sviluppatori.

L'innovazione introdotta consiste nel creare un sistema per l'autenticazione di utenti i quali dispongano come strumento di interazione verso il sistema non solo macchine fisse messe a disposizione ad hoc dal soggetto controllante ma anche di *mobile device* che possono essere di proprietà dell'utente oppure forniti in comodato d'uso.

Per lo sviluppo del sistema sviluppato in questo lavoro di tesi si è ipotizzato che l'utente possegga uno smartphone con sistema operativo Android 2.1 nel quale possa essere installato non solo un runtime di JADE-Leap Android, ma anche del software che permetta di utilizzare algoritmi propri della crittografia, sia a chiave pubblica sia a chiave privata, che risulteranno fondamentali per garantire la segretezza e la fidatezza delle informazioni scambiate durante la fase di autenticazione.

La demo realizzata utilizza il dispositivo mobile come deposito per il template dell'utente mentre la fase di matching viene seguita all'interno di un server. Ma prima di procedere ad una descrizione dettagliata del funzionamento e dell'archi-

tettura interna del progetto sviluppato, è necessario definire in modo chiaro gli aspetti legati all'autenticazione e capire il perché ci si orienta verso metodologie biometriche.

4.1 Autenticazione a più fattori

Un fattore di autenticazione può essere un'informazione o un processo usati per verificare l'identità di una persona a scopi di sicurezza. Quando un soggetto vuole procedere alla fase di autenticazione presenta, quindi, delle credenziali che possono essere classificate in:

- qualcosa che l'utente *conosce*;
- qualcosa che l'utente *possiede*;
- qualcosa che l'utente *ha in sé e può fisicamente esplicitare*;

Si parla di *strong authentication* (autenticazione a due o più fattori) quando vengono utilizzati almeno 2 fra le tipologie di credenziali sopra riportate.

4.1.1 Qualcosa che l'utente *conosce*

Questo tipo di credenziale è caratterizzata dalla conoscenza di un *PIN* o di una *password*, generalmente alfanumerica e di lunghezza variabile. L'utente deve essere l'unico depositario di questa informazione che fornirà al sistema per dimostrare la sua identità. Sono varie le problematiche derivanti dall'impiego di tale metodologia:

- l'utente deve essere conscio che l'inserimento della propria password su un sistema comporta un atto di fiducia nei confronti di quest'ultimo, in quanto potrebbe essere gestito da un soggetto maligno il cui unico scopo è ottenere le informazioni personali;
- l'utente poi può dimenticare questa informazione visto ormai l'enorme uso che ne si fa per molteplici servizi e risulta quindi facile confondersi. D'altra parte la sua annotazione scritta porta ad altri problemi come la copia o il furto;
- infine il segreto può essere indovinato o attraverso metodologie di forza bruta, che possono essere combattute usando password alfanumeriche molto lunghe, o attraverso tecniche basate su dizionari o attraverso stringhe create sulla base del profilo del soggetto, per esempio si tende ad usare come password la propria data di nascita oppure il nome dei familiari o una loro combinazione;

4.1.2 Qualcosa che l'utente *possiede*

Credenziale basata sul solo possesso di un oggetto non facilmente clonabile che garantisce all'utente di effettuare l'autenticazione. Il più delle volte si tratta di possedere un *badge magnetico* oppure una *chiave meccanica* o un qualche dispositivo elettronico, come ad esempio una *chiave USB* o una *smartcard*.

È una tipologia eterogenea, nel senso che sono molteplici gli oggetti che possono costituire credenziale, e il livello di protezione è fortemente dipendente dall'oggetto scelto. Ad esempio una smartcard con capacità crittografiche fornirà una protezione maggiore rispetto ad un badge magnetico. In ogni caso c'è da risolvere il problema legato allo smarrimento e/o alla sottrazione dell'oggetto.

4.1.3 Qualcosa che l'utente *ha in sé e può fisicamente esplicitare*

In questo caso ci si trova nell'ambito della verifica di una caratteristica fisiologica o comportamentale del soggetto che richiede di essere autenticato, ovvero di una sua *caratteristica biometrica*. Tra le varie caratteristiche (presentate nel capitolo 2) quella maggiormente utilizzata è l'impronta digitale.

Questo tipo di credenziale presenta alcune problematiche come il costo necessario al rilevamento della caratteristica, l'invasività che per alcuni utenti potrebbe essere percepita come lesiva della propria privacy, l'accuratezza e la verifica dei vari sistemi di acquisizione che sono disponibili sul mercato e la non sostituibilità del parametro, nel senso che se il parametro si danneggia si ha un numero limitato di possibilità di sostituirlo.

I lati positivi, però, sono certamente maggiori, in quanto si risolve facilmente il problema della *sottraibilità*: le credenziali non possono essere smarrite, dimenticate o sottratte, anche se tuttavia questo ultimo punto non va dato per scontato e dovrebbe essere valutato a seconda della caratteristica biometrica e della tecnologia utilizzata. Non è poi da sottovalutare la comodità in quanto l'utente non deve più ricordarsi password od oggetti ma porta sempre con sé le credenziali di accesso.

4.2 Sistemi di autenticazione ibrida

Cercando di rispondere alle richieste della *strong authentication* sono stati sviluppati dei sistemi nei quali vanno a fondersi le varie tipologie di credenziali appena descritte, al fine di creare un sistema che offra un adeguato livello di sicurezza.

I casi più frequenti di autenticazione ibrida sono:

- qualcosa che l'utente *conosce* e qualcosa che l'utente *possiede*: è il classico caso basato su carta magnetica e PIN numerico come ad esempio avviene per il sistema Bancomat/POS. Anche in questo caso la sicurezza è fortemente legata alle caratteristiche dell'oggetto posseduto: una smartcard crittografica rende il sistema molto più sicuro che non una tessera magnetica. La prima infatti è quasi impossibile da duplicare illecitamente, la seconda invece rappresenta soltanto un supporto di memorizzazione in cui non è effettuato alcun controllo sull'accesso ai dati, è quindi duplicabile quasi quanto una password. Ne sono una prova i vari episodi di clonazione delle tessere Bancomat;

- qualcosa che l'utente *possiede* e qualcosa che l'utente *possa esplicitare*: sistemi di questo tipo accoppiano informazioni biometriche che possono essere memorizzate su un particolare dispositivo. Il passaporto e la carta d'identità nazionale ne sono un esempio;

Durante lo sviluppo di questo progetto di tesi è stato scelto di adottare un sistema di autenticazione a tre fattori che si possono riassumere brevemente:

1. conoscenza di un PIN per l'accesso al sistema;

2. possesso di un telefonino di ultima generazione sul quale vengono memorizzate informazioni relative all'identità dell'utente e ai suoi parametri biometrici;

3. disponibilità al rilevamento live di un parametro biometrico dell'utente che verrà confrontato con quello salvato all'interno del telefonino;

4.3 Descrizione del sistema sviluppato

In questa sezione verranno descritti tutti gli scenari applicativi che sono stati sviluppati per questo lavoro di tesi. La presentazione qui si concentrerà sugli aspetti progettuali, in quanto l'analisi del codice e il manuale d'uso saranno esposti nei capitoli successivi. D'ora in poi quando si parlerà di parametro biometrico si intenderà un'impronta digitale rilevata tramite uno scanner.

L'algoritmo di matching utilizzato è basato sul confronto delle minuzie ed effettua quindi una ricerca al fine di trovare la migliore sovrapposizione tra esse tenendo conto sia delle possibili traslazioni sia delle possibili rotazioni verificabili. Nello sviluppo poi sono state utilizzati gli algoritmi RSA per la crittografia a chiave asimmetrica ed AES per la cifratura a chiave simmetrica.

4.3.1 Fase di Enrollment

La registrazione dell'utente è una parte estremamente importante al fine di un corretto funzionamento dell'intero sistema in quanto consiste nell'acquisizione dei dati personali dell'individuo, compresa la sua impronta digitale dalla quale poi sarà estratto un template.

Per il salvataggio dei dati dell'utente si è utilizzato un database risiedente su un server dove fosse presente un DBMS MySQL. Dal momento che si devono salvare solo informazioni relative a singoli soggetti (ovvero informazioni 1 a 1) il tutto è codificato all'interno del database *UserData* per mezzo di una singola tabella di nome *Clients*. I dati che vengono richiesti al momento della registrazione dell'utente sono:

- nome;
- cognome;
- numero di telefono;
- indirizzo e-mail;
- PIN;

Vengono generate una chiave pubblica ed una privata, da utilizzare per funzioni crittografiche a chiave asimmetrica, ed un'altra chiave privata da utilizzare invece per funzioni crittografiche a chiave simmetrica. Viene poi richiesto di rilevare l'impronta digitale (che può essere ad esempio sempre l'indice destro) e da quest'ultima si estrae un pattern che viene memorizzato all'interno del database.

4.3.2 Demo

In questa demo si fa uso dello smartphone come repository di dati, non solo per quanto riguarda il template biometrico ma anche per i dati identificativi del cliente e per le sue chiavi pubbliche e private. A livello generale questa demo permette di eseguire un'autenticazione biometrica di un utente al sistema: l'utente avvia sul proprio telefonino l'applicazione la quale, dopo aver richiesto a video l'inserimento del PIN personale dell'utente per l'accesso al sistema, crea un agente mobile sulla piattaforma nell'host client. A questo punto l'utente può richiedere alla postazione client, dove è presente lo scanner per le impronte, di iniziare l'autenticazione. Il server richiederà in automatico al telefonino di fornirgli i dati dell'utente per confrontarli con quelli presenti nel database MySQL e, se trovata una corrispondenza, verrà richiesto l'invio del template dell'utente e di quello live ricavato tramite lo scanner nella postazione client. A questo punto il server può comunicare l'esito dell'autenticazione.

Le operazioni step-by-step eseguite dal programma sono le seguenti:

1. nella postazione server è presente un'applicazione che attende richieste da parte di una applicazione client;

2. nella postazione client sono presenti lo scanner biometrico e un'applicazione che, per mezzo di una GUI, permette di richiedere l'avvio del processo di autenticazione (alla fine mostrerà il risultato del processo);

3. un utente che voglia avviare il processo di autenticazione deve avviare sul proprio smartphone, presso la postazione client, l'apposita applicazione. In primo luogo viene richiesto un PIN numerico di 5 cifre (che verrà utilizzato più avanti) e poi si collega al server JADE, crea un container speciale (di nome "User Phone Agent-"+ indirizzo logico server JADE + "-" + numero progressivo) e avvia un agente al suo interno di nome "User Phone Agent". A questo punto l'utente può richiedere l'avvio del processo di autenticazione biometrica cliccando sul pulsante dell'applicazione client;

4. l'applicazione client genera una richiesta di avvio del processo di autenticazione all'applicazione server;

5. appena giunta una richiesta viene instaurata una connessione SSL[1] tra le parti;

6. l'applicazione client genera un numero casuale (ID) e lo invia al server. Genera un container all'interno dello JADE Server con il nome "CLIENT-" + ID);

7. l'applicazione server, una volta ricevuto l'ID, genera un container all'interno dello JADE Server con il nome "SERVER-" + ID ed avvia un agente mobile al suo interno di nome "Biometric_User_Authenticator-" + ID;

8. l'agente del server crea un messaggio e lo salva all'interno di un Message-Container: quest'ultimo è un oggetto che può contenere un messaggio (come array di byte) detto *payload* ed un *challenge* (un valore float). La creazione del messaggio in questa fase iniziale prevede solo la generazione di un numero casuale come nuovo challenge e payload nullo;

9. l'agente cifra con la chiave AES del server il messaggio e poi lo firma digitalmente con la propria chiave privata;

[1]protocollo crittografico che permette una comunicazione sicura e una integrità dei dati su reti TCP/IP

10. viene stanziato un agente mobile dal nome “Mobile_Agent-” + n + “-” + ID (dove n è un numero progressivo che parte da 0 e viene incrementato ogni qual volta viene creato un agente) il quale prende il messaggio generato dal server, migra nel container del client ed invia il messaggio all’agente nel telefonino;

11. l’agente all’interno del telefono riceve il messaggio, ne controlla la firma digitale usando la chiave pubblica del server (nel caso in cui non fosse valida manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento), decifra il messaggio usando la chiave AES del server, aggiorna il challenge (sommando 1 al precedente), inserisce come payload del messaggio la propria chiave pubblica, cifra il messaggio con la chiave AES del server, ne calcola la firma digitale con la propria chiave privata e invia il messaggio all’agente mobile nel container client;

12. quest’ultimo ritorna al server e consegna il messaggio dopodiché termina;

13. l’agente nel server riceve il messaggio dall’agente mobile lo decifra con la chiave AES del server ed estrae la chiave pubblica dell’utente (che è salvata come payload del messaggio). A questo punto può controllare la firma digitale del messaggio usando appunto la chiave pubblica dell’utente (nel caso in cui non fosse valida manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento) dopodiché cifra la chiave pubblica del client con la chiave AES del server e la salva in memoria (questo per non lasciare in memoria dati sensibili in chiaro). Estrae il challenge, ne valuta la correttezza controllando che sia uguale al valore del challenge precedente più 1 (nel caso in cui non fosse valido manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento) ed infine lo aggiorna (sommando uno). Ora azzera il payload, cifra il messaggio con la chiave AES del server e poi lo firma digitalmente con la propria chiave privata. Stanzia un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest’ultimo;

14. il nuovo agente mobile migra nel container client ed invia il messaggio all’agente nel telefonino;

15. l’agente nel telefono riceve il messaggio ne controlla la firma digitale usando la chiave pubblica del client (nel caso in cui non fosse valida manda un messaggio di fallimento all’agente mobile che fa terminare l’intero processo di autenticazione segnalando l’avvenuto fallimento), decifra il messaggio con la chiave AES del server, estrae il challenge e lo aggiorna, imposta come

payload del messaggio il proprio valore di hash (presente nella SDCard dello smartphone sotto forma di file) ed il PIN inserito all'avvio dell'agente. Cifra il messaggio con la chiave AES del server e poi lo firma digitalmente con la propria chiave privata. Alla fine lo invia all'agente mobile in attesa nel container client;

16. quest'ultimo ritorna al server e consegna il messaggio dopodiché termina;

17. l'agente nel server riceve il messaggio dall'agente mobile, decifra usando la chiave AES del server la chiave pubblica dell'utente che aveva salvato cifrata in memoria e la usa per verificare la firma digitale del messaggio (nel caso in cui non fosse valida manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento). Decifra il messaggio con la chiave AES del server, effettua un controllo sul valore del challenge (nel caso in cui non fosse valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento) e poi lo aggiorna. Estrae il digest[2] ed il PIN, si collega al server MySQL e verifica la presenza dell'utente mediante l'hash (nel caso in cui non fosse un utente valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento). Controlla che il PIN nel database sia uguale a quello ricevuto (nel caso in cui non fosse un utente valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento) e in caso affermativo richiede al database anche la chiave AES dell'utente. Azzera il payload del messaggio, lo cifra con la chiave AES dell'utente (poi la cifra con la chiave AES del server e la salva in memoria) e poi lo firma digitalmente con la propria chiave privata. Stanzia un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest'ultimo;

18. il nuovo agente mobile migra nel container client ed invia il messaggio all'agente nel telefonino;

19. l'agente nel telefono riceve il messaggio ne controlla la firma digitale usando la chiave pubblica del client, decifra il messaggio con la propria chiave AES, estrae ed aggiorna il challenge, imposta come payload del messaggio il proprio valore di fingerprint (presente nella SDCard dello smartphone sotto forma di file) cifra il messaggio con la propria chiave AES e poi lo firma digitalmente con la propria chiave privata. Alla fine lo invia all'agente mobile in attesa nel container client;

[2] calcolo di una particolare firma del dato

20. quest'ultimo ritorna al server e consegna il messaggio dopodiché termina;

21. l'agente nel server riceve il messaggio dall'agente mobile, decifra usando la chiave AES del server la chiave pubblica dell'utente che aveva salvato cifrata in memoria e la usa per verificare la firma digitale del messaggio (nel caso in cui non fosse valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento). Decifra il messaggio con la chiave AES dell'utente ed effettua un controllo sul valore del challenge (nel caso in cui non fosse valido manda un messaggio di fallimento all'agente mobile che fa terminare l'intero processo di autenticazione segnalando l'avvenuto fallimento) e poi lo aggiorna, azzera il payload e salva il messaggio in memoria. Estrae il fingerprint e lo salva direttamente in memoria (non serve cifrarlo in quanto già cifrato) dopodiché crea un nuovo messaggio (quindi anche un nuovo challenge) il cui payload è la chiave pubblica del server, crea un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest'ultimo;

22. l'agente mobile migra nel client ed ottiene il live fingerprint dell'utente tramite il lettore biometrico, lo cifra usando la chiave pubblica del server (che era salvata come payload del messaggio che portava con sé) e poi ritorna al server consegnando il messaggio al server. A questo punto termina;

23. l'agente nel server decifra il live fingerprint usando la propria chiave privata, decifra usando la chiave AES del server la chiave AES dell'utente salvata in memoria, la usa per decifrare il fingerprint dell'utente salvato in memoria e poi effettua l'operazione di match. Estrae il messaggio dalla memoria (salvato al punto 21) e imposta come payload il risultato del match. Cifra il messaggio con la chiave AES dell'utente e lo firma digitalmente con la propria chiave privata. Stanzia un nuovo agente mobile (il cui nome avrà il contatore aumentato di 1 rispetto al valore precedente) ed invia il messaggio a quest'ultimo. Invia la risposta all'applicazione server la quale tramite la connessione SSL la invierà all'applicazione client. A questo punto l'agente server può terminare ed anche il container viene eliminato;

24. l'agente mobile migra nel container client, invia il messaggio all'agente nel telefonino dopodiché termina ed il container client viene eliminato;

25. l'agente all'interno del telefono riceve il messaggio, ne verifica la firma digitale (in caso di errore avvisa l'utente) con la chiave pubblica del server, lo decodifica con la propria chiave AES e stampa a video il risultato dell'autenticazione. A questo punto può terminare;

4.3.3 Considerazioni sulle scelte implementative

Per questa dimostrazione, dal momento che fa uso di un dispositivo mobile dotato di una potenza di calcolo limitata, si è cercato di limitare al massimo l'utilizzo della crittografia a chiave pubblica preferendole quella a chiave privata, la quale permette di ottenere un ottimo livello di sicurezza con un costo computazionale notevolmente inferiore. Come si deduce infatti dalla lettura delle varie operazioni, la crittografia asimmetrica trova utilizzo solamente nell'instaurazione della comunicazione SSL tra il client ed il server, che sono macchine desktop dotate di potenza di calcolo sufficiente allo scopo, e per la firma digitale dei messaggi, dove si deve codificare solamente un digest di pochi byte. Per le altre operazioni di codifica, invece, si utilizza l'algoritmo crittografico a chiave simmetrica che prende il nome di AES. Le operazioni coinvolte sono:

- *connessione SSL*: operazioni 4, 5, 6 per l'instaurazione della connessione e per l'invio dell'ID e operazione 23 per l'invio dei risultati dell'autenticazione. Vengono svolte solamente da macchine desktop;
- *firma digitale dei messaggi e relativa verifica della loro validità*: operazioni 9, 11, 13, 15, 17, 19, 23, 25 dove alcune di esse vengono eseguite dal dispositivo mobile ma grazie alla limitata dimensione dei dati da codificare risultano essere estremamente veloci;
- *cifratura messaggio con chiave pubblica*: operazione 22 eseguita solamente su macchina desktop;
- *cifratura con chiave simmetrica*: stesse operazioni relative alla firma digitale dei messaggi. Ovviamente anche in questo caso alcune di esse vengono eseguite sul dispositivo mobile ma grazie a delle librerie opportunamente ottimizzate si ottengono tempi di esecuzione paragonabili a quelli misurabili su macchine desktop;

Dai test effettuati, grazie agli accorgimenti presi in fase di progettazione in riferimento all'uso degli opportuni algoritmi crittografici, una sessione di autenticazione viene eseguita in un tempo massimo di 5 secondi.

Capitolo 5

Manuale Utente

L'obiettivo di questo capitolo è quello di fornire al lettore una guida da seguire nel caso in cui si voglia testare personalmente il software oggetto di questa tesi. Verranno quindi illustrate le corrette procedure di installazione dei vari programmi necessari e le modifiche da apportare ai vari script scritti appositamente per lanciare in modo rapido le varie applicazioni.

Si suppone che l'utente abbia già installato sul proprio computer una distribuzione Linux, come Ubuntu o Fedora. Il progetto è stato sviluppato e testato su Ubuntu 10.04 LTS 64-bit, tuttavia l'esecuzione del software non è vincolata a questa particolare distribuzione.

5.1 Installazione del software necessario

La prima cosa da fare è predisporre il sistema ovvero è necessario installare tutti i software di terze parti e le librerie utilizzate dalle varie dimostrazioni:

- *Java Development Kit v1.6 o superiore*: l'installazione non risulta particolarmente problematica in quanto è sufficiente scaricare dal sito della Sun il relativo pacchetto e seguire le istruzioni proposte a video;

- *Java Advanced Image*: una volta scaricato il pacchetto dal sito della Sun si dovrà entrare nella cartella ove risiedente la JVM (su Ubuntu dopo installazione JDK 1.6 /usr/lib/jvm/java-6-openjdk/) e da lì lanciare l'eseguibile che provvederà all'installazione della libreria;

- *Smartphone Android*: per poter utilizzare l'applicazione sul proprio smartphone è sufficiente installare il file *BiometricAuthenticationSystem.apk* seguendo il manuale d'uso del telefono. Nel caso invece si voglia utilizzare l'emulatore, è sufficiente scaricare ed installare Android SDK e la piattaforma Android v2.1

o successiva. L'installazione di queste due componenti risulta molto semplice ed intuitiva. Dopodiché sarà possibile installare l'applicazione sull'emulatore grazie al tool *adb*;

- *Server MySQL*: scaricabile dal sito ufficiale l'installazione non presenta particolari problemi. Una volta installato è necessario importare il database fornito assieme al programma. Utilizzando *MySQL Administrator* sarà sufficiente collegarsi al database come utente root e poi, cliccando su *Restore Backup*, scegliere il file che contiene le istruzioni SQL per il ripristino del database;

- *JADE*: è sufficiente copiare nell'home dell'utente tutta la cartella fornita assieme al programma. Al suo interno saranno presenti tutte le librerie, in formato JAR, necessarie per il suo utilizzo e che saranno automaticamente aggiunte nel *CLASSPATH* dagli script forniti. Si ricorda inoltre che è necessario scaricare anche gli add-on *JADE-Leap* e *JADE-Android*. Entrambi andranno poi copiate all'interno della cartella JADE;

- *Driver Biometrika*: è da premettere che per l'installazione dei driver è necessario disporre dei sorgenti del kernel in uso. Una volta disponibili i sorgenti si potrà utilizzare la funzione *build.pl* (presente all'interno della cartella dei driver) per procedere alla compilazione dei driver. Una volta compilato il modulo dovrà essere installato (si dovrà avere l'accesso root al sistema) tramite la funzione *fxdriverinstall*, alla quale dovrà essere fornito il suo percorso. Una volta installato il modulo si dovrà copiare nella cartella /etc/ di sistema la cartella e tutte le sottocartelle presenti in etc/ all'interno della cartella dei driver. Fatto questo si potrà procedere all'avvio del modulo tramite la funzione *rc/fx2000.init.udev start*. Una volta caricato il modulo (si può verificare l'avvenuto caricamento tramite il comando dmesg) si dovrà avviare lo scanner tramite i seguenti comandi: *bin/FxReset -p 0* e *bin/FxReset -i 0*. L'avvio del modulo e l'inizializzazione dello scanner sono operazioni che devono essere ripetute ad ogni avvio del sistema (si può quindi inserire almeno il caricamento del modulo nel file `etc/rc.local` per farlo in automatico);

A questo punto sono stati svolti tutti i passi preliminari e il sistema è pronto per l'uso.

5.2 Enrollment Tool

Lo script di avvio di tale software si trova nella sotto-cartella *SCRIPT* della cartella *ENROLLMENT* e ha nome *ENROLLMENT.start*. Per poter avviare correttamente il programma sarà necessario modificare solamente la variabile *HOME* inserendo l'indirizzo completo della propria directory home tra apici.

Il software, una volta avviato, mostra la propria interfaccia grafica. La prima operazione da compiere consiste nell'impostazione dei parametri per l'accesso al server MySQL: cliccando sul menù a tendina *MySQL -> Change connection parameters* apparirà una nuova finestra nella quale inserire i dati relativi alla connessione (cliccando su *Load* verranno caricati quelli di default); una volta inseriti cliccando su *Save* verranno memorizzati su un file e d'ora in poi utilizzati per connettersi al DBMS. Una volta completato il passo di configurazione si potrà avere accesso ai dati del database (ovviamente nel DBMS devono essere impostati i diritti di acceso e/o modifica relativi all'utente se non questi non è root) e di conseguenza si potranno compiere le seguenti operazioni:

- aggiunta di un utente: dopo aver inserito i dati in forma testuale nelle relative caselle, si procederà all'inserimento del template biometrico utilizzando come sorgente una immagine `.tif` (nel qual caso si utilizzerà il pulsante *Load Fingerprint Image from File*). La finestra di log mostrerà le operazioni compiute. Per ottenere una scansione da utilizzare come sorgente per l'enrollment è sufficiente aprire un terminale e, dopo aver inizializzato lo scanner ed aver appoggiato il dito sulla parte sensibile dello scanner stesso, digitare il comando *bin/FxReset -a fingerprint.tif*;

- visionare la lista degli utenti presenti nel database: cliccando sul pulsante *List* apparirà un'altra finestra con una tabella contenente una lista di tutti i record presenti nel database. Una volta selezionata una tupla, cliccando sul pulsante *Load selected data on main menu* i dati relativi all'utente selezionato verranno automaticamente caricati (ad esclusione dell'immagine del parametro biometrico che non viene salvata nel database ma viene salvato solo il template) e potranno essere editati oppure cancellati (per cancellazione si intende l'eliminazione completa del record dal database). Per l'aggiunta di un nuovo utente sarà necessario cliccare sul menù a tendina *File -> New*;

- editare un utente presente nel database: dopo aver caricato i dati relativi dell'utente prescelto sarà sufficiente editare i valori presenti nelle caselle di testo e poi cliccare sul pulsante *Edit* per salvare i cambiamenti. Si fa notare che non è possibile sostituire il parametro biometrico. Per l'aggiunta di un nuovo utente sarà necessario cliccare sul menù a tendina *File -> New*;

- eliminare un utente presente in lista: una volta caricato i dati di un utente cliccando su *Remove* si procederà alla sua eliminazione dal database;

Accanto alle funzioni necessarie ad inserire, editare e cancellare utenti all'interno del database, sono presenti anche metodi per:

- estrazione della chiave AES: cliccando sul pulsante *Extract AES Key* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che

conterrà al suo interno la chiave privata dell'utente utilizzata nella cifratura simmetrica;

- estrazione del template biometrico: cliccando sul pulsante *Extract Fingerprint Pattern* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno il template biometrico cifrato con la chiave AES dell'utente;
- estrazione chiave pubblica RSA: cliccando sul pulsante *Extract RSA Public Key* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno la chiave pubblica dell'utente utilizzata nella cifratura asimmetrica;
- estrazione chiave privata RSA: cliccando sul pulsante *Extract RSA Private Key* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno la chiave privata dell'utente utilizzata nella cifratura asimmetrica;
- estrazione del digest dei dati dell'utente: cliccando sul pulsante *Extract HASH* apparirà una finestra che permetterà di scegliere dove e con che nome salvare il file che conterrà al suo interno il valore del digest dei dati dell'utente (il digest SHA-256);
- preparazione dei dati da salvare nella memoria del telefono: cliccando sul pulsante *Prepare data for PHONE* apparirà una finestra che permetterà di scegliere la cartella ove salvare i dati che dovranno essere copiati all'interno della memoria del telefonino dell'utente affinché possa autenticarsi(si ricorda che devono essere copiate anche la chiave pubblica del server e la sua chiave AES);

5.3 Biometric Authentication System

Prima di procedere all'avvio della dimostrazione è necessario compiere alcuni operazioni preliminari, ovvero modificare i seguenti file:

All'interno della cartella *CLIENT/bua*:

- *Address.dat*: vanno inseriti l'indirizzo IP della macchina su cui è in esecuzione il server (può essere la stessa su cui è in esecuzione il client), e l'indirizzo IP del server JADE;

All'interno della cartella *SERVER/appListener*:

- *Address.dat*: vanno inseriti l'indirizzo IP della macchina su cui è in esecuzione il server, e l'indirizzo IP del server JADE;

All'interno della cartella *SERVER/biometricUserAuthenticator*:

- *KEYPATH*: qui vanno inseriti i percorsi completi delle chiavi pubbliche e private RSA del server (generabili con il tool fornito *RSA Key Generator*), della chiave pubblica dell'host e della chiave AES del server (generabile con il tool fornito *AES Utility*);
- *MySQLParameters*: dove si inseriscono i dati relativi alla connessione al DBMS MySQL;
- *CLIENTCONTAINERDATA*: all'interno del quale va inserito l'indirizzo IP della macchina su cui è in esecuzione il client (può essere la stessa macchina su cui è in esecuzione il server);

Per quanto riguarda il telefono è necessario copiarvi i dati creati al momento della registrazione dell'utente. Per fare questo basta aprire l'Enrollment Tool, caricare l'utente che si intende autenticare e, dopo aver collegato il telefono al pc tramite usb, esportare i suoi dati (usando il comando *Prepare data for PHONE*) all'interno del telefono. Nel caso in cui si utilizzi l'emulatore questa operazione eseguita tramite il comando *push* del tool *adb*.

Si dovranno poi aggiungere manualmente, sempre all'interno della memoria, i dati relativi al server ovvero la sua chiave privata AES (che deve essere nominata *SERVERKEY.AES*) e la sua chiave pubblica RSA (che deve essere nominata *SERVERPublicKey.dat*).

Ora non resta che modificare gli script di avvio sostituendo in tutti e tre la variabile HOME inserendo l'indirizzo completo della propria directory home. Nello script *JADE.start* va inoltre modificato il valore associato all'opzione *-host*, inserendo l'indirizzo IP del computer su cui verrà eseguito lo script.

Ovviamente la demo può essere eseguita anche su una singola macchina in locale. In questo caso, nei file precedentemente elencati, i valori dei vari indirizzi IP coincideranno con l'indirizzo locale (memorizzato nel file `/etc/hosts`).

A questo punto si possono avviare gli script in questo ordine:

- *JADE.start*: per avviare la piattaforma contenente il main container;
- *SERVER.start*: per avviare il server;
- *CLIENT.start*: per avviare il client;

Ora è possibile avviare l'applicazione sullo smartphone (o, in alternativa, sull'emulatore).

L'utilizzo della demo è estremamente semplice: prima di tutto, dopo aver cliccato il pulsante *Launch* sul display del telefono e aver selezionato l'indirizzo IP del JADE Server, si inserisce il PIN dell'utente che richiede l'autenticazione e

si clicca su *Send PIN*. Nel caso in cui il telefono in uso sia fornito di touchscreen, basterà toccare due volte all'interno della finestra di inserimento del PIN per visualizzare la tastiera virtuale. Ora non resta che appoggiare il dito sullo scanner biometrico e cliccare su *Start Biometric User Authentication* nell'applicazione client. Se i dati presenti all'interno del telefonino non sono corretti (ovvero se l'hash ed il PIN forniti coincidono con quelli di un utente presenti nel database) il sistema lo comunicherà all'utente che potrà effettuare un nuovo tentativo. Se invece essi dovessero rivelarsi corretti, dopo aver effettuato la fase di matching dell'impronta, il sistema visualizzerà il risultato dell'autenticazione sia nella finestra apposita dell'applicazione client sia sul telefonino. A questo punto si potrà eseguire un'altra autenticazione cliccando sulla freccia indietro dello smartphone. Se si riscontrassero problemi di comunicazione tra i container JADE è necessario verificare che il file `/etc/hosts` contenga:

- indirizzo locale: nella forma 127.0.0.1 *localhost*;
- indirizzo di rete: nella forma IP *networkname machinename*, come ad esempio 147.26.53.192 bio2.dei.unipd.it;

Capitolo 6

Manuale Tecnico

Dopo aver descritto il funzionamento generale del sistema e aver fornito gli strumenti necessari per testare personalmente il sistema, in questa sezione verranno presentate le modalità di implementazione dei vari software riportando, quando opportuno, alcuni pezzi di codice che possano aiutare a comprenderne meglio la struttura e a favorirne lo sviluppo.

Tutto il progetto è stato sviluppato in linguaggio Java. Sia la fase di sviluppo sia la fase di testing sono state condotte sia su un computer desktop con sistema operativo Linux[1], sia su macchine collegate in rete in quanto tutti i parametri necessari ad un suo funzionamento, come esposto nelle sezioni successive, sono letti da file di configurazione tranquillamente editabili; per la parte del telefonino si è utilizzato uno smartphone HTC Desire con sistema operativo Android 2.1 e come ambiente di sviluppo si è utilizzato Eclipse. Accanto al sistema sono state prodotte due librerie, *Utility* e *Utility Android*, che raccolgono le funzioni più utilizzate dai vari programmi come ad esempio l'accesso ai file, la lettura dei parametri di configurazione e la scrittura/lettura delle chiavi di cifratura.

Nella nostra demo, il software di autenticazione, come si è già avuto modo di constatare, è caratterizzato, oltre che da una piattaforma ad agenti mobili, anche da tre applicazioni:

- lato smartphone, con il nome di Biometric Authentication System;
- lato client, con il nome di Biometric User Authentication;
- lato server, che prende il nome di Application Listener;

[1] distribuzione Ubuntu 10.04

6.1 Biometric Authentication System

Come già illustrato nei capitoli precedenti, il JADE runtime disponibile per Android è una versione molto semplificata rispetto a quella che può essere eseguita su un computer desktop e il suo unico scopo è quello di collegarsi ad un host, a sua volta collegato al main container della piattaforma, e lì creare il *BackEnd* container il quale si carica del compito della registrazione presso il main container e, allo stesso tempo, creare il *FrontEnd* container sullo smartphone.

L'indirizzo di rete della postazione host in cui verrà creato il *BackEnd* potrà essere scelto e selezionato da una lista che verrà visualizzata sul display del telefono. Per l'avvio del runtime, JADE-Android prevede una propria procedura completamente diversa da quella utilizzata da JADE-Leap.I passi principali da seguire sono:

1. creare una classe Jade Agent che estenda *GatewayAgent*;
2. all'interno della classe creata al punto precedente sovrascrivere il metodo *processCommand*;
3. creare un'activity che implementi *ConnectionListener*;
4. chiamare, all'interno del metodo *onCreate* dell'Activity, il metodo JadeGateway.connect, passandogli, attraverso un oggetto di tipo *Properties*, tutti i parametri necessari per la connessione;
5. implementare il metodo onConnected(JadeGateway gateway) dell'interfaccia *ConnectionListener* per creare un'istanza dell'oggetto GatewayAgent;
6. chiamare il metodo *execute* dell'oggetto di tipo *JadeGateway* per mandare un comando all'agente;
7. chiamare il metodo *disconnect* per terminare la connessione;

Listing 6.1: Avvio ambiente JADE su telefonino

```
1  public class Start extends Activity implements ConnectionListener {
2
3          private JadeGateway gateway;
4          private BridgeJadeAndroid updater;
5
6          @Override
7      public void onCreate(Bundle savedInstanceState) {
8          super.onCreate(savedInstanceState);
9          setContentView(R.layout.start);
10
```

```
11          updater = new BridgeJadeAndroid(this);
12
13          Properties props = new Properties();
14          props.setProperty(Profile.MAIN_HOST,
15                  UserPIN.getSelectedIP());
16          props.setProperty(Profile.MAIN_PORT, getResources().
17                                  getString(R.string.port));
18          props.setProperty(JICPProtocol.MSISDN_KEY,
19                                  getResources().
20                                  getString(R.string.msisdn));
21
22          try {
23                  JadeGateway.connect(UserAuthenticatorAgent.
24                          class.getName(),
25                          null, props, this, this);
26                  }
27          catch (Exception e) {
28                  Toast.makeText(this, e.getMessage(), 5000);
29                  }
30      }
31
32          public void onConnected(JadeGateway gw) {
33                  gateway = gw;
34
35                  try {
36                          gateway.execute(updater);
37                          }
38                  catch (StaleProxyException e) {
39                          e.printStackTrace();
40                          }
41                  catch (ControllerException e) {
42                          e.printStackTrace();
43                          }
44                  catch (InterruptedException e) {
45                          e.printStackTrace();
46                          }
47                  catch (Exception e) {
48                          e.printStackTrace();
49                          }
50          }
51
52          public void onDisconnected() {
53
54          }
55
56          // Chiusura della connessione con JADE
57          protected void onDestroy() {
58                  super.onDestroy();
59
```

```
60                  try {
61                          if (gateway != null)
62                                  gateway.shutdownJADE();
63
64                  } catch (ConnectException e) {
65
66                          Toast.makeText(this, e.getMessage(),
67                                  Toast.LENGTH_LONG);
68                  }
69                          if (gateway != null)
70                                  gateway.disconnect(this);
71          }
72  }
```

6.2 Biometric User Authentication

Tale applicazione ha lo scopo di presentare una GUI per instaurare una connessione SSL con l'applicazione server, generare l'ID di sessione ed avviare l'agente mobile client. Per conoscere quale sia l'indirizzo del server con il quale richiedere l'avvio della connessione protetta il software ne legge l'indirizzo su un file, editabile con i propri dati, di nome `Address.dat`. Per la generazione della connessione sicura poi necessita del percorso dei certificati che contengono le chiavi pubbliche e private delle due entità: tali percorsi vengono letti da un file di nome `DataCertificates.dat`.

Listing 6.2: Richiesta di connessione al server e connession SSL

```
1  String [] data=new String [2]; //data=Address + Port Number
2  data[0]=utility.FileUtility.getDataFromDataFile
3          (AddressFile, "ADDRESS");
4  data[1]=utility.FileUtility.getDataFromDataFile
5          (AddressFile,"PORT");
6  socket=utility.NetUtility.getConnection
7          (utility.NetUtility.getAddress(data[0]),
8          Integer.parseInt(data[1])); //get connection
9
10 //Read ACK from Application Listener
11 if(!utility.NetUtility.readFromSocket(socket).equals("ACK")) {
12         return("An error occured. I can't bind my application to
13                 Server" +"\nTry Again...");
14     }
15
16 //Read port of SSL Connection from socket
17 int portSSLConnection=
18         Integer.parseInt(utility.NetUtility.readFromSocket(socket));
19
```

```
20 socket.close(); //close clear connection
21
22 //Create an SSL Connection with Proxy
23 //Validate trustStoreServer
24 while(sslSocket==null) {
25         sslSocket=utility.NetUtility.getClientSSLSocket(
26                         utility.FileUtility.getDataFromDataFile
27                         (DataCertificatesFile, "CERTIFICATES_PATH"),
28                         utility.FileUtility.getDataFromDataFile
29                         (DataCertificatesFile, "CLIENT_CERTIFICATE_NAME"),
30                         utility.FileUtility.getDataFromDataFile
31                         (DataCertificatesFile, "SERVER_CERTIFICATE_NAME"),
32                         utility.FileUtility.getDataFromDataFile
33                         (DataCertificatesFile, "CLIENT_PASSWORD"),
34                         utility.FileUtility.getDataFromDataFile
35                         (DataCertificatesFile, "SERVER_PASSWORD"),
36                         utility.FileUtility.getDataFromDataFile
37                         (AddressFile, "ADDRESS"),
38                         portSSLConnection );
39         }
```

L'avvio dell'agente mobile sfrutta le API messe a disposizione dalla piattaforma JADE. Prima di tutto è necessario creare un oggetto *Runtime* che permetta di accedere all'ambiente runtime JADE; poi si deve creare un profilo attraverso la creazione dell'oggetto *Profile* che permette di impostare tutti i parametri necessari al corretto avvio dell'agente come l'indirizzo della piattaforma (anch'esso reso disponibile all'interno del file `Address.dat`), il nome del container che deve essere creato, i servizi che devono essere caricati, ecc.... Alla fine, tramite l'oggetto *Runtime* si possono creare il container ed avviare l'agente al suo interno.

Listing 6.3: Avvio dell'agente su piattaforma JADE

```
1  //Create CLIENT Agent in JADE Platform
2  //Parameters for CLIENT
3  Runtime runtime=Runtime.instance(); //get runtime environment
4  Profile pClient=new ProfileImpl(); //make new profile for client
5  pClient.setParameter(Profile.MAIN_HOST,
6          utility.FileUtility.getDataFromDataFile(AddressFile,
7          "JADE_HOST")); //set JADE Host
8  pClient.setParameter(Profile.MAIN_PORT,
9          utility.FileUtility.getDataFromDataFile(AddressFile,
10         "JADE_PORT")); //set JADE Port
11 pClient.setParameter(Profile.CONTAINER_NAME, "CLIENT-" +ID);
12 ccClient=runtime.createAgentContainer(pClient);
```

Una volta avviato l'agente l'applicazione rimane in attesa del risultato dell'autenticazione.

6.3 Application Listener

Tale applicazione risiede sul server e ha il compito di ricevere le richieste da parte delle applicazioni client e di instaurare, con ognuna di esse, una connessione protetta SSL (i percorsi dei certificati sono presenti su un file di testo del tutto simile a quello cui ha accesso l'applicazione client). Una volta creato il collegamento sicuro viene stanziato un oggetto di tipo *Proxy* il quale ha il compito di avviare il container server e i relativi agenti mobili al suo interno (le modalità sono del tutto simili a quelle descritte in ambito client) e di inviare il risultato dell'autenticazione non appena è a disposizione. Al momento non è previsto che l'*ApplicationListener* crei un thread specifico per ogni connessione, quindi è ammissibile una sola connessione per volta: non appena viene restituito il risultato è possibile effettuare un'altra sessione di autenticazione.

6.4 Agenti Mobili

La parte principale della dimostrazione consiste nell'utilizzo degli agenti mobili. La programmazione di essi in ambito JADE prevede di strutturare il loro set di operazioni come una insieme di comportamenti che possono essere eseguiti nell'ordine voluto. In fase di progettazione si è scelto di strutturare l'insieme complessivo delle operazioni dell'agente come una sequenza di comportamenti, dove ogni singolo comportamento è stato implementato come classe al fine di poter riutilizzare parte del codice nelle occasioni in cui un comportamento dovesse essere utilizzato più volte.

6.4.1 BiometricUserAuthenticator

L'agente principale risiede sul server e presenta questa sequenza di comportamenti:

- *richiesta dell'hash dell'utente*: prima di tutto viene creato un oggetto di tipo *MessageContainer* formato da un array di byte che serviranno a contenere il messaggio da trasmettere, e da un'istanza dell'oggetto *Challenge* che servirà a memorizzare il valore di un challenge generato dal server che, assieme alla firma digitale, servirà come mezzo per verificare l'autenticità delle risposte ricevute. Una volta stanziato tale oggetto (al cui interno quindi è presente un nuovo challenge e una richiesta per l'hash dell'utente) viene calcolata la sua firma digitale (ovvero viene calcolato un digest che poi viene cifrato con la chiave privata del server), viene cifrato con la chiave AES del server ed il tutto viene passato come parametro ad un nuovo agente mobile, di nome *Mobile_ Agent*, il quale migra sul container client (il cui indirizzo viene

ritrovato tramite una funzione scritta ad hoc presente sulla libreria Utility che interroga l'AMS[2]) e successivamente invia il messaggio all'agente dello smartphone. L'agente legge le chiavi pubbliche e private del server da un file di testo, editabile a piacimento, di nome `KEY_PATH`;

- *attesa di una risposta*: l'agente attende che *Mobile_ Agent* ritorni sul server con le relative risposte;

- *controllo del challenge*: *Mobile_ Agent* è ora ritornato sul server e ha riportato il messaggio modificato dall'agente del telefono; quest'ultimo viene decifrato, ne viene verificata la firma digitale e poi viene controllato il valore del challenge. Se non si sono verificati errori il challenge viene aggiornato in caso contrario viene attivato il comportamento di autodistruzione e l'autenticazione termina con risultato negativo;

- *controllo della presenza dell'utente nel database*: una volta ricevuto l'hash dell'utente si crea una connessione al DBMS MySQL, e lo si interroga al fine di verificare la presenza dell'utente e, se esiste, si prosegue, altrimenti si richiama il comportamento di autodistruzione e l'autenticazione termina con risultato negativo;

- *invia Mobile_ Agent a rilevare il live fingerprint*: viene inviato al client l'agente mobile affinché rilevi il template biometrico live dell'utente;

- *attesa di una risposta*: l'agente attende che *Mobile_ Agent* ritorni sul server con le relative risposte;

- *esecuzione del match e terminazione*: una volta decifrato il contenuto del messaggio, si interroga nuovamente il database per estrarre il template dell'utente memorizzato, si stanzia la classe *FindMatch* e tramite il metodo *getTemplatesFromByteArray(byte firstT, byte secondT)*, si caricano i due pattern (quello live e quello memorizzato); infine tramite una chiamata alla funzione *matchTemplate()* si ricava uno score che se superiore alla soglia prefissata (in fase di progettazione si è deciso di scegliere 30 minuzie) permette di autenticare correttamente l'utente. A questo punto non resta che avvisare il proxy dell'avvenuta autenticazione e poi il tutto viene distrutto tramite il comportamento di autodistruzione;

[2]Agent Management System: componente che supervisiona gli accessi alla piattaforma da parte di entità esterne

Listing 6.4: Esecuzione del match ed invio risultato

```
1  //Get template from byte
2  FindMatch matcher=new FindMatch();
3
4  matcher.getTemplatesFromByteArray(userFingerPrint,
5          liveFingerPrint); //Get user name
6  String user=new String(utility.AES.BCdecode
7          (utility.AES.readKey(SERVER_AES_KEY),
8          (byte[])getDataStore().get("USER_NAME")));
9
10 //Calculate Match and print results of matching
11 //if I find more than 30 equal minutiae
12 if(matcher.matchTemplate()>30) {
13          String  message=new String("USER "+user+" AUTHENTICATED");
14          System.out.println("["+(String)getDataStore().get("MY_NAME")+
15                  "]:" +message);
16          //set message like payload of message container
17          mc.setMessage(message.getBytes());
18          //create a mobile agent that inform phone of result
19
20          BehaviourCreateMobileAgent cma=new BehaviourCreateMobileAgent
21                  (thisAgent,mc,USER_AES_KEY);
22          cma.setDataStore(this.getDataStore());
23          thisAgent.addBehaviour(cma);
24          //create a file with result
25          File result=new File("result.dat");
26          utility.FileUtility.writeFile(message.getBytes(),result);
27          //now I finished
28          return;
29          }
30 else { //NOT AUTHENTICATED
31          String  message=new String("USER "+user+" NOT AUTHENTICATED");
32          System.out.println("["+(String)getDataStore().get("MY_NAME")+
33                  "]:" +message);
34          //set message like payload of message container
35          mc.setMessage(message.getBytes());
36          //create a mobile agent that inform phone of result
37          BehaviourCreateMobileAgent cma=new BehaviourCreateMobileAgent
38                  (thisAgent,mc,USER_AES_KEY);
39          cma.setDataStore(this.getDataStore());
40          thisAgent.addBehaviour(cma);
41          //create a file with result
42      File result=new File("result.dat");
43      utility.FileUtility.writeFile(message.getBytes(),result);
44          //now I finished
45      return;
46      }
```

6.4.2 Mobile

Accanto all'agente che risiede sul server è presente un agente mobile che migra verso il client e che ha il compito di trasportare i messaggi dal server al telefono e di acquisire la live fingerprint. Esso ha un codice strutturato secondo una serie di comportamenti di cui, i più importanti, sono quelli relativi alla migrazione, in cui è necessario conoscere il container di destinazione rappresentato da un AID, e alla ricezione ed invio di messaggi ACL verso il telefono che, essendo eseguito in modalità asincrona, sfrutta l'utilizzo di un buffer. È stato implementato anche un controllo sul nome del mittente del messaggio e sulla performative ad esso associata al fine di evitare attacchi del tipo mail bombing da utenti sconosciuti in quanto tutti i messaggi che non provengono da agenti facenti parte del sistema vengono scartati (notare che questa metodologia risulta efficace in quanto ad ogni sessione di autenticazione i nomi degli agenti contengono, nella parte finale, un numero casuale valido solo per quella sessione).

Listing 6.5: Mobilità dell'agente

```
1  //Migration of agent
2  //if I have a destination AID
3  if(destination!=null) {
4          thisAgent.doMove(destination);
5          }
6  else { //I have data of container
7          ContainerID destinationContainer=new ContainerID();
8      destinationContainer.setName(destinationContainerName);
9          destinationContainer.setAddress(destinationAddressContainer);
10     destinationContainer.setPort(destinationPortContainer);
11     thisAgent.doMove(destinationContainer);
12     }
```

Listing 6.6: Send e receive di messaggi ACL

```
1  //send message
2  if(send) {
3          this.sendMessage(); //cal function that send message
4          }
5  else { //receive message
6          ACLMessage incoming=thisAgent.receive();
7          //receive incoming message
8
9      if(incoming!=null) {
10         //Control that sender is correct sender
11         if((incoming.getSender().getLocalName().equals(senderName))
12                 && ((incoming.getPerformative()==ACLMessage.REQUEST)
13                  || (incoming.getPerformative()==ACLMessage.INFORM)
14                  || (incoming.getPerformative()==ACLMessage.FAILURE)))
```

```
15                          {
16                                  received=true;
17                  //CONTROL IF IT'S A FAILURE
18                  if(incoming.getPerformative()==ACLMessage.FAILURE) {
19                          //a failure is arrived
20                          //so send a message to server and
21                      //destroyed myself
22                      ACLMessage mex=new ACLMessage(ACLMessage.FAILURE);
23                      AID server=new AID();
24                      server.setLocalName("Biometric
25                          User Authenticator");
26                      mex.addReceiver(server);
27                      thisAgent.send(mex);
28                      System.out.println("[Mobile User Data Requester-"
29                          +(String)getDataStore().get("ID")+"]:" +
30                          "an error was occured.
31                          Authentication process " +
32                          "fail.");
33                      //Failure so I must kill agent
34                      thisAgent.addBehaviour
35                          (new BehaviourKillAgent(thisAgent));
36                      return;//terminate this behaviour
37                          }
38                                  //Save the message in DataStore
39                  this.getDataStore().put("MESSAGE",
40                          incoming.getContent());
41                          }
42                  else {
43                //I have receive the message so I can exit from
44                //blocking mode
45                received=false;
46                }
47                  }
48          else {
49                  //block agent until message arrive
50          this.block();
51          received=false;
52          }
53          }
```

6.4.3 UserPhoneAgent

Anche l'agente stanziato sul telefono è stato strutturato come un sequenza di comportamenti:

- *attesa del challenge da parte del server*: l'agente rimane in attesa che nel suo buffer arrivi un messaggio proveniente dal server contenente il challenge

da modificare e rinviare al server come primo metodo per dimostrare la sua fidatezza;

- *decodifica del messaggio e modifica del challenge*: una volta arrivato il messaggio ne viene verificata la firma digitale (ovviamente facendo terminare tutto il processo di autenticazione se questa non fosse valida), viene decifrato usando la chiave AES del server (che è salvata assieme agli altri dati nella memoria del telefono), si procede all'aggiornamento del challenge e all'impostazione della propria chiave pubblica come payload del messaggio,ed infine si codifica il MessageContainer e lo si firma digitalmente con la propria chiave privata, dopodiché lo si invia all'agente in attesa sul client (si noti che nel codice si fa molto ricorso alle librerie sviluppate, in questo caso UtilityAndroid);

Listing 6.7: Elaborazione del messaggio

```
1  //get message with signature
2  byte [] messageWithSignature=getMessageByteArray();
3  //get signature
4  byte [] signature=utility.Android.RSAAndroid.
5  extractSignatureFromMessageWithSignature
6          (messageWithSignature);
7  //get message
8  byte [] message=utility.Android.RSAAndroid.
9          extractMessageFromMessageWithSignature
10                 (messageWithSignature);
11 //verify signature of message if failure stop
12 //authentication process
13 if(!verifyServerSignature(signature,message)) {
14         printMessage(R.id.TextView02,"ERROR - Signature
15         is not ok.
16     \nMessage was mod by someone
17     or it was not send by Server.
18     \nAuthentication process will stop.");
19     thisAgent.addBehaviour
20     (new BehaviourShutDownAuthentication(thisAgent));
21     }
22
23 [...]
24
25 printMessage(R.id.TextView02,"SERVER is ready to start
26         authentication proccess");
27
28 //decode message
29 byte[] messageDecoded;
30 try {
31         messageDecoded = startAES(message, false, "SERVER");
32
```

```
33      //Set new message with payload = my public key
34      byte [] replyMessage=setNewMessage(messageDecoded,
35          getMyRSAKey(true));
36      //Crypt replyMessage with ServerKey
37      byte[] replyEncryptMessage;
38
39      replyEncryptMessage = startAES(replyMessage, true, "SERVER");
40
41      //Sign message
42      byte [] signReplyMessage=utility.Android.RSAAndroid.RSASign
43          (replyEncryptMessage,
44          utility.Android.RSAAndroid.readPrivateKey
45          (getMyRSAKey(false)));
46      String replyMessageHEX=utility.Android.HexUtilityAndroid.
47          toHex(signReplyMessage)+utility.Android.
48          HexUtilityAndroid.toHex(replyEncryptMessage);
49      //Send reply
50      ACLMessage reply=new ACLMessage(ACLMessage.INFORM);
51      AID destination=new AID();
52          destination.setLocalName((String)getDataStore().
53          get("MOBILE_AGENT_NAME"));
54      reply.addReceiver(destination);
55      reply.setContent(replyMessageHEX);
56      thisAgent.send(reply);
57      printMessage(R.id.TextView03,
58          "I sent reply that I'm also ready to start");
59      //update number of mobile agent
60      int value=Integer.parseInt((String)getDataStore().
61                              get("NUM_MOBILE_AGENT"));
62      getDataStore().put("NUM_MOBILE_AGENT",""+(++value));
63      //Update name of mobile agent
64      updateNameOfMobileAgent();
65
66          }
67  catch (FileNotFoundException e) {
68          // TODO Auto-generated catch block
69          e.printStackTrace();
70          }
71
72  return; //finish
```

- *attesa della risposta da parte del server*: l'agente rimane in attesa di un messaggio proveniente dall'agente mobile sul client che si fa carico di trasportare le risposte del server;

- *invio hash e PIN su server*: una volta ricevuta la risposta da parte del server estrae dalla propria memoria l'hash dei suoi dati ed il PIN che era stato

inserito manualmente dall'utente all'avvio dell'agente (mediante interfaccia grafica) e li inserisce come payload del nuovo messaggio (ovviamente esegue sempre i controlli sulla validità della firma digitale) lo cifra e lo invia all'agente sul client;

- *attesa risposta server e invio fingerprint*: se il server ha identificato l'utente allora l'agente può inviargli il template. Da questo punto in poi i messaggi vengono cifrati con la chiave AES dell'utente anch'essa presente nella memoria del telefono (notare l'utilizzo della funzione startAES(byte [] msg,boolean encrypt,String nameOfKey) dove come nome della chiave viene passato " MY " ovvero quella relativa all'utente);

Listing 6.8: Invio template

```
1  printMessage(R.id.TextView08,"SERVER wants fingerprint");
2  //decode message
3  byte[] messageDecoded;
4  try {
5          messageDecoded = startAES(message, false, "MY");
6
7          //set new message with payload Fingerprint
8          byte [] fingerprint=getData(hash);
9          byte [] replyMessage=setNewMessage(messageDecoded,
10                 fingerprint);
11         //Crypt replyMessage with My AES Key
12         byte[] replyEncryptMessage;
13
14         replyEncryptMessage = startAES(replyMessage, true, "MY");
15
16         //Sign message
17         byte [] signReplyMessage=utility.Android.RSAAndroid.
18                 RSASign(replyEncryptMessage,utility.
19                 Android.RSAAndroid.readPrivateKey
20                 (getMyRSAKey(false)));
21         String replyMessageHEX=utility.Android.
22                 HexUtilityAndroid.toHex
23                 (signReplyMessage)+
24                 utility.Android.HexUtilityAndroid.toHex
25                 (replyEncryptMessage);
26         //Send reply
27         ACLMessage reply=new ACLMessage(ACLMessage.INFORM);
28         AID destination=new AID();
29         destination.setLocalName((String)getDataStore().
30         get("MOBILE_AGENT_NAME"));
31         reply.addReceiver(destination);
32         reply.setContent(replyMessageHEX);
33         thisAgent.send(reply);
```

```
34          //update number of mobile agent
35          int value=Integer.parseInt((String)getDataStore().
36                  get("NUM_MOBILE_AGENT"));
37          value+=2;
38          getDataStore().put("NUM_MOBILE_AGENT",""+(value));
39          //Update name of mobile agent
40          updateNameOfMobileAgent();
41          printMessage(R.id.TextView09,
42                  "I sent my fingerprint to SERVER");
43          }
44  catch (FileNotFoundException e) {
45          // TODO Auto-generated catch block
46          e.printStackTrace();
47          }
48
49  return; //finish
```

- *attesa risultato autenticazione e relativa stampa a video*: l'agente ora rimane in attesa del risultato dell'autenticazione e, dopo averlo stampato a video, richiama il comportamento di autodistruzione.

Conclusioni

L'obiettivo di questa tesi era realizzare un sistema di autenticazione biometrica efficiente sia dal punto di vista delle prestazioni che della sicurezza, dove per sicurezza si intende quella riguardante i dati trasmessi all'interno del sistema e quella dell'autenticazione degli utenti. Il sistema realizzato ha come scopo quello di fornire una struttura estremamente valida sulla quale sviluppare servizi di vario tipo.

Attualmente le impronte digitali risultano essere il parametro biometrico più utilizzato nel campo dei sistemi di autenticazione, anche grazie alla loro persistenza ed individualità. Inoltre la presenza sul mercato di scanner anche professionali dal prezzo altamente competitivo ha dato un ulteriore impulso all'espansione di questa metodologia.

La scelta fatta in fase di progettazione di utilizzare il paradigma ad agenti mobili si è rivelata assolutamente valida; infatti, come è stato illustrato nei capitoli precedenti, gli agenti mobili sono in grado di interagire con un utente umano, crittografare informazioni riservate, trasportarle da un calcolatore ad un altro ed eseguire interrogazioni per memorizzarle in un database.

Altra caratteristica del sistema realizzato è l'utilizzo di uno smartphone di ultima generazione dal quale l'utente fa partire il processo di autenticazione e all'interno del quale sono memorizzati i dati relativi all'utente stesso. In questo modo è stato possibile implementare un sistema a tre fattori: conoscenza di un PIN, possesso di uno smartphone, rilevamento di un parametro biometrico.

Tra i possibili sviluppi futuri è opportuno sottolineare come se si riuscisse ad utilizzare l'add-on della piattaforma JADE per la sicurezza JADE-S anche su dispositivi mobili si aggiungerebbe un ulteriore grado di sicurezza al sistema. Un altro passo avanti sarebbe rappresentato dalla possibilità di implementare la mobilità degli agenti su dispositivi dalle limitate risorse hardware come gli smartphone.

Bibliografia

[1] La distribuzione Linux Ubuntu, http://www.ubuntu-it.org/.

[2] Il linguaggio di programmazione Java, http://www.java.com/it/.

[3] Il Java Development Toolkit, http://java.sun.com/javase/downloads/index.jsp.

[4] La libreria JAI, http://java.sun.com/javase/technologies/desktop/media/2D/.

[5] Il sistema operativo Android, http://www.android.com

[6] Il Database MySQL, http://www.mysql.it/.

[7] Lo scanner BiometriKa, http://www.biometrika.it/.

[8] Il software development kit di Android, http://developer.android.com/sdk/index.html

[9] Genco, A. (2004) *Agenti Mobili in Internet*, Firenze, Franco Angeli Editore.

[10] Lo standard KQML, http://www.cs.umbc.edu/kqml/.

[11] Lo standard MASIF, http://www.omg.org/.

[12] Lo standard FIPA, http://www.fipa.org/.

[13] La piattaforma Aglets, http://aglets.sourceforge.net/.

[14] La piattaforma Cougaar, http://www.cougaar.org/.

[15] La piattaforma Agent Factory, http://www.agentfactory.com/index.php/Main_Page.

[16] La piattaforma Semoa, http://semoa.sourceforge.net/about/about.html.

[17] La piattaforma JADE (*Java Agent DEvelopment platform*), http://jade.tilab.com/.

[18] Definizione di impronta digitale, `http://it.wikipedia.org/wiki/Impronta_Digitale`.

[19] BiometriKa S.r.l., azienda produttrice dello scanner biometrico Fx2000, `http://www.biometrika.it/`.

[20] Unione Europea - Gruppo dei Garanti Europei, *Working Party art. 29*, 1 Agosto 2003.

[21] Organisation for Economic Co-operation and Development, *Biometric-based technologies*, 9 marzo 2004.

[22] Gianpiero Paolo Cirillo, *Il codice sulla protezione dei dati personali*, Giuffrè Editore, Milano 2004