SVILUPPO DI UN SISTEMA DOCUMENTALE PER I BACK OFFICE BANCARI

RELATORE: Ch.mo Prof. Moro Michele

LAUREANDO: Michele Pantano

Corso di Laurea Magistrale in Ingegneria Informatica

A.A. 2011-2012

UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

SVILUPPO DI UN SISTEMA DOCUMENTALE PER I BACK OFFICE BANCARI

RELATORE: Ch.mo Prof. Moro Michele

LAUREANDO: Michele Pantano

A.A. 2011-2012

Ai miei genitori,
a mio fratello,
a Marica.

Indice

Sommario

L'obiettivo generale della tesi è quello di sviluppare un software per computer che consenta l'interfacciamento con periferiche di acquisizione di immagini (scanner, ma anche fotocamere), al fine di effettuare scansioni di documenti da inviare, una volta acquisiti, ad un sistema documentale modellato appositamente per gestirli. Il contesto in cui è inserito tale lavoro prevede la realizzazione di un progetto pilota riguardante il tema della dematerializzazione e della gestione documentale attraverso l'utilizzo di prodotti Open Source.
Per la realizzazione del software sono stati utilizzati strumenti Open Source facilmente reperibili in rete: Eclipse come ambiente di sviluppo, Alfresco come sistema documentale. L'applicazione è stata scritta con linguaggio di programmazione Java, facendo uso anche di alcune librerie di terze parti (apache-commons, Cook-Swing, jgoodies, OpenCMIS, jtwain) e della tecnologia Java Web Start, di cui ci si servirà per la distribuzione del software e di eventuali aggiornamenti o bug fix. Il risultato ottenuto da questo lavoro di tesi è quello di aver realizzato un applicativo desktop (lato client) in grado di acquisire immagini da periferiche di scansione o fotocamere, elaborarne l'aspetto grafico (aumentare o diminuire contrasto e luminosità, ruotarle, capovolgerle rispetto ad un'asse orizzontale o verticale, ridimensionarle, ecc.), salvarle su file system locale e successivamente inviarle al sistema documentale (la cui caratterizzazione è oggetto di un altro lavoro di tesi).

Capitolo 1

Introduzione

Se si guarda in generale al fenomeno dell'archiviazione elettronica di documenti cartacei, si può notare come esso stia assumendo un'importanza sempre più strategica nel mondo delle aziende e delle banche: introduce un alleggerimento ed una dematerializzazione dei processi aziendali con il vantaggio di ottenere, inoltre, un patrimonio informativo facilmente accessibile. I punti di forza di tale processo possono essere riassunti in tre caratteristiche:

- **Incremento della produttività**: le mansioni di gestione dei documenti (compilazione, archiviazione, ricerca, ...) vengono semplificate e velocizzate, liberando in questo modo risorse e riducendo i tempi operativi, a vantaggio della produttività;
- **Riduzione dei costi**: un sistema documentale in formato cartaceo comporta costi che possono essere eliminati riducendo il volume degli archivi fisici e delle risorse umane dedicate, ma anche abbattendo le spese riguardanti i consumabili d'ufficio;
- **Flessibilità nella condivisione delle informazioni**: un'architettura digitalizzata per la gestione documentale garantisce, se ben strutturata, un sistema in grado di fornire a tutti gli operatori abilitati un agevole accesso ai dati archiviati.

Per evidenziare l'importanza dell'archiviazione digitale dei documenti, in particolare all'interno delle banche (ambito che è stato oggetto del lavoro di tesi), si prenda in considerazione il Rapporto dell'*Associazione Bancaria Italiana (ABI)* intitolato *"Gestione documentale in banca"* (febbraio 2009). Il Rapporto sottolinea come le banche producano, ogni anno, circa 5,7 miliardi di fogli (documentazione cartacea) a fini comunicativi sia con la clientela, sia con i dipendenti;

tale mole di carta equivale ad una dimensione pari a quattro volte quella della superficie di Parigi.
Lo studio, realizzato da *ABI Lab*, il Consorzio dell'ABI per la Ricerca e lo Sviluppo delle Tecnologie per la Banca, mette in risalto inoltre come il deposito, la conservazione, la gestione logistica, la ricerca e la spedizione dei documenti in formato cartaceo prodotta in banca, costi al settore circa 105 milioni di Euro l'anno. A cui devono essere aggiunte le spese sostenute dalle singole filiali per l'acquisto di carta (altri 20 milioni di Euro l'anno), lo spazio dedicato agli archivi cartacei e l'archiviazione. In particolare, sul bilancio di ogni sportello pesano i costi di deposito e conservazione dei documenti bancari per lunghi periodi (1500 Euro l'anno) e la gestione logistica, la ricerca e la spedizione dei fogli che ha un costo totale di poco più di 1600 Euro l'anno.

Questo rapporto mette in luce i vantaggi della digitalizzazione dei documenti:

- l'aumento dell'efficienza degli istituti di credito;
- la riduzione dello spazio occupato per immagazzinare i documenti;
- l'ottimizzazione della loro classificazione che si riflette in un miglioramento nelle prestazioni di ricerca;
- la semplificazione delle operazioni di invio, duplicazione e condivisione;
- la possibilità di contribuire a rendere maggiormente eco-sostenibile la gestione documentale.

A riassumere il Rapporto dell'ABI sono le parole del presidente di ABI Lab, Domenico Santececca:

> *"Digitalizzare documenti e procedure con l'obiettivo di rendere più efficiente la gestione documentale per i principali settori produttivi pubblici e privati del Paese significa migliorare i processi operativi riducendo i costi."*

In questo scenario si inserisce il progetto svolto per la tesi: fornire agli istituti di credito un software che consenta di automatizzare e rendere più semplici e trasparenti le operazioni di digitalizzazione dei documenti cartacei e di invio degli stessi ai back office bancari. Scendendo un po' più nel dettaglio, quello che si è realizzato è un applicazione multipiattaforma per pc (con la previsione di realizzarne anche una versione per smartphone), scritto con linguaggio di

programmazione Java e che si è pensato di distribuire al cliente finale attraverso tecnologia Java Web Start. Il progetto conta 13 classi, 76 metodi per un totale di oltre 3600 righe di codice e coinvolge, tra le molte librerie, anche alcune di terze parti:

- *jgoodies* per la creazione di particolari layout grafici da applicare ai `JFrame` Java;
- *CookSwing* e *cookXml* per la creazione dinamica di oggetti Java, generati a partire dalla lettura di un file XML;
- *OpenCMIS* e *HTTPClient* per l'invio dei file tramite lo standard CMIS ed il protocollo HTTP al sistema documentale Alfresco;
- *jdom* per la scrittura ed il parsing di file XML;
- *log4j* per la creazione "automatizzata" di un log di programma.

Concentrandosi brevemente sul funzionamento del programma, si può notare come esso operi in maniera del tutto semplice e lineare. L'utente, una volta avviata l'applicazione, può scegliere se consultare lo storico dei documenti già digitalizzati (che raccoglie i dati in forma tabellare e li mostra a video in un'apposita finestra)o eseguire una nuova scansione; in caso scelga questa seconda opzione, dopo aver indicato la tipologia del documento da scansionare (tramite selezione di un radio button) ed aver inserito in un form alcuni dati legati al documento da convertire in formato digitale, si avvia il "cuore" del programma. Esso presenta un'area su cui viene mostrata l'immagine del documento scasionato ed un pannello laterale contenente i pulsanti per la scansione, la modifica e l'invio al sistema documentale del documento acquisito.

Nei prossimi capitoli si andrà ad analizzare i singoli passi che hanno portato a realizzare il prodotto descritto, soffermandosi in maniera particolare sull'utilizzo degli strumenti e delle librerie meno note e sottolineando le difficoltà incontrate in questo percorso e le relative soluzioni applicate. I prerequisiti necessari alla comprensione dell'elaborato sono le nozioni base di programmazione Java, la conoscenza dei descrittori XML e JNLP, la conoscenza del protocollo HTTP.

Capitolo 2

L'azienda E-project s.r.l.

E-project s.r.l. è l'azienda che mi ha ospitato e supportato nella realizzazione di questo lavoro di tesi.

E-project s.r.l. è specializzata nella realizzazione, nella gestione e nello sviluppo di progetti di tipo organizzativo ed informatico; l'azienda è stata fondata nel 2001 da professionisti provenienti dal mondo della consulenza aziendale ed informatica e dal mondo del "document managment". I settori in cui E-project s.r.l. prevalentemente opera sono i seguenti:

- Servizi;
- Assicurazioni;
- Banche;
- Pubblica Amministrazione.

Nel settore dei Servizi ha contribuito a realizzare soluzioni legate alla conservazione sostituitiva.
Nel settore delle assicurazioni ha realizzato soluzioni con l'utilizzo di prodotti di document managment e BPM[1]
Nel settore Bancario possiede grandi competenze nelle seguenti aree:

- Sportello di filiale
- Incassi e Pagamenti

[1]Business Process Management: l'insieme delle attività necessarie per definire, ottimizzare, monitorare e integrare i processi aziendali, con lo scopo di soddisfare bisogni e volontà dei clienti e quindi migliorare le attività economiche dell'azienda.

- Gestione documentale
- Organizzazione dell'area "operations" e dei "back-office centralizzati"

Nel settore Pubblica Amministrazione vanta progetti e consulenze presso:

- il gruppo InfoCamere/UnionCamere nell'area Registro Imprese, Albo Vigneti, Albo Cooperative, area Sistemistica (DARS);
- alcuni progetti o parti di progetti realizzati sono: conciliazione on-line (Concilianet), sistema di legalità per la regione Campania (Legalità), interventi su Telemaco, Copernico, Quorum, Albo delle Cooperative etc. Comuni dell'Estense: è stato realizzato lo Sportello Unico sia dal punto di vista organizzativo sia informatico.

E-project vanta 14 operatori nel settore informatico con il 92% di laureati e un network di aziende partner di prodotto e system integrator che permettono di rispondere tempestivamente alle esigenze progettuali.

Figura 2.1: Logo della E-project s.r.l.

Capitolo 3

TWAIN

Una delle componenti fondamentali per il progetto a cui ho lavorato è quella di scansione, che si occupa di realizzare un tramite tra il software applicativo e le periferiche d'acquisizione d'immagini, per la "cattura" in formato digitale di documenti. Tale componente avrebbe anche potuto utilizzare i software proprietari dei dispositivi, ma sfruttando le librerie JTwain (per ulteriori approfondimenti si veda 3.3) il processo non solo diviene più rapido, ma consente anche all'utente di evitare la noiosa installazione di ulteriore software (favorendo così la distribuzione dell'applicazione tramite tecnologia Java Web Start).

Con l'introduzione di scanner, macchine fotografiche digitali ed altre periferiche di acquisizione di immagini è divenuto sempre più semplice ed alla portata di tutti allegare ai documenti o ai propri lavori dei file d'immagine; per mezzo di questi si possono aggiungere ulteriori informazioni agli elaborati allo scopo di renderli più precisi e dettagliati. Sul finire degli anni '80 ed agli inizi degli anni '90, in concomitanza con la nascita di nuovi dispositivi in grado di digitalizzare immagini, ai programmatori viene richiesta l'abilità di creare applicazioni che sappiano mostrare e modificare le immagini acquisite; il compito è davvero impegnativo poiché è necessario implementare un'interfaccia grafica e soprattutto una serie di funzioni che siano capaci di controllare l'ampia gamma di dispositivi per l'acquisizione di immagini. Inoltre, una volta che l'applicazione è pronta a supportare un particolare apparecchio, ne vengono introdotti nel mercato di nuovi, con nuove caratteristiche e nuove funzionalità, vanificando così gli sforzi di molti sviluppatori e costringendoli a modificare in maniera pressoché continuativa i loro applicativi. Sia le grandi aziende produttrici di periferiche che gli sviluppatori di software sentono quindi l'esigenza di un insieme di "vincoli" che regolino la comunicazione tra i dispositivi e le applicazioni e che portino stabilità

e semplicità in questo particolare settore dell'informatica. La creazione di uno standard sembra poter essere la soluzione che risolve i problemi di entrambe le parti: le case produttrici di dispositivi possono commercializzare dei prodotti in grado di interfacciarsi con un maggior numero di software diversi; i programmatori invece possono dedicarsi ad un unico progetto che sia in grado di dialogare con dispositivi diversi.

Il progetto **TWAIN** nasce nel 1991 dall'esigenza di creare un *protocollo software standard* ed un'*interfaccia di programmazione delle applicazioni* (API) che regoli la comunicazione tra applicativi software e strumenti di acquisizione di immagini. L'obiettivo di questo piccolo gruppo di lavoro è, fin da subito, quello di fornire una soluzione open source e multipiattaforma, per soddisfare le esigenze di interfacciamento tra le periferiche di input delle immagini ed i software. Al progetto iniziale prendono parte i rappresentanti di cinque società (Aldus, Caere, Eastman Kodak, Hewlett-Packard e Logitech), che in seguito ricevono importanti contributi anche da altre aziende (tra cui Adobe, Howtek e Software Architects). Nel corso degli anni il progetto ha assunto dimensioni sempre maggiori, basti pensare che ad oggi il gruppo di lavoro TWAIN conta circa 300 persone, che rappresentano approssimativamente 200 diverse società e che continuano ad influenzare e guidare il progetto; gli sviluppi futuri del protocollo sono mirati a migliorare lo standard per riuscire ad accogliere le nuove tecnologie.

Vantaggi nell'utilizzo di TWAIN

Lo standard TWAIN ha portato molteplici vantaggi ai programmatori di applicazioni, agli sviluppatori di software di periferica[1], ma anche agli utenti finali. Per uno sviluppatore di applicativi, l'utilizzo di TWAIN permette infatti:

- di realizzare un'applicazione che consenta all'utente finale di acquisire immagini in modo semplice ed immediato da qualsiasi dispositivo compatibile, senza dover abbandonare l'applicazione che si sta utilizzando;

- di ottenere un risparmio in termini di tempo e di costi, rendendo non più necessaria la fase di scrittura e di distribuzione di driver specifici per ogni singola periferica;

[1]Software di periferica o Source software: software per il controllo e la gestione dei dispositivi cui sono associati

- di consentire all'applicazione di accedere alle periferiche TWAIN-compatibili semplicemente attraverso il proprio codice, utilizzando l'API offerta da TWAIN;
- di determinare le proprietà (risoluzione, acquisizione a colori o in bianco e nero, ecc.) che un dispositivo può fornire. In tal modo, è possibile effettuare una "limitazione" sul Source software ed offrire solo le carateristiche che possono essere compatibili con le esigenze della propria applicazione;
- di far tralasciare allo sviluppatore l'implementazione di un'interfaccia per il controllo del processo d'acquisizione dell'immagine: in TWAIN esiste già un modulo che si occupa di questo.

Per lo sviluppatore di software di periferica invece TWAIN offre i seguenti vantaggi:

- aumenta l'utilizzo e la compatibilità del proprio prodotto. Molte applicazioni possono infatti integrare le funzioni di acquisizioni di immagini, grazie alla facilità di implementazione fornita da TWAIN;
- fornisce all'utente finale un'interfaccia ad hoc per il proprio dispositivo che ne comanda con esattezza tutti i parametri;
- permette un risparmio economico: anziché creare e fornire supporto per diverse versioni del proprio software di controllo della periferica, è possibile creare un singolo software di dispositivo che sia TWAIN-compatibile;
- non obbliga l'applicazione a fornire un'interfaccia per controllare il processo d'acquisizione dell'immagine: esiste un apposito modulo TWAIN che si occupa di questo.

Anche per l'utente finale TWAIN ha portato un grosso vantaggio: permette in modo semplice ed intuitivo di allegare immagini ai propri documenti. È possibile quindi incorporare degli allegati in pochi semplici passi poiché non serve uscire dall'applicazione che si sta utilizzando.

3.1 L'architettura di TWAIN

Gli elementi chiave di TWAIN sono tre semplici componenti:

- **L'applicativo software**: un'applicazione che deve essere strutturata per poter utilizzare TWAIN;
- **Il software di gestione delle periferiche (Source Manager software)**: la componente in grado di gestire l'interazione fra le periferiche e l'applicazione;
- **Il software di periferica (Source software)**: è il software che si occupa del controllo dell'acquisizione delle immagini, scritto dai produttori delle periferiche in modo da soddisfare le specifiche TWAIN. I driver tradizionali dei dispositivi sono, di norma, sempre inclusi nelle periferiche mediante questi software.

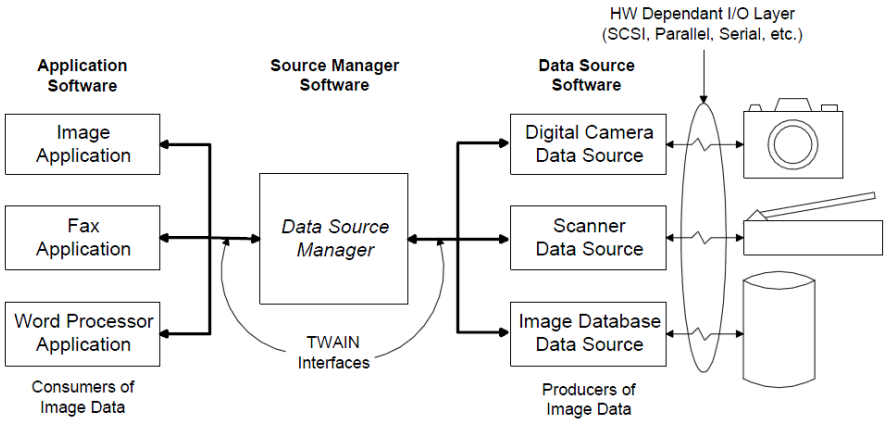


Figura 3.1: Elementi di TWAIN

Il trasferimento di dati dalle periferiche al computer è quindi possibile grazie al funzionamento di queste componenti che interagiscono tra loro. Questi tre elementi usano l'architettura di TWAIN per comunicare l'un l'altro; l'architettura prevede quattro livelli, disposti come nell'immagine che segue (3.2):

- livello 1. *Applicazione*
- livello 2. *Protocollo*
- livello 3. *Acquisizione*
- livello 4. *Dispositivo*

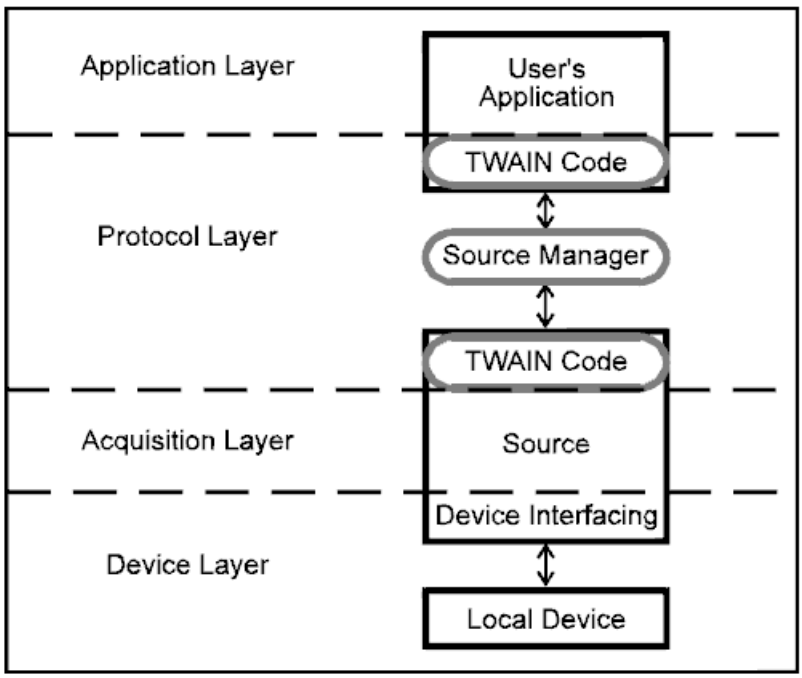


Figura 3.2: Elementi software di TWAIN

Livello di Applicazione

In questo livello viene eseguito il software dell'utente. TWAIN descrive le linee guida dell'interfaccia utente per gli sviluppatori di programmi, per quanto concerne il modo in cui gli utenti accedono alle funzionalità di TWAIN ed il modo in cui una particolare risorsa possa essere selezionata. TWAIN non fornisce ulteriori dettagli o suggerimenti riguardanti l'implementazione dell'applicativo.

Livello di Protocollo

Il protocollo può essere inteso come l'insieme di regole che definiscono il "linguaggio parlato" e la sintassi utilizzata da TWAIN. Esso implementa precise istruzioni e richieste di comunicazioni per il trasferimento dei dati. Il *Livello di Protocollo* comprende tre componenti:

- una porzione di software dell'applicazione che si occupa di fornire un'interfaccia tra il programma e TWAIN;

- il Source Manager, interamente affidato a TWAIN;

- il software incluso con la periferica di cattura delle immagini, che consente al prodotto di ricevere le istruzioni dal Source Manager e di trasferire dati e Return Codes[2];

Livello di Acquisizione

Gli strumenti di acquisizione possono essere sostanzialmente di due tipi:

- **logici** - database di immagini;
- **fisici** - scanner, macchine fotografiche digitali, telefoni cellulari.

Gli elementi del software scritti per controllare le acquisizioni sono detti *Sources* e risiedono principalmente in questo livello. Un Source ha il compito di trasferire i dati destinati all'applicazione, usando un formato ed un meccanismo di trasferimento che viene stabilito in fase di progettazione dal programmatore. Ciascun Source dovrebbe fornire un'interfaccia per l'utente, così da permettergli di controllare il dispositivo per cui è stata scritta: tuttavia tale interfaccia può anche non esser sviluppata, qualora non dovesse essercene bisogno.

Livello di Dispositivo

In questo livello viene posto, di norma, il driver del dispositivo. Un driver converte i comandi specifici ad alto livello indirizzati al dispositivo, in comandi hardware ed azioni specifiche a basso livello per il particolare dispositivo per cui esso è stato scritto.
Le applicazioni che fanno uso di TWAIN non necessitano più di driver proprietari, perché essi sono "inglobati" nei software di periferica. Tra i compiti di un software di dispositivo infatti c'è anche quello di occuparsi della traduzione delle operazioni TWAIN e delle interazioni con l'interfaccia utente, in comandi equivalenti per il driver della periferica, che fan sì che il dispositivo si comporti nel modo desiderato. TWAIN non fa parte del *Livello di Dispositivo*; infatti uno dei compiti del Source software è anche quello di nascondere all'applicazione questo livello.

[2]Return Codes: codici indicanti lo stato del dispositivo o l'esito dell'esecuzione di un comando

3.2 L'interfaccia utente di TWAIN

Il processo di acquisizione di un'immagine attraverso un'applicazione che utilizza TWAIN, può essere visibile all'utente nelle seguenti tre aree:

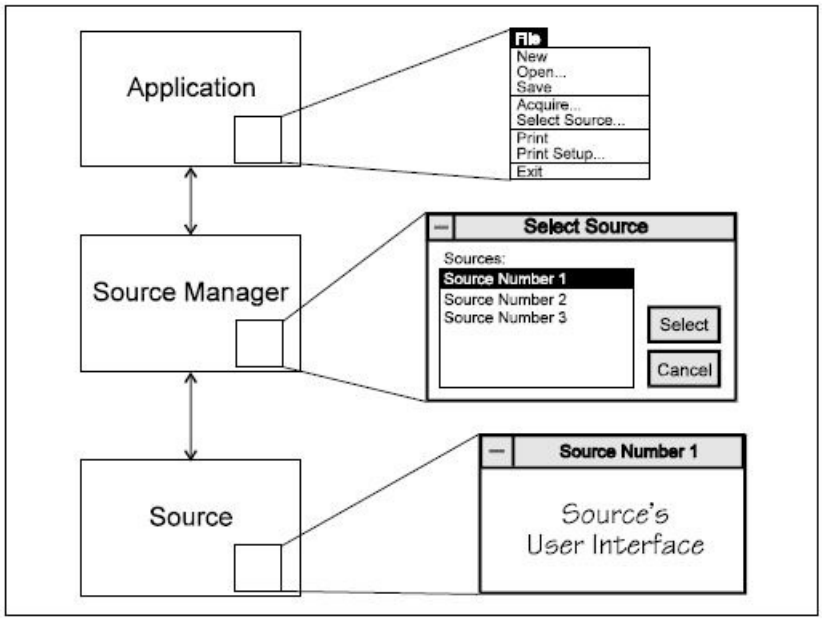


Figura 3.3: Processo di acquisizione dei dati

L'applicazione

L'obiettivo dell'applicazione è quello di acquisire e gestire dati da una sorgente di input. L'applicazione tuttavia non può "dialogare" direttamente con il dispositivo: Return Codes, informazioni sulle caratteristiche del dispositivo, sugli errori ecc. devono essere gestiti attraverso il Source Manager. È necessario quindi che l'utente specifichi per prima cosa il dispositivo da cui intende catturare dati; inoltre è essenziale per l'utente capire quando tali periferiche siano pronte ad acquisire i dati. Per consentire questa caratteristica, il gruppo di lavoro TWAIN sottolinea ai programmatori l'importanza di introdurre nel *File Menù* due semplici opzioni:

- **Seleziona dispositivo** - per selezionare la periferica
- **Acquisisci** - per iniziare il processo di trasferimento

Il Source Manager

Il Source Manager è il componente che fornisce "il canale di dialogo" tra l'applicazione e la periferica hardware di cattura delle immagini. Il Source Manager consente la selezione del dispositivo da parte dell'utente ed al contempo rende disponibile all'applicazione un'"istanza" di tale dispositivo. Inoltre il Source Manager può dar vita a tre operazioni che non sono direttamente originate dall'applicazione: identificare le risorse disponibili, attivarle (`SourceManager.instance().openSource()`) e disattivarle (`SourceManager.instance().closeSource()`). Quando l'utente sceglie l'opzione *Seleziona dispositivo* dal *File Menù*, l'applicazione richiede al Source Manager che mostri a video la sua finestra di dialogo nella quale sono elencati tutti i dispositivi disponibili; per mezzo di questa dialog box l'utente può selezionare la periferica desiderata. Se necessario, lo sviluppatore software può sovrascrivere con una propria verione l'interfaccia utente per la scelta del dispositivo (grazie ai metodi `SourceManager.instance().getAllSources()` e `SourceManager.instance().selectSourceByName(String sourceName)`).

Il software di periferica

Il software di periferica riceve comandi sia dall'applicazione, attraverso il Source Manager, sia direttamente dal Source Manager. Esso processa le richieste e ritorna un appropriato Return Code per ciascuna di esse, indicando al Source Manager il risultato dell'operazione. Ogni Source software TWAIN-compliant fornisce all'utente un'interfaccia specifica per l'utilizzo della periferica scelta. Quando l'utente seleziona l'opzione *Acquisisci*, l'interfaccia utente della periferica può essere visualizzata (previo l'aver inserito tra il proprio codice il comando `source.setUIEnabled(true)`); come già visto per l'interfaccia di selezione del dispositivo, anche in questo caso lo sviluppatore, a propria discrezione, può decidere di implementare la propria versione.

3.2.1 Protocollo di comunicazione

Il processo per l'acquisizione di un'immagine richiede che l'applicazione, il Source Manager ed il Source software comunichino rispettando un particolare ordine. Un banale esempio può essere quello che vede l'applicazione effettuare una richiesta per il trasferimento dei dati ad una periferica: tale richiesta non può essere soddisfatta se prima non è stato "attivato" il Source Manager e "predisposto" ad effettuare una richiesta. Affinché sia assicurata una sequenza di comunicazione corretta, il protocollo TWAIN definisce, all'interno di una *sessione*, sette stati.

Una sessione è un lasso temporale che vede l'applicazione connessa ad una particolare periferica attraverso il Source Manager. Il periodo di tempo nel quale un'applicazione risulta connessa al Source Manager è considerata un'unica sessione (una sessione quindi inizia col comando `SourceManager.instance().open()` e termina col comando `SourceManager.instance().close()`).
Analizzando ora una singola sessione, ad un dato istante si può osservare come ciascuno dei componenti TWAIN in gioco assuma uno stato ben definito; le transizioni verso un nuovo stato sono determinate da operazioni richieste dal software di periferica o dall'applicazione. Non c'è un senso univoco per le transizioni, esse possono essere uscenti od entranti da o in ciascuno stato e la maggior parte di esse sono transizioni che portano ad un unico stato.
Osservando il protocollo di comunicazione è utile ricordare che gli stati 1, 2, 3 sono occupati esclusivamente dal Source Manager, il quale invece non occupa mai stati superiori al terzo. Gli stati 4, 5, 6, 7 vengono esclusivamente occupate dal software di periferica. Il Source software, qualora sia attivo, non si trova mai in uno stato inferiore al quarto; nel caso sia inattivo, non occupa alcuno stato. Se una stessa applicazione utilizza più dispositivi, ciascuna connessione appartiene ad una sessione indipendente e ciascuna periferica si trova in uno stato indipendentemente dallo stato in cui si trovano le altre periferiche.

Descrizione degli stati

Nella seguente sezione, verranno descritti gli stati del protocollo di comunicazione di TWAIN.

Stato 1 Pre-Sessione

Il Source Manager risiede nello stato 1 prima che l'applicazioni stabilisca una sessione con esso. In questa stato, il codice del Source Manager è installato su disco ma non è ancora caricato in memoria. L'unico caso in cui il Source Manager può essere già caricato ed in esecuzione è negli ambienti Windows, perché la sua implementazione è una DLL[3] (quindi la stessa istanza del Source Manager può

[3]Dynamic Link Library: è una libreria software che viene caricata dinamicamente in fase di esecuzione. La separazione del codice in librerie a collegamento dinamico permette di suddividere il codice eseguibile in parti concettualmente separate, che verranno caricate solo se effettivamente necessarie. Inoltre, una singola libreria, caricata in memoria, può essere utilizzata da più programmi, senza la necessità di essere nuovamente caricata, il che permette di risparmiare le risorse del sistema.

essere condivisa da più applicazioni). Se ciò accade, il Source Manager potrebbe trovarsi nello stato 2 o 3 nell'applicazione che lo sta utilizzando.

Stato 2 Source Manager Caricato
A questo punto il Source Manager viene caricato in memoria, ma non è ancora in esecuzione. Esso è pronto a ricevere comandi di istruzione dall'applicazione.

Stato 3 Source Manager Aperto
Il Source Manager è ora avviato ed è pronto ad interagire con il Source software. Il Source Manager può, in questo stato, elencare le periferiche connesse al computer e, per ciascuna di esse, può decidere se renderle operative o meno. Il Source Manager resterà nello stato 3 per tutta la durata della sessione, fino al momento in cui la sua esecuzione verrà terminata. Il Source Manager non potrà essere terminato nel caso in cui ci sia ancora qualche periferica attiva.

Stato 4 Source software Aperto
Il Source software è stato caricato ed è stato avviato dal Source Manager in risposta ad un'operazione dell'applicazione ed è quindi inoltre pronto a ricevere ulteriori comandi. Il software di periferica dovrebbe aver verificato che le risorse richieste (memoria, periferica, ecc.) siano disponibili; l'applicazione è quindi pronta per recuperare informazioni sulle caratteristiche della periferica (risoluzione, supporto di immagini a colori, alimentazione automatica dei fogli da scansionare, ecc.). L'applicazione può inoltre impostare i valori desiderati per ciascuna delle caratteristiche offerte dal dispositivo: per esempio, può far sì che l'acquisizione delle immagini avvenga esclusivamente in bianco e nero.

Stato 5 Source software abilitato
Il Source software è stato abilitato da un comando dell'applicazione attraverso il Source Manager ed è pronto ad effettuare il traferimento di immagini. Se l'applicazione consente al software di dispositivo di mostrare la propria interfaccia (`source.setUIEnabled(true)`), questo avviene quando entra nello stato 5.

Stato 6 Il Trasferimento è pronto
Il software di periferica è pronto a trasferire una o più immagini all'applicazione. La transizione dallo stato 5 allo stato 6 è innescata dal Source software che notifica all'applicazione che il trasferimento è pronto. Prima che l'invio dei dati abbia inizio, l'applicazione deve aver conosciuto i dettagli dell'immagine (risoluzione, dimensione, ecc.). È possibile anche il trasferimento di più immagini in successione.

Stato 7 Trasferimento

Il Source software sta trasferendo l'immagine all'applicazione. Il meccanismo di trasferimento utilizzato è negoziato durante lo Stato 4. Il trasferimento dei dati può esser completato con successo oppure può esser interrotto prematuramente; in entrambi i casi, il software di dispositivo invia un *Return Code* all'applicazione indicante l'esito del trasferimento. Una volta che il Source software indica che il trasferimento è completato, l'applicazione deve riconoscere la fine del trasferimento.

In figura è riportato il diagramma degli stati e delle transizioni:

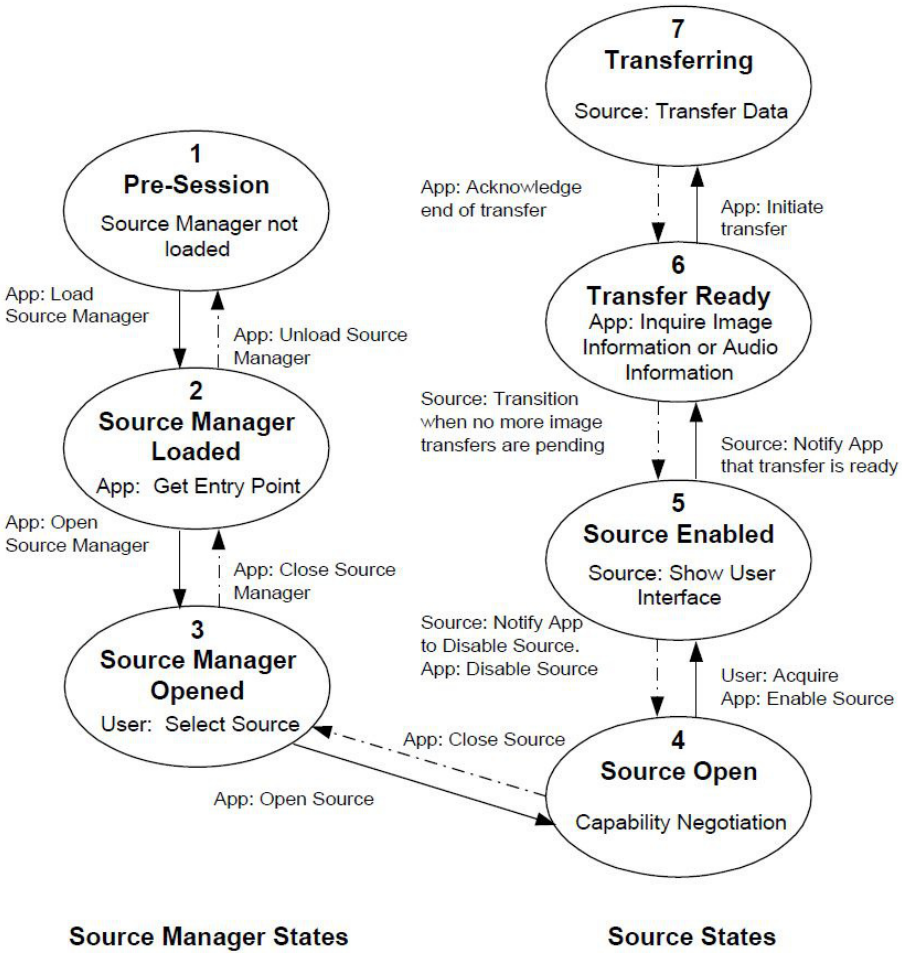


Figura 3.4: Protocollo di comunicazione - Diagramma degli stati

3.2.2 Interazione con le caratteristiche dei dispositivi

Uno dei benefici di TWAIN è che consente all'applicazione di interagire facilmente con una grande varietà di dispositivi d'acquisizione di immagini. Tali dispositivi possono avere diverse caratteristiche, ad esempio:

- alcuni dispositivi godono del caricamento automatico dei documenti da scansionare;
- alcuni dispositivi non si limitano all'acquisizione di una singola immagine, ma possono trasferire immagini multiple;
- alcuni dispositivi supportano la scansione a colori;
- alcuni dispositivi riescono a gestire anche una varietà di mezzi toni;
- alcuni dispositivi supportano un range di risoluzioni mentre altri possono offrire range differenti.

Gli sviluppatori di applicazioni hanno bisogno di conoscere le caratteristiche di un Source e possono influenzarne le feature che esso offre all'utente finale. Per far ciò, l'applicazione può eseguire una negoziazione delle caratteristiche. L'applicazione in genere segue questo processo:

1. determina se il Source selezionato disponga di una particolare caratteristica;
2. acquisisce il valore corrente della caratteristica selezionata. Inoltre, acquisisce i valori di default delle caratteristiche e l'insieme dei valori disponibili che sono suppoortati dal Source per quel particolare dispositivo;
3. richiede che il Source imposti il valore corrente di una caratteristica ad un particolare valore espresso dall'applicazione;
4. limita, se necessario, i valori disponibili per il Source ad un sottoinsieme di valori che siano compatibili con quelli che devono essere offerti. Ad esempio, se l'applicazione richiede scansioni in bianco e nero, si possono restringere le capacità del Source affinché si limiti ad acquisire immagini di questo tipo. Se una limitazione ha effetti sull'interfaccia utente del Source, allora il programmatore dovrebbe andare a sovrascrivere l'interfacccia in modo che rispecchi tali limitazioni, per esempio rimuovendo o colorando di grigio le opzioni che non sono disponibili.
5. verifica che i nuovi valori siano accettati dal Source.

3.3 JTwain

JTwain è la controparte Java di TWAIN. È una libreria Java commerciale sviluppata e distribuita da LAB Asprise! a partire dal 1998. Oltre alla gestione di periferiche TWAIN-compatibili, offre altri strumenti e funzionalità, come ad esempio un semplice programma di editing per le immagini (sviluppato in linguaggio Java) o un sistema di publicazione (simultanea) su WEB (tool questi non presi in considerazione poiché non rientranti negli scopi del progetto). L'utilizzo di questa libreria è semplice, perché non prevede la possibilità di eseguire scansioni parallele e quindi esclude realizzazioni particolarmente complesse e delicate che scaturiscono dall'uso dei Thread. Uno dei punti che gioca invece a sfavore dell'utilizzo di questa libreria è il fatto che JTwain operi solamente su sistemi Win32. Per un'integrazione verso i sistemi UNIX di qualsiasi tipo (quindi non solo Linux, ma anche Mac) bisogna acquistare un altro prodotto, il JSANE. *SANE* è uno standard *de facto* per l'accesso a scanner e fotocamere da parte dei sistemi MAC, Linux, OpenBSD, Solaris, Unix. JSANE fornisce le API Java per l'utilizzo di SANE.

3.3.1 Installazione del Software Development Kit JTwain

L'installazione del Software Development Kit (SDK) JTwain è di fondamentale importanza per lo sviluppatore che voglia creare un'applicazione che utilizzi lo standard TWAIN.
Innazitutto, sulla propria macchina deve essre installata una versione di JRE uguale o superiore alla 1.2 (versione comunque obsoleta, risalente all'anno della prima release di JTwain - 1998). Al momento, JTwain supporta solo i seguenti sistemi operativi: Windows 98, NT, ME, 2000, XP e tutte le piattaforme Windows Server. Una volta scaricata copia del file di installazione di JTwain dal sito `http://www.asprise.com/product/jtwain`, è necessario scompattare l'SDK in una cartella, che assumerà il ruolo della JTWAIN_HOME. Dopo questi semplici passaggi, sarà sufficiente eseguire il file LaunchDemo.bat e selezionare TestJTwain per testare la corretta installazione di JTwain.
In seguito all'installazione, è necessario configurare l'ambiente di sviluppo per poter creare applicazioni Java che utilizzino JTwain. Le operazioni essenziali per un corretto setup dell'ambiente di sviluppo sono:

- porre il file *JTwain.jar* nel proprio *class path*;

- porre il file *AspriseTwain.dll* nel proprio *system path* (in ambienti Windows va posto in C: Windows System32).

Dati i costi elevati per l'acquisto di una licenza JTwain, per il mio applicativo ho fatto uso di una verisone *demo*. Questa consente l'utilizzo di tutte le funzioni messe a disposizione dalla versione a pagamento (per un periodo di tempo limitato a 30 giorni), salvo applicare sulle immagini scansionate un watermark che ricorda che si sta utilizzando la versione di prova della libreria.

3.4 Implementazione

Come anticipato nell'introduzione, l'applicazione sviluppata in questo progetto di tesi, prevede una componente di scansione. Si analizzerà all'interno di questo paragrafo come è stata realizzata tale componente, quali comandi sono necessari per la sua esecuzione e che tipo di difficoltà si sono incontrate nell'implementazione di tale caratteristica.
Il metodo che andiamo ad analizzare è il metodo `scanAction(ActionEvent event)` che viene eseguito alla pressione del JButton *Scansione* presente nel pannello principale dell'applicazione. Il funzionamento delle librerie jtwain è davvero molto semplice pertanto anche il codice Java sarà altrettanto chiaro e lineare.
Per prima cosa c'è da dire che l'intero codice per l'interfacciamento con scanner o altri dispositivi, va racchiuso all'interno del paradigma `try/catch/finally` ed il motivo è presto spiegato: con `finally` si mira a rilasciare immediatamente una risorsa che può esser rimasta bloccata dal prolungarsi di un'operazione di gestione di un'eccezione. Nel blocco `try` vanno inseriti due comandi fondamentali, che servono ad iniziare una sessione di scansione; il primo comando è `Source source = SourceManager.instance().selectSourceUI()` mentre il secondo è `source.open()`. Tramite il primo comando viene mostrata una dialog box mediante la quale è possibile scegliere la periferica desiderata (tra una lista di periferiche collegate alla macchina su cui viene eseguita l'applicazione), tramite il secondo viene invece "aperta" la comunicazione con il dispositivo d'acquisizione di immmagini. L'oggetto di tipo `SourceManager` rappresenta effettivamente il *Source Manager* di TWAIN, le cui caratteristiche sono state illustrate in 3.2; può esserci una ed una sola istanza di SourceManager attiva per volta. È obbligatorio l'inserimento di questa porzione di codice all'interno del blocco `try` in quanto essi possono lanciare l'eccezione `JTwainException` che va obbligaotriamente gestita

(nel blocco `catch`). Una volta che si è individuato quale sia la periferica che verrà utilizzata e che la si è resa disponibile all'acquisizione di dati, si procede con la scansione tramite il comando `source.acquireImage()`. Il blocco `finally` include una sola istruzione `SourceManager.closeSourceManager()` per mezzo della quale si rilascia il Source Manager.
Il blocco di codice nella sua interezza è il seguente:

Listing 3.1: Il metodo void scanAction (ActionEvent event)

```
1  try{
2      SourceManager.instance().selectSource(source);
3      source.open();
4
5      source.setXResolution(100);
6      source.setYResolution(100);
7      bufferedImage = source.acquireImageAsBufferedImage();
8      scanDate = Tools.date();
9      scanTime = Tools.time();
10
11     //salvo le immagini in file .tiff
12     Tools.saveToTIFF(countPages, bufferedImage);
13     countPages++;
14
15     //mostro l'immagine nel main panel
16     Tools.showImage(bufferedImage, imageScrollPane);
17     source.close();
18     }
19     catch(JTwainException e) {
20         logger.error("scanAction(ActionEvent)",e);
21     }
22     finally{
23   SourceManager.closeSourceManager();
24     }
```

Rispetto a quanto precedentemente illustrato, ci sono alcune piccole differenze che è bene spiegare: il primo comando eseguito è `SourceManager.instance()` `.selectSource(Source source);` che a differenza di quanto detto pocanzi, non consente all'utente di scegliere la periferica da utilizzare, ma ne assegna una prestabilita. Questa è una scelta progettuale che è stata effettuata in base al tipo di utilizzo cui è destinato il software: l'operatore di banca non può (e non deve) ogni qual volta voglia effettuare la scansione di un documento, perdere del

tempo per selezionare una periferica piuttosto di un'altra. Per questo motivo il comando `Source source = SourceManager.instance().selectSourceUI()` è utilizzato un'unica volta (al primo avvio assoluto dell'applicazione); una volta scelta la periferica viene salvato un file XML contenente il nome del dispositivo selezionato e ad ogni successivo avvio del programma, verrà letto tale file e si cercherà di utilizzare il nome della risorsa indicata nel file. Se per errore al primo avvio venisse selezionata la periferica sbagliata, viene data all'utente una duplice possibilità: la cancellazione del file XML riportante il nome del dispositivo prescelto oppure la possibilità di scegliere un altro dispositivo attrraverso apposita funzione presente nel menù File del programma. Il codice che gestisce la selezione dello scanner (o altro dispositivo) da utilizzare è il seguente:

Listing 3.2: Il metodo Source selectSource(File file)

```
1  Source selectSource(File file){
2      File fileSource = file;
3      Source source = null;
4
5      try{
6          Source[] sources = SourceManager.instance().
             getAllSources();
7    if(sources.length==0){  //no sources
8        Tools.warningFrame("No source found", "Try to connect a
           source, then retry");
9    }
10           else{
11       if(!fileSource.exists()){
12           source = SourceManager.instance().selectSourceUI();
13           writeSourceName(source, fileSource);
14       }
15       else{
16           source = readSourceName(sources, fileSource);
17       }
18           }
19     }
20     catch(JTwainException e){
21         logger.error("selectSource(File)JTwainException",e);
22     }
23     return source;
24 }
```

Anche in questo caso il codice riportato è estremamente banale: si fa prima di tutto un controllo sulla presenza o meno di dispositivi. Se sono presenti dei dispositivi collegati al computer, allora si va a leggere il file XML contenente il nome della periferica da utilizzare per la scansione (il file è passato come parametro al metodo). Se tale file non esiste (in caso di prima esecuzione del programma o di cancellazione da parte dell'utente) viene allora invocato il comando `SourceManager.instance().selectSourceUI();` con cui viene scelto il dispositivo (il cui nome verrà quindi scritto nel file XML tramite metodo `writeSourceName(source, fileSource)`). Nel caso in cui il file sia già esistente invece ci si limita, tramite il metodo `Source readSourceName(sources, fileSource)`, a leggere il nome della periferica "di default" e la si utilizza per effettuare la scansione. La periferica viene ricercata nell'array che contiene l'elenco dei dispositivi di scansione collegati alla propria macchina
`Source[] sources = SourceManager.instance().getAllSources();`.

Per quanto riguarda invece le periferiche utilizzate, ho verificato il funzionamento dell'applicazione su tre diversi dispositivi:

- HP Photosmart C4780: tale apparecchio è un dispositivo multifunzione (scanner e stampante); non prevede il caricamento automatico dei fogli, che devono essere quindi posizionati manualmente. Per un corretto utilizzo dell'applicazione è consigliato disabilitare l'interfaccia utente del software proprietario.

- HP Scanjet 5550c: scanner dotato di funzione di auto-alimentazione dei fogli da digitalizzare. E' stato quindi possibile testare una delle funzionalità messe a disposizione da JTwain: l'acquisizione in successione di più documenti. Sono stati eseguiti dei test in ambiente WindowsXP emulato, per mancanza di driver compatibili con versioni Windows più recenti.

- Apple iPhone: se connesso al computer tramite cavo USB, viene riconosciuto come una fotocamera digitale. Per poterlo utilizzare correttamente è necessario attivare l'interfaccia utente (`source.setEnableUI(true)`) che permette la scelta delle immagini salvate nel telefono ed il conseguente utilizzo all'interno dell'applicazione.

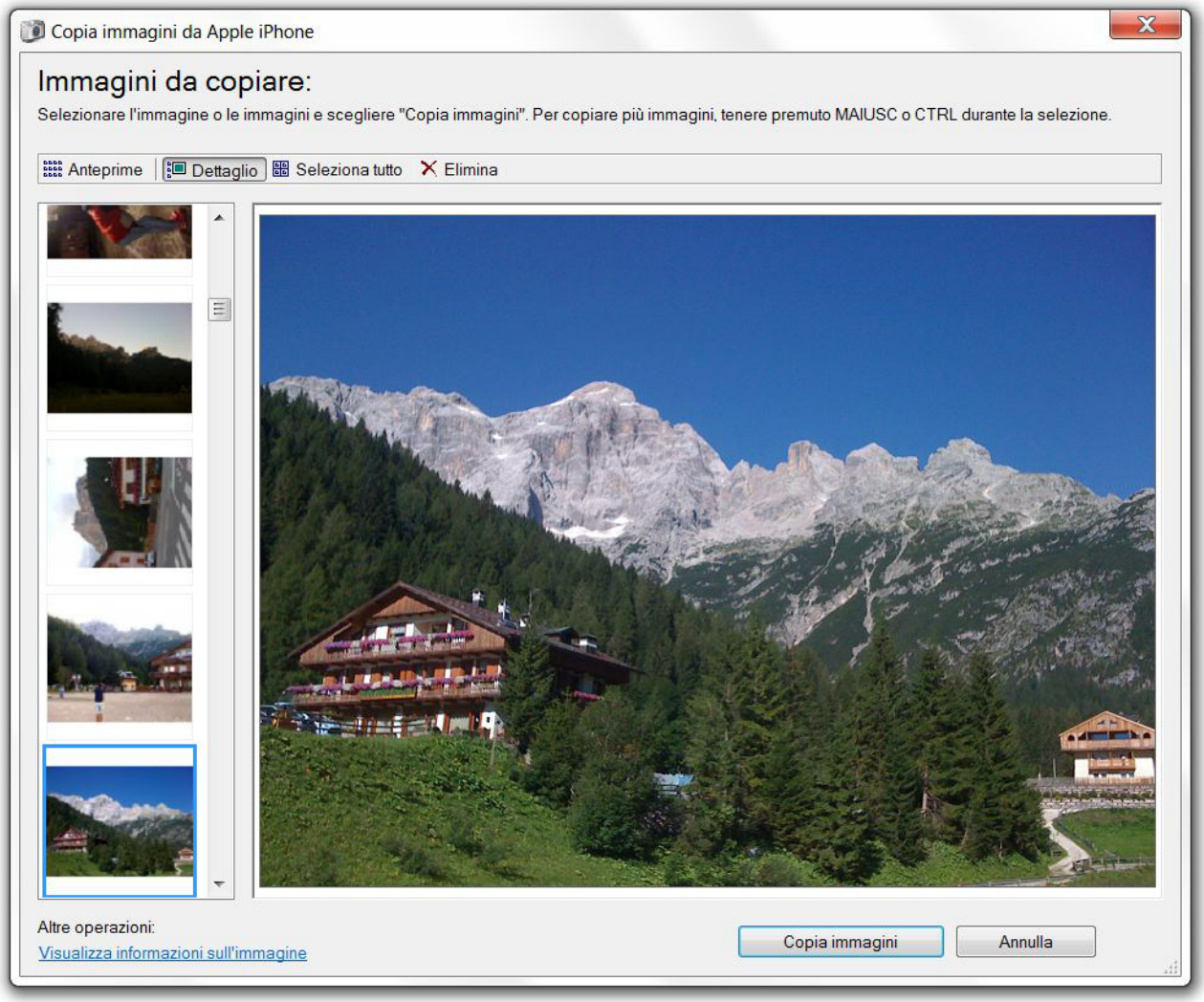


Figura 3.5: Dettaglio dell'interfaccia utente per la selezione di un file da iPhone

Com'è possibile notare dai segmenti di codice riportati, l'utilizzo della libreria JTwain non è particolarmente complesso; tuttavia si deve prestare attenzione alla gestione del SourceManager: se ci si dimentica di porre nel blocco `finally` il comando `SourceManager.closeSourceManager()`, si manda in *out of memory* la JVM che provoca una chiusura anomala dell'applicazione.

Capitolo 4

CMS, ECM ed Alfresco

Come già detto, il progetto a cui ho lavorato ha come obiettivo quello di inviare i documenti scansionati ad un sistema documentale (nella fattispecie Alfresco) per essere poi elaborati dagli operatori di back office. Seguirà ora un'introduzione sulle soluzioni di CMS ed ECM ed una descrizione di quello che è considerato il sistema documentale open source più conosciuto ed utilizzato, Alfresco.

Il *Content Management System* (CMS) è un sistema software (lato server) essenziale per l'azienda che voglia implementare un progetto di *knowledge management*[1]. È un insieme di tecniche, di comandi, di metodi che consentono di automatizzare la raccolta, la gestione, la ricerca e la pubblicazione di contenuti aziendali (qualsiasi sia il tipo a cui essi appartengono). In particolare i Content Management System possono "controllare" i processi di creazione e distribuzione delle informazioni e, di conseguenza, sono in grado di conoscere il valore delle informazioni ed a chi esse sono destinate. In generale i CMS sono considerati sistemi che gestiscono quei contenuti che necessitano di una pubblicazione ed una consultazione online, che sia Internet o una Intranet aziendale. Esistono poi CMS creati appositamente per un particolare tipo di contenuto (un'enciclopedia on-line, un blog, un forum) ed altri che sono invece più generici e flessibili e consentono la pubblicazione di tipi differenti.
I CMS più evoluti sono composti da tre specifici "moduli", che offrono ciascuno una propria particolare potenzialità: la raccolta, la gestione e la pubblicazione dei

[1]Knowledge Management: termine coniato nel 1986 da Karl Wiig, che indica una varietà di pratiche e di strategie usate all'interno di un'organizzazione per migliorare la collaborazione e l'efficienza dei gruppi di lavoro. In particolare, mediante un sistema di condivisione della conoscenza e dell'esperienza che ogni membro del gruppo ha maturato negli anni, si mira ad aumentare la produttività di ciascun componente a beneficio dell'intera organizzazione

contenuti. La **raccolta** costituisce la prima fase del sistema ed incorpora le azioni di creazione ed acquisizione delle informazioni; una volta acquisite devono essere convertite in un formato "universale" (ad esempio XML) ed associate ad alcuni metadati che ne facilitino l'organizzazione, l'archiviazione e la ricerca. La **gestione** dei contenuti, successiva alla raccolta, avviene attraverso appositi archivi (digitali) in cui vengono memorizzati i file; infine il **sistema di pubblicazione** permette l'impiego delle informazioni raccolte per qualsiasi tipo di pubblicazione (newsletter, documenti pronti per la stampa, in siti web).
I motivi che spingono un'organizzazione ad adottare un CMS sono fondamentalmente due:

- la grande quantità di informazioni da gestire, che altrimenti andrebbero analizzate e processate manualmente (con conseguente aggravio nei costi di gestione);
- il numero di persone coinvolte nel processo di creazione e della gestione dei contenuti. Qualora infatti la creazione dei contenuti fosse gestita da un unico "autore", esso tenderebbe ad avere un proprio stile di immagazzinamento delle informazioni. Nel caso in cui i contenuti siano invece creati da molte persone, diventa necessario adottare una serie di regole e di formalizzazioni. Un CMS permette di applicare con semplicità tali vincoli, consentendo all'azienda di decidere quali sono le regole che il sistema deve adottare e che di conseguenza ciascun autore deve rispettare.

Accanto ai CMS troviamo poi gli Enterprise Content Management (ECM); spesso questi due termini vengono confusi, ma presentano caratteristiche differenti che ne permettono la distinzione. Una soluzione di ECM, infatti, è un insieme di tecnologie che consentono la gestione della documentazione prodotta e ricevuta all'interno di un'organizzazione, indipendentemente dal suo formato. Più che ad aspetti software, esso si riferisce ad una strategia di creazione, acquisizione, salvataggio, gestione, conservazione e condivisione dei contenuti aziendali, senza avere un particolare sistema (software) che aiuti in questo compito.
Spesso si fa un distinguo tra informazioni strutturate e non strutturare. Le informazioni strutturate consistono in dati che si ripetono in una forma sempre identica o quasi; essi mantengono la loro struttura anche dopo lunghi periodi di tempo. Le informazioni strutturate possono essere gestite in maniera efficiente nei database relazionali.
Molti processi aziendali invece trattano con dati che non sono strutturati, con-

tenuti in documenti audio, video o in immagini; dati sotto forma di linguaggio naturale, di pixel, di campioni audio o di frame video sono più difficili da processare: dovrebbero essere categorizzati in maniera efficiente per poter essere ricercabili e ritrovabili anche dopo molti anni. I sistemi ECM curano la gestione proprio di questi tipi di informazioni.
Le aziende, col passare degli anni, hanno compreso come la propria ricchezza non derivi dalla quantità delle informazioni che circolano al loro interno, quanto dalla capacità di fruirne in maniera efficiente. L'automazione dei processi di gestione dei contenuti e la loro integrazione con i diversi sistemi di business consentono di ridurre tempi e costi, determinando una notevole crescita dell'efficienza aziendale.
Una soluzione di ECM permette quindi di catalogare, organizzare, archiviare, condividere e spedire ogni tipologia di documento; grazie agli ECM tutti gli utenti avranno disponibile, in qualsiasi momento ed in qualunque luogo, l'ultima versione aggiornata in tempo reale del documento e, se autorizzati, potranno revisionarla o modificarla. Tale processo porta notevoli benefici all'interno di un'organizzazione: una maggior efficienza e tempestività delle informazioni, un abbattimento dei costi e la riduzione dei tempi di ricerca, con un impatto positivo sul soddisfacimento della clientela.

Come naturale evoluzione ed integrazione tra questi due CM, si è assistito alla nascita degli Enterprise Content Management System (ECMS) per la gestione documentale. I software ECMS sono dei prodotti creati con lo scopo di integrare le caratteristiche dei CMS a quelle degli ECM e quindi di realizzare uno strumento che offra, in un unico sistema, un semplice utilizzo delle funzioni quali la cattura, l'organizzazione, l'indicizzazione, la conservazione, la trasformazione e la pubblicazione di qualsiasi tipo di contenuto e di documento riguardante i processi interni alle aziende. Il primo ed immediato vantaggio offerto è quello di consentire una totale (o quasi) sostituzione di un archivo fisico/cartaceo con uno digitale (garantendo così anche un notevole risparmio in termini economici); inoltre una corretta gestione facilita la ricerca dei documenti al suo interno.
Tra i molti ECMS esistenti spicca **Alfresco**, che è riconosciuto come il Software di ECM/gestione documentale più avanzato tra quelli disponibili sul mercato e, dettaglio non trascurabile, *open source*. Nato nel 2005, il gruppo di lavoro di Alfresco possiede un'esperienza nel settore della gestione documentale davvero notevole; basti pensare che il team di sviluppo è diretto da persone di assoluta competenza, con esperienze in progetti di prim'ordine quali *Documentum*, *Interwoven*, *FileNet*,

OpenText e *Vignette* (importanti soluzioni di CM). Alfresco si pone in opposizione alle tradizionali tecnologie documentali proprietarie: queste sono soluzioni molto costose, estremamente complesse e spesso poco personalizzabili, che ne limitano il controllo da parte del cliente. Il supporto di Alfresco agli standard open source ed aperti come CMIS (Content Management Interoperability Services) fa di Alfresco la piattaforma adatta a supportare la gestione del documentale aziendale. Alfresco, nella sua versione Enterprise (a pagamento) è utilizzata da oltre 2.500 aziende distribuite su 55 Paesi diversi. La piattaforma Alfresco 4 (l'ultima versione rilasciata) è stata testata in ambienti lavorativi con più di 100 milioni di documenti e con decine di migliaia di utenti. I benefici dell'utilizzo di Alfresco si possono riassumere nei seguenti punti:

- facilità di utilizzo;
- elevata produttività nello sviluppo delle soluzioni;
- benefici derivati da collaborazione grazie a forum e template pre-configurati;
- ricerche avanzate/Knowledge Management;
- architetture distribuite;
- open source.

Andando più nel dettaglio, voglio analizzare l'architettura di Alfresco: si presenta come una web application Java, pertanto è eseguibile su tutti i tipi di piattaforma; per la propria esecuzione Alfresco fa uso di un container *Tomcat*[2], di un database relazionale *PostgreSQL*[3] e della jdk versione 1.6. Per la comunicazione con Alfresco sono supportati diversi protocolli, CIFS (Common Internet File System), FTP, HTTP. Per le personalizzazioni Alfresco espone, oltre alle proprie API, una serie di servizi *SOAP*[4] o *REST*[5] modificabili a seconda delle proprie necessità.

[2]Tomcat: è un contenitore servlet open source che fornisce una piattaforma per l'esecuzione di web application Java. È sviluppato dalla Apache Software Foundation.

[3]PostgreSQL: è un database relazionale rilasciato con licenza BDS.

[4]Simple Object Access Protocol: è un protocollo leggero per lo scambio di messaggi tra componenti software.

[5]Representational State Transfer: è un insieme di principi di architetture di rete, che determinano come le risorse siano definite ed indirizzate. Il termine è spesso usato per descrivere ogni semplice interfaccia che trasmette dati su protocollo HTTP senza un livello opzionale come SOAP o la gestione della sessione tramite i cookie.

La gestione dei contenuti avviene utilizzando una serie di funzioni indispensabili per un ECM, come:

- *attributi* o *tag* per la classificazione dei documenti;
- *versionamenti* per tener traccia delle modifiche effettuate;
- *lock* e *unlock* nel momento in cui si va a modificare un documento o si rilascia un documento dopo una modifica;
- *full text search* per la ricerca di testo in documenti contenuti nei database.

All'interno del lavoro di tesi, Alfresco ha assunto il ruolo di back-end documentale. Sarebbe stato possibile in alternativa creare un database relazionale ed organizzare manualmente il file system, realizzando in questo modo un documentale del tutto personalizzato, ma ciò avrebbe portato ad inutili sforzi (ed un impiego di tempo non indifferente) data la presenza di un sistema ECM come Alfresco che, nella sua versione CE (Community Edition), è totalmente gratuito ed implementa tutti i servizi necessari al conseguimento del mio progetto.
La sua installazione è estremamente semplice e non presenta passaggi complicati: è sufficiente scaricare l'installer dal sito `http://wiki.alfresco.com/wiki/Download_and_Install_Alfresco` ed avviarlo sulla propria macchina. L'unica personalizzazione a cui si deve prestare attenzione è il setting della porta su cui Alfresco esporrà i propri servizi. Quella di default è la numero 8080: è necessario assicurarsi, quindi, che non vi siano altri servizi attivi su tale porta. Se si fosse costretti a cambiare porta, è sufficiente modificare il file *server.xml* contenuto nella cartella di configurazione di Tomcat.

4.1 Lo standard CMIS

Per inviare il documento digitalizzato al sistema documentale sono stati applicati due diversi approcci. La prima via intrapresa è stata quella di avvalersi delle librerie Java **OpenCMIS**, il cui scopo è rendere lo standard CMIS semplice da implementare per gli sviluppatori Java. La seconda via sarà invece esplicata in 4.3.
CMIS (Content Management Interoperability Services) è uno standard ideato per rendere più efficiente l'interoperabilità e la comunicazione tra sistemi ECM differenti. Le specifiche CMIS (documentate all'indirizzo web `http://docs.oasis`

`open.org/cmis/CMIS/v1.0/cs01/cmis-spec-v1.0.html`) forniscono un'interfaccia Web service che è indipendente dal tipo di linguaggio utilizzato (*language agnostic*), poiché REST e SOAP sono implementate in molti linguaggi di programmazione. CMIS presenta tre casi d'uso fondamentali:

- **Repository-to-Repository** (R2R): accade quando due repository cercano di comunicare direttamente l'uno con l'altro. Ciò accade in particolare quando si opera una gestione centralizzata dei record o quando devono essere pubblicati dei contenuti da un repository ad un'altro. Un esempio può essere il caso in cui si si voglia pubblicare un contenuto presente in un ECM direttamente su un WCM (Web Content Management) per la pubblicazione sulla rete Intra/Internet.
- **Application-to-Repository** (A2R): avviene se un'applicazione che utilizza contenuti è collegata ad un repository in grado di gestirli. È il caso implementato nel progetto e verrà illustrato in maniera approfondita nel prosieguo dell'elaborato.
- **Federated Repositories** (FR): è il caso in cui un'applicazione, con una singola interfaccia, si "rapporta" con differenti repository. Avviene ad esempio nel caso in cui sia effettuata una *Federated Search*, ovvero una ricerca orientata a più repository.

4.1.1 Il Modello Concettuale (Domain Model)

Il modello concettuale dello standard CMIS definisce i seguenti elementi:

- **Oggetti**: rappresentano le entità contenute nei repository. Ogni oggetto appartiene ad una categoria (o tipo, in totale sono 4): Cartelle, Documenti, Relazioni e Policy. Ciascun oggetto è identificatoo da un ID univoco e può essere associato ad una serie di proprietà (definite dal tipo di oggetto. In aggiunta alle proprietà, gli oggetti posseggono altri attributi (Versionable, Fileable, Queryable) che ne identificano il comportamento all'interno del repository.
  - *Documenti*: rappresentano le entità per cui viene utilizzato un repository. I documenti sono gli unici oggetti che godono della proprietà *Versionable* ed il cui valore può essere interrogato attraverso CMIS;
  - *Cartelle*: sono dei contenitori utilizzati per organizzare i documenti all'interno del repository. Ciascuna cartella ha un percorso, automaticamente generato, che la identifica all'interno del repository; può essere

definita in modo tale che sia limitata a contenere solo oggetti di un tipo specifico;

- *Relazioni*: definiscono relazioni tra due oggetti (source e target) appartenenti al repository. Modificando la relazione, non si apportano modifiche agli oggetti sorgente ed obiettivo. Gli oggetti relazione sono opzionali per le repositori CMIS-compliant;
- *Policy*: anch'essi sono oggetti opzionali che possono esser applicati ad oggetti *controllabili* e definiscono criteri amministrativi sulla gestione degli stessi. Una policy non è direttamente modellata dallo standard CMIS e può esser applicata a più file, cosiccome un file può godere di più di una policy. Una policy applicata ad oggetti controllabili non può esser cancellata. Per esempio, una *Lista di Controllo degli Accessi* (ACL) è un oggetto di tipo policy;

- **Proprietà**: le proprietà sono valori associati a ciascun oggetto. Le proprietà possono essere a valore singolo o multi valore e ciascuna di esse è di uno specifico tipo (testo, numero intero, ecc...). Ci sono proprietà che possono essere solo lette, altre che possono essere aggiornate in particolari intervalli temporali.

- **Controllo degli Accessi**: è utilizzato per specificare chi (e con che tipo di permessi) può trattare un particolare oggetto del repository. Se il repository lo supporta, allora la Lista di Controllo degli Accessi è applicata a ciascun oggetto. CMIS definisce tre tipi di permesso: lettura, scrittura, ed entrambi;

- **Change Log**: il repository può mantenere un log opzionale contenente una voce per ogni modifica effettuata ai contenuti del repository. Le voci del log includono l' identificativo dell'oggetto modificato ed il tipo di modifica (creazione, aggiornamento, cancellazione).

4.1.2 I Servizi

CMIS offre all'utente una serie di servizi che possono consentire alcune funzionalità opzionali, quindi non supportate da tutti i sistemi documentali. Tra questi troviamo:

- **servizi di repository**
  - *Get Repositories*: per ottenere la lista dei repository accessibili mediante quel particolare servizio;

  - *Get Repository Info*: per ottenere informazioni riguardo un repository specifico;
  - *Get Type Children, Get Type Descendant*: comandi per conoscere il tipo di oggetto in un repository;
  - *Get Type Definition*: comando utilizzato per ottenere una lista di proprietà riguardanti un tipo di oggetto specifico;

- **servizi di navigazione**
  - *Get Folder Tree, Get Descendants, Get Children*: per recuperare i riferimenti agli oggetti figlio di un particolare oggetto specificato;
  - *Get Folder Parent, Get Object Parents*: per ottenere il riferimento alla cartella padre di un oggetto;

- **servizi di ricerca**
  - *Query*: esegue una query CMIS;
  - *Get Content Changes*: ritorna una lista delle modifiche apportare al repository;

- **servizi sugli oggetti**
  - *Get Object, Get Object By Path*: ritorna un oggetto;
  - *Get Properties, Get Allowable Actions, Get Renditions*: per recuperare informazioni su uno o più oggetti;
  - *Create Relationship, Create Document, Create Document From Source, Create Policy, Create Folder*: comandi per creare oggetti;
  - *Update Properties, Move Object*: metodi per la modifica e l'aggiornamento dei documenti;
  - *Delete Object, Delete Tree*: esegue la rimozione degli oggetti specificati;
  - *Set Content Stream, Delete Content Stream*: aggiorna o cancella il contenuto dei documenti;

- **servizi di validazione**
  - *Get Properties Of Latest Version, Get Object Of Latest Version*: ottiene infromazioni riguardo l'ultima versione di un oggetto;

  - *Get All Versions*: recupera lo storico delle versioni di un oggetto;
  - *Check Out, Check In, Cancel Check Out*: controlla il lock e l'unlock di un oggetto con lo scopo di consentirne la modifica;
  - *Delete All Versions*: elimina lo storico delle versioni;

- vi sono infine ulteriori servizi (di Policy, di Relazione, di Controllo degli Accessi) nei quali si preferisce non addentrarsi, sia per non appesantire ulteriormente la trattazione del tema, sia perché sono servizi di cui non si è fatto uso durante il presente lavoro.

4.1.3 Le librerie OpenCMIS

OpenCMIS è il nome di un progetto *Apache Chemistry*, che mette a disposizione librerie, framework e tool per il programmatore Java che voglia implementare le specifiche CMIS all'interno del proprio applicativo. L'obiettivo di OpenCMIS è quello di rendere semplice da utilizzare lo standard CMIS nello sviluppo di applicazioni Java lato client e server; esso nasconde i dettagli implementativi e fornisce delle comode API e degli strumenti per la creazione di repository di contenuti e di applicazioni client.
Per la comunicazione con altri repository o con i client, CMIS definisce due strategie di *binding*[6]: l'Atom Publishing Protocol[7] (AtomPub o APP) ed i Web Services[8].
Essendo Alfresco una tecnologia CMIS-compliant, espone entrambi questi meccanismi: nel mio progetto ho testato l'implementazione del Web Services binding ed anche dell'AtomPub binding. Riporto il codice, leggibile ed intuitivo; i commenti suggeriscono il significato delle istruzioni utilizzate.

Listing 4.1: AtomPub binding

[6]Binding: un insieme di regole che definiscono come un terminale (sia esso un client o un server) comunichi con gli altri terminali. Un binding definisce il protocollo di trasporto (come ad esempio HTTP o TCP) e la codifica che dev'essere utilizzata (come l'alfabeto binario piuttosto che il testo); inoltre un binding può contenere elementi che specificano dettagli, quali i meccanismi di sicurezza usati per lo scambio dei messaggi oppure i modelli dei messaggi

[7]AtomPub: protocollo di livello applicazione che utilizza il livello di trasporto HTTP per la pubblicazione e la modifica di risorse Web

[8]Web Service: è un componente applicativo (software) in grado di mettersi al servizio di un applicazione (che si dice che "espone i servizi") comunicando su di una medesima rete tramite il protocollo di trasporto HTTP. Un Web Service consente quindi alle applicazioni che vi si collegano di usufruire delle funzioni che mette a disposizione.

```
1  SessionFactory factory = SessionFactoryImpl.newInstance();
2  Map<String, String> parameter = new HashMap<String, String>();
3
4  // credenziali utente
5  parameter.put(SessionParameter.USER, "admin");
6  parameter.put(SessionParameter.PASSWORD, "admin");
7
8  // parametri di connessione
9  parameter.put(SessionParameter.ATOMPUB_URL, "http
     ://127.0.0.1:8080/alfresco/service/cmis");
10 parameter.put(SessionParameter.BINDING_TYPE, BindingType.
     ATOMPUB.value());
11 parameter.put(SessionParameter.REPOSITORY_ID, "50308dd4-2e36
     -48e1-9703-a6c53875e822");
12
13 // creo sessione
14 Session session = factory.createSession(parameter);
```

Il parametro di sessione REPOSITORY_ID è un parametro univoco per ogni installazione di Alfresco, ne identifica l'ID del repository ed è individuabile tramite l'accesso alla seguente pagina Web `http://[host]:[port]/alfresco/cmis-browse?url=http://[host]:[port]/alfresco/cmisatom` che riporta una serie di informazioni riguardanti la propria installazione.
Come si può notare dal frammento di codice che verrà ora riportato, la differenza implementativa tra un AtomPub binding ed un Web Services binding risiede esclusivamente nella definizione dei parameti di connessione.

Listing 4.2: Web Services binding

```
1  SessionFactory factory = SessionFactoryImpl.newInstance();
2  Map<String, String> parameter = new HashMap<String, String>();
3
4  // credenziali utente
5  parameter.put(SessionParameter.USER, "admin");
6  parameter.put(SessionParameter.PASSWORD, "admin");
7
8  // parametri di connessione
9  parameter.put(SessionParameter.BINDING_TYPE, BindingType.
     WEBSERVICES.value());
10 parameter.put(SessionParameter.WEBSERVICES_ACL_SERVICE, "http
     ://localhost:8080/alfresco/cmisws/ACLService?wsdl");
```

```
11 parameter.put(SessionParameter.WEBSERVICES_DISCOVERY_SERVICE,
     "http://localhost:8080/alfresco/cmisws/DiscoveryService?wsdl"
     );
12 parameter.put(SessionParameter.WEBSERVICES_MULTIFILING_SERVICE
     , "http://localhost:8080/alfresco/cmisws/MultiFilingService?
     wsdl");
13 parameter.put(SessionParameter.WEBSERVICES_NAVIGATION_SERVICE,
      "http://localhost:8080/alfresco/cmisws/NavigationService?
     wsdl");
14 parameter.put(SessionParameter.WEBSERVICES_OBJECT_SERVICE, "
     http://localhost:8080/alfresco/cmisws/ObjectService?wsdl");
15 parameter.put(SessionParameter.WEBSERVICES_POLICY_SERVICE, "
     http://localhost:8080/alfresco/cmisws/PolicyService?wsdl");
16 parameter.put(SessionParameter.
     WEBSERVICES_RELATIONSHIP_SERVICE, "http://localhost:8080/
     alfresco/cmisws/RelationshipService?wsdl");
17 parameter.put(SessionParameter.WEBSERVICES_REPOSITORY_SERVICE,
     "http://localhost:8080/alfresco/cmisws/RepositoryService?wsdl
     ");
18 parameter.put(SessionParameter.WEBSERVICES_VERSIONING_SERVICE,
      "http://localhost:8080/alfresco/cmisws/VersioningService?
     wsdl");
19 parameter.put(SessionParameter.REPOSITORY_ID, 50308dd4-2e36-48
     e1-9703-a6c53875e822);
20
21 // creo una sessione
22 Session session = factory.createSession(parameter);
```

Per una più chiara ed approfondita analisi dei parametri di connessione, si rimanda alla pagina presente sul sito ufficiale del progetto OpenCMIS[9].

4.2 Realizzazione dell'invio dei file con lo standard CMIS

Il software da me sviluppato offre all'utente la seguente possibilità: una volta acquisita l'immagine da scanner, ne permette l'invio al documentale Alfresco (installato in locale solo per questioni puramente pratiche, ma sono stati rea-

[9] http://chemistry.apache.org/java/developing/dev-session-parameters.html

lizzati degli invii anche in repository Alfresco remoti). Riporto il metodo `void connectToAlfresco(String user, String pass, String repoID, String name, String dir)` che in input richiede username e password per l'autenticazione della connessione, l'ID del repository Alfresco, il nome ed il path del file da inviare nella cartella di root del repository (nella fattispecie si tratta di un file d'immagine con estensione *.tiff*).

Listing 4.3: Il metodo public void connectToAlfresco(String user, String pass, String repositoryID, String name, String dir)

```
1  public void connectToAlfresco(String user, String pass, String
       repoID, String name, String dir){
2      Session session = Tools.createSession(user, pass,
         repositoryID);
3
4      // creo ContentStream
5      ContentStream contentStream = null;
6      try {
7    contentStream = new ContentStreamImpl(name, null, "image/
       tiff", new FileInputStream(dir));
8      }
9      catch (FileNotFoundException e) {
10   logger.error("connectToAlfresco(String, String) -
       FileNotFound", e);
11     }
12     if (contentStream != null) {
13   Map<String, Object> properties = new HashMap<String, Object
       >();
14   properties.put(PropertyIds.NAME, contentStream.getFileName()
       );
15   properties.put(PropertyIds.OBJECT_TYPE_ID, "cmis:document");
16
17   //creo un documento nella root folder con le proprietà ed il
        ContentStream specificati
18   Document doc = session.getRootFolder().createDocument(
       properties, contentStream , null);
19     }
20 }
```

In caso si volesse invece creare o inviare un file in una cartella diversa dalla root folder, il seguente frammento di codice mostra come creare una nuova car-

tella.

Listing 4.4: OpenCMIS - Creazione di una cartella

```
1  //Root Folder
2  Folder root = session.getRootFolder();
3
4  // proprietà
5  // nome e id del tipo di oggetto
6  Map<String,Object> properties = new HashMap<String,Object>();
7  properties.put(PropertyIds.OBJECT_TYPE_ID, "cmis:folder");
8  properties.put(PropertyIds.NAME, "a new folder");
9
10 // creo la cartella
11 Folder childrenFolder = root.createFolder(properties);
```

L'utilizzo delle librerie OpenCMIS non è stato particolarmente impegnativo, anche se bisogna ammettere che ci si è serviti di un range ristretto di funzionalità. Nella prossima sezione, si vedrà come effettuare l'invio al documentale Alfresco di un file tramite il classico comando *POST* del protocollo *HTTP*.

4.3 La libreria apache-commons HttpClient

L'*Hyper-Text Transfer Protocol* è il protocollo di livello applicazione più utilizzato in Internet al giorno d'oggi. Lo sviluppo degli standard HTTP è stato coordinato dall'*Internet Engineering Task Force* (IETF) e dal *World Wide Web Consortium (W3C)* ed è culminato nella pubblicazione di una serie di *Request For Comments* (RFCs), in particolare la RFC 2616 (giungo 1999), che definisce la versione di HTTP comunemente in uso (HTTP/1.1). HTTP funziona con un meccanismo di richiesta e risposta: il client esegue una richiesta ed il server restituisce la risposta; nell'uso comune il client corrisponde al browser ed il server al sito web.
Nonostante le classi del *package java.net* forniscano le funzionalità base per accedere a risorse via HTTP, queste non garantiscono la massima flessibilità o le funzionalità richieste da molte applicazioni. Il componente **HttpClient**[10], appartenente al progetto *Apache HttpComponents*, cerca di colmare queste lacune offrendo un insieme di soluzioni efficienti, aggiornate e ricche di funzionalità che

[10] http://hc.apache.org/httpcomponents-client-ga/tutorial/html/index.html

implementino il lato client dei più recenti standard HTTP e delle raccomandazioni W3C.

Per inviare il documento scansionato al sistema documentale, sono stati utilizzati due approcci. Il primo, illustrato in 4.1, si avvale dello standard CMIS e fa uso delle librerie Java *OpenCMIS*; il secondo invece sfrutta il protocollo HTTP e l'interazione con i Web service REST messi a disposizizone da Alfresco (ed appositamente modificati per soddisfare le esigenze del mio programma). Vediamo quindi come è stato possibile realizzare l'invio di un file d'immagine TIFF al sistema ECM Alfresco.
Per prima cosa, si è dovuto modellare il Web service perché accettasse richieste POST in una forma "particolare". Questo lavoro è stato svolto da un altro tesista (che si è occupato esclusivamente degli aspetti dedicati ad Alfresco, alla modellizzazione del repository e dei processi di acquisizione e gestione dei documenti), che mi ha fornito quindi le "specifiche" per l'invio di un file TIFF tramite il comando HTTP **POST**. POST è uno dei metodi supportati dal protocollo HTTP (gli altri sono GET, HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT) ed è utilizzato quando un client necessita di inviare ad un server alcune informazioni (o nei casi di trasferimento di file); queste informazioni sono contenute nel "corpo del messaggio" e possono essere di lunghezza arbitraria. L'*header* della richiesta di POST deve quindi indicare al server il tipo di oggetto che si intende trasferire e lo fa specificandolo nel campo *Content-Type*. Nello specifico del mio lavoro, il *Content-Type* è di tipo "*multipart/form-data*". La POST HTTP è strutturata per creare un'autenticazione di "base"[11] con Alfresco e per inviare, insieme al file d'immagine, le seguenti informazioni:

- *submitterId*: il codice identificativo dell'operatore;
- *submissionId*: codice (univoco) di trasferimento del file;
- *submissionDate*: la data e l'ora di trasferimento, nel formato 2011-11-12T10:39:55.369Z;
- *submitterPositionId*: il codice della postazione dell'operatore che ha effettuato il trasferimento del file;

[11] L'**autenticazione di base** fa parte della specifica HTTP 1.0. Durante l'autenticazione di base il server richiede al client un nome utente e una password, che trasmette tali informazioni tramite HTTP utilizzando la codifica Base64. Per informazioni su tale algoritmo di codifica, si veda `http://www.faqs.org/rfcs/rfc3548.html`

- *cro*: Codice Riferimento Operazione per mezzo del quale ogni istituto bancario identifica in maniera univoca ciascuna transazione;

Riporto parte del codice utilizzato per effettuare il trasferimento del file scansionato; il codice è facilmente comprensibile e non necessita, a mio avviso, di molti commenti. Mi preme sottolineare quelli che sono i passaggi chiave per la creazione del comando POST mediante la libreria *HttpClient*:

1. creazione dell'oggetto *HttpClient*;
2. creazione dell'oggetto *HttpPost*, con specificato il parametro del URL verso cui eseguire la richiesta di POST;
3. codifica delle credenziali d'accesso ed aggiunta di queste all'header del messaggio;
4. creazione di una MultipartEntity, a cui vanno associati il file da inviare e le relative infromazioni;
5. aggiunta dell'oggetto MultipartEntity al *content body* della richiesta di POST;
6. invio della richiesta di POST;

Listing 4.5: Struttura base di un documento JNLP

```
1  DefaultHttpClient httpclient = new DefaultHttpClient();
2
3  HttpPost httppost = new HttpPost("http://192.168.0.10:8080/
     alfresco/service/up-ct-tiff");
4
5  //codifica delle credenziali d'accesso
6  String encoding = Base64.encodeString("admin:pass");
7  httppost.setHeader("Authorization", "Basic "+encoding);
8
9  File file = new File("C:/Users/Michele/Desktop/file.tiff");
10
11 MultipartEntity mpEntity = new MultipartEntity();
12 ContentBody cbFile = new FileBody(file, "image/tiff");
13 mpEntity.addPart("file", cbFile);
14 mpEntity.addPart("submitterId", new StringBody("F1111111"));
15 mpEntity.addPart("submissionId", new StringBody("45678"));
```

```
16 mpEntity.addPart("submissionDate", new StringBody("2012-01-20
     T10:05:56.369Z"));
17 mpEntity.addPart("submitterPositionId", new StringBody("1003")
     );
18 mpEntity.addPart("cro", new StringBody("4453764"));
19
20 httppost.setEntity(mpEntity);
21
22 HttpResponse response = httpclient.execute(httppost);
```

Capitolo 5

La tecnologia Java Web Start

Uno tra gli aspetti fondamentali nella realizzazione di progetti software è quello che riguarda la strategia di vendita. Accanto alla classica distribuzione su supporto fisico (CD o DVD), sta prendendo sempre più piede, col passare degli anni, la vendita di software tramite Internet, attraverso il download di file d'installazione. Questo cambiamento è stato determinato dal numero sempre maggiore di prodotti disponibili e di produttori, ma anche da una questione di opportunità: i costi di distribuzione sono notevolmente inferiori e si può raggiungere un bacino d'utenza decisamente più ampio.
In questo scenario si inserisce la strategia pensata per la commercializzazione del software realizzato nel mio progetto: Java Web Start. Tramite questo comodo strumento è possibile scaricare il programma dal Web ed eliminare le noiose procedure di installazione ed aggiornamento, con la certezza di eseguire sempre l'ultima release disponibile del programma.

Java Web Start[1] (nota anche come JavaWS o JWS) è una tecnologia di distribuzione di applicazioni sviluppata dalla *Sun Microsystems* (ora rilevata da *Oracle*), che consente all'utente di scaricare ed avviare applicativi software Java direttamente da un web server, quindi avvalendosi di tre soli strumenti: una macchina con installata la piattaforma Java SE, una connessione ad internet ed un web browser. Con Java Web Start è possibile scaricare ed avviare un'applicazione senza dover eseguire le classiche (e noiose) procedure di installazione. Il software Java Web Start si avvia automaticamente quando viene scaricata per la prima volta un'applicazione che faccia uso della tecnologia JWS; Java Web Start si occupa di memorizzare localmente (nella *cache*) l'intera applicazione: ogni ul-

[1] La prima versione di Java Web Start, la 1.0, è datata marzo 2001.

teriore avvio avviene in modo pressoché immediato, dal momento che tutte le risorse necessarie sono già disponibili sulla propria macchina. Inoltre ogni volta che si avvia l'applicativo, il componente software Java Web Start controlla il sito Web dell'applicazione per verificare se è disponibile una nuova versione e, in tal caso, la scarica e l'avvia automaticamente.

Dal punto di vista delle opportunità, Java Web Start presenta un buon numero di caratteristiche che lo rendono particolarmente interessante per la distribuzione delle applicazioni. Infatti:

- è realizzato per eseguire esclusivamente applicazioni scritte per la piattaforma Java SE. Una singola applicazione può così essere disponibile su un Web server e poi distribuita su un ampio ventaglio di piattaforme: da Windows a Linux, da Mac a Solaris;

- supporta diverse versioni della piattaforma Java SE. Se un'applicazione richiede quindi una particolare release della JSE non presente nella macchina su cui viene eseguita, questa verrà scaricata ed installata automaticamente. Con JWS è possibile eseguire diverse applicazioni contemporaneamente, che richiedono versioni della piattaforma Java differenti, senza causare conflitti;

- permette agli applicativi di venir eseguiti anche in modalità off-line (tuttavia almeno la prima volta devono essere scaricati da un sito Internet). Questi possono essere lanciati anche da collegamenti presenti su desktop, come quando si vuole eseguire un qualsiasi programma;

- aggiorna automaticamente le applicazioni, grazie ad un sistema di controllo versione dei file;

- implementa le componenti di sicurezza della piattaforma Java SE, cosicché dati e file presenti sulla propria macchina non vengono mai esposti a rischi. Le applicazioni vengono eseguite in un ambiente protetto, la *sandbox*[2], con un accesso ristretto al disco locale ed alle risorse di rete: in questo modo l'utente può eseguire applicazioni provenienti anche da fonti non sicure. Un'applicazione può anche richiedere un'esecuzione "non limitata" (è il caso

[2]Una **sandbox** è un meccanismo di sicurezza per l'esecuzione di programmi ritenuti "non affidabili". La sandbox offre un ambiente protetto ed un insieme di risorse strettamente controllato e le applicazioni che vi vengono eseguite hanno pesanti limitazioni: l'accesso alla rete, la capacità di recuperare informazioni di sistema o la lettura di input da periferiche sono disabilitate o comunque estremamente limitate.

delle applicazioni *firmate*) per accedere a tutte le risorse della macchina su cui è eseguita; in questo caso comparirà una finestra di avviso riportante le informazioni sul produttore che ha sviluppato l'applicazione: se si darà la conferma, il programma verrà eseguito.

Java Web Start è incluso in *Java Runtime Environment* (JRE) (a partire dalla versione 5.0) pertanto, installando Java, viene installato automaticamente anche Java Web Start.
La tecnologia che sta alla base della Java Web Start è la *Java Network Launching Protocol and API* (cui ci si riferisce con l'acronimo JNLP). La JNLP definisce, tra le altre cose, un formato di file standard che descrive come lanciare un'applicazione, ovvero un file con estensione *.jnlp*. Con il software Java Web Start quindi l'utente può eseguire un'applicazione cliccando un collegamento in una pagina web: tale link punta ad un file JNLP, che incarica JWS di scaricare, salvare ed avviare il programma.

Sviluppo di un'applicazione Java Web Start

Questi sono i passi chiave per lo sviluppo di un'applicazione che si preveda di rendere fruibile mediante Java Web Start:

1. compilare il codice sorgente e crearne il relativo pacchetto JAR[3];
2. creare un file JNLP ad-hoc per la propria applicazione. Le applicazioni Java infatti vengono lanciate attraverso il Java Network Launch Protocol (JNLP);
3. creare una pagina internet con un collegamento al proprio file JNLP.

Verranno di seguito analizzati, singolarmente, strumenti e procedure utilizzate per la realizzazione di un'applicazione distribuibile con tecnologia Java Web Start.

[3]JAR: acronimo di *Java Archive*, è un archivio compresso tipicamente utilizzato per raggruppare classi Java, metadati e risorse (testo, immagini ecc.) ad esse associate e librerie di terze parti in un unico file, per distribuire un'applicazione software su piattaforma Java.

5.1 Creazione di un file *.jar*

Per realizzare un'applicazione Java Web Start è necessario, prima di tutto, aver scritto un programma Java. Una volta compilato il programma, si deve creare un archivio JAR che contenga tutto ciò di cui il programma ha bisogno: classi, librerie, eventuali file d'immagine e quant'altro. La creazione del file non è particolarmente ostica se effettuata mediante ambienti di sviluppo (Eclipse), merita un po' più d'attenzione se viene realizzata invece da riga di comando.
Si osservi un particolare: se il programma Java usa librerie di terze parti che non appartengono alla piattaforma standard, ci sono due strade per far sì che l'archivio venga eseguito correttamente. La prima consiste nell'installare i file necessari sulla piattaforma Java del sistema su cui vogliamo far girare il programma (copiando i JAR nella cartella *ext* del JRE), la seconda nell'inserimento delle librerie nel progetto. Con Eclipse questa procedura è molto semplice:

- si crea una cartella all'interno del progetto;
- si inseriscono nella cartella le librerie necessarie;
- si importano le librerie nel progetto, scegliendo l'opzione Properties –> Java Build Path –> Libraries –> Add JARs.

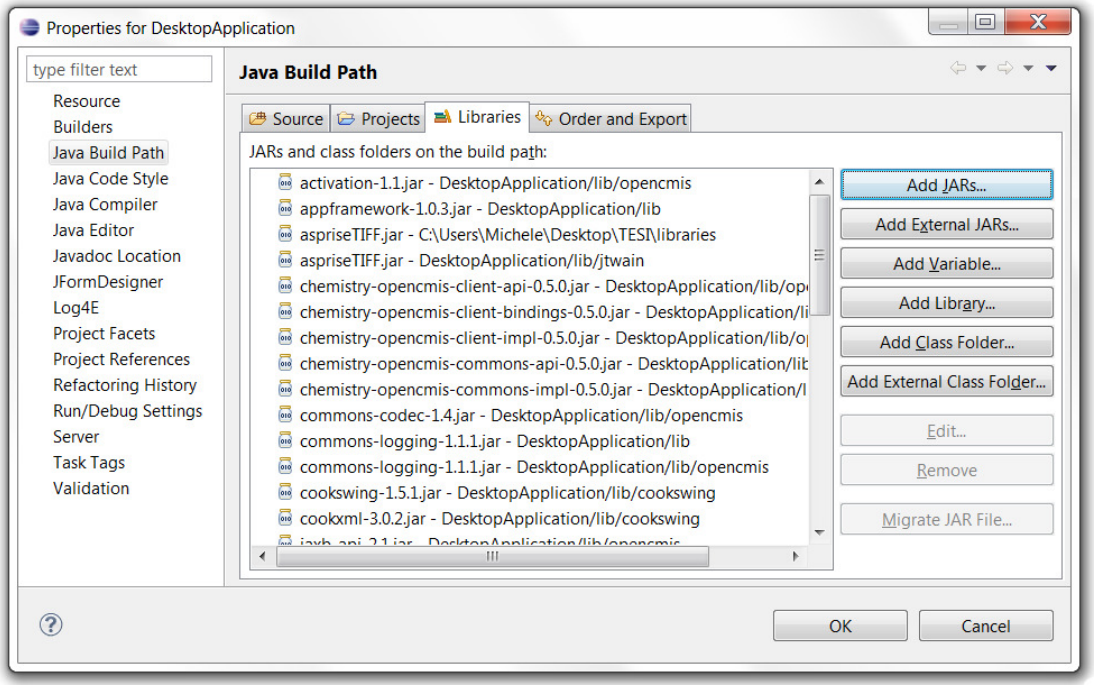


Figura 5.1: Librerie di terze parti importate nel progetto

5.1.1 Creazione di un file *.jar* da console

La procedura da console richiede che sia realizzato innanzitutto un file *manifest*[4] che contiene alcune informazioni riguardo le proprietà dell'archivio che si andrà a creare. Nel file manifest è indicato ad esempio il nome della classe che contiene il metodo `main` ed il *class path* delle librerie utilizzate dal programma. L'ultima riga del file manifest deve essere lasciata bianca o, in altre parole, si deve terminare il contenuto del file con un comando di *newline*.
Una volta creato il file MANIFEST.MF si può eseguire il comando:

```
jar cvfm MANIFEST.MF jar-file input-files
```

Le opzioni e gli attributi utilizzati sono i seguenti:

- l'opzione *c* indica che si vuole creare un file *.jar*;
- l'opzione *v* indica che si produrrà un output "verbose". Durante la creazione del *.jar* quindi, sulla console vengono visualizzati i nomi di tutti i file che sono aggiunti al pacchetto;
- l'opzione *f* indica che si vuole che l'output sia un file;
- l'opzione *m* permette di indicare il nome del file manifest. Se non si usa l'opzione *m*, viene creato un file manifest di default;
- *MANIFEST.MF* è il nome del file manifest associato al file JAR;
- *jar-file* è il nome che si vuole assegnare all'archivio JAR. Non è necessario aggiungere alla fine del nome il suffisso *".jar"* che verrà assegnato di default;
- *input-files* è il nome dei file, separati da uno spazio, che si vogliono includere nel pacchetto. E' possibile aggiungere all'archivio anche intere cartelle.

Terminata la creazione, per verificare la correttezza dell'archivio creato è sufficiente digitare il seguente comando:

```
java -jar jar-files.jar
```

e se il processo è andato a buon fine, partirà l'esecuzione del programma Java.

[4] è un file di testo non formattato, cui di solito viene assegnato il nome MANIFEST.MF

5.1.2 Creazione di un file *.jar* in Eclipse

Per creare il file JAR, *Eclipse* mette a disposizione un utile wizard: basta cliccare col pulsante destro del mouse sul nome del progetto, presente nel pannello di navigazione di sinistra (o in alternativa si seleziona il nome del progetto e poi si sceglie il menù *File*), scegliere la voce *Export*, poi *JAR file*. Ci si troverà di fronte alla seguente vista (fig. 5.2):

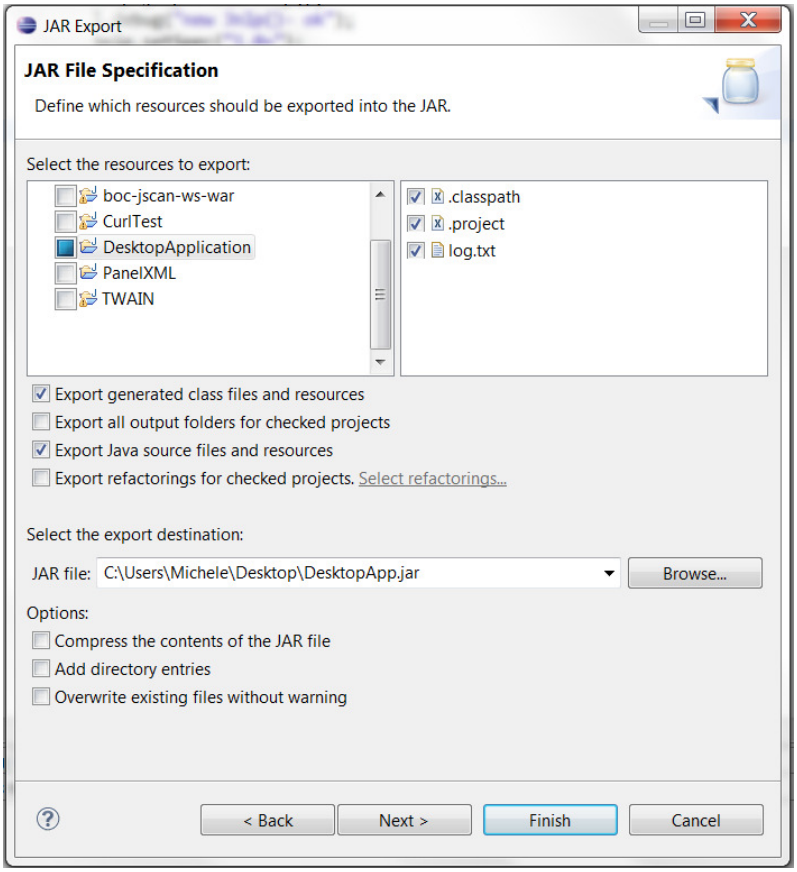


Figura 5.2: Personalizzazione file JAR - Scelta dei file da esportare

Si può ottenere quindi un archivio JAR eseguibile in pochi semplici passi. Per includere o escludere dei file dall'archivio, è possibile selezionarli dalla vista situata in alto nella finestra; nel caso si voglia allegare anche i file sorgenti del progetto, è necessario mettere il segno di spunta alla voce *Export Java source file and resources*.
Proseguendo con il wizard viene data inoltre la possibilità all'utente di utilizzare un file *manifest* generato in precedenza (al posto di quello automatico generato da Eclipse) e di inserire il nome della classe che contiene il metodo `main` del programma (Figura 5.3).

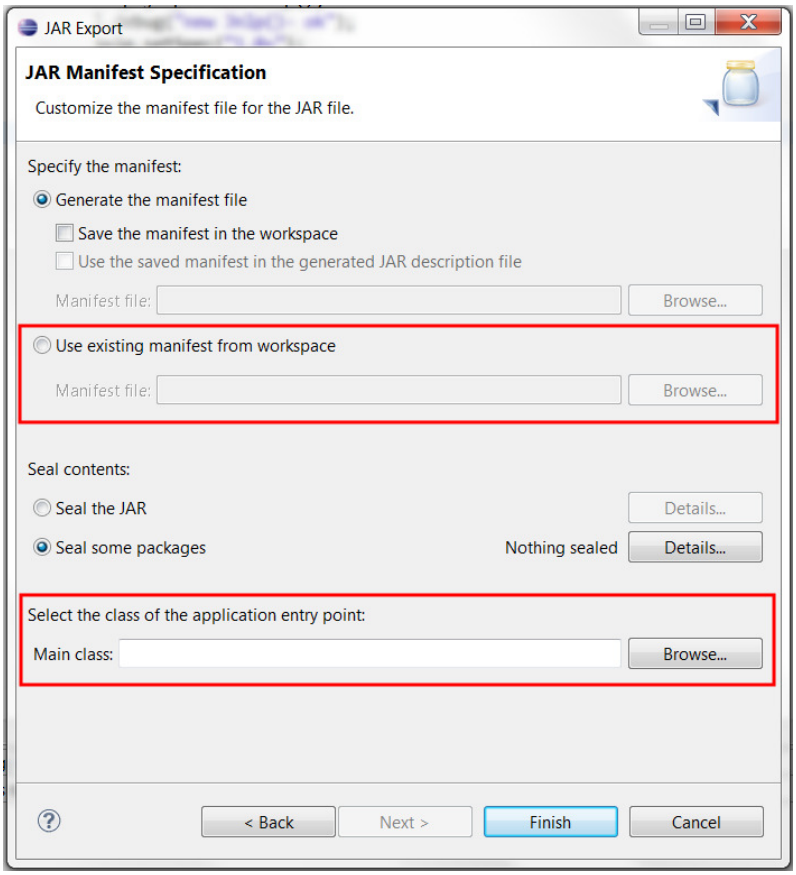


Figura 5.3: Personalizzazione del file JAR - Manifest e Main class

Al termine del processo di personalizzazione, premendo il tasto *Finish* verrà generato un archivio JAR nella posizione da noi scelta (figura 5.2, alla voce *select destination*), che conterrà i file specificati in precedenza.

5.2 Firma di un file *.jar*

Uno dei problemi senz'altro più sentiti, in ambito informatico, è quello della sicurezza. Per ovviare a questo tipo di problematiche, Java propone una serie di strumenti per la gestione di chiavi pubbliche e private, certificati e firme digitali. Questi concetti sono essenziali qualora si voglia distribuire un'applicazione sfruttando la tecnologia Java Web Start ed i motivi sono presto detti:

- si deve garantire all'utente l'affidabilità di un particolare software. Se l'autorità competente riconosce un'entità come "trusted", è probabile che l'u-

tente finale non abbia remore nell'utilizzare un'applicazione distribuita da quell'entità;

- come si vedrà in 5.3, Java Web Start esegue il software potenzialmente dannoso in un ambiente protetto, la *sandbox*, che ne limita però le potenzialità e l'accesso alle risorse. Perché l'applicazione abbia pieno accesso alla macchina su cui viene eseguita, deve essere riconosciuta come "affidabile".

Si procederà quindi con l'analisi degli strumenti necessari a firmare un archivio JAR e si vedrà poi nel dettaglio la procedura che porta alla creazione di un archivio firmato.

Il **keytool** è uno strumento per la gestione di *chiavi* e *certificati*. Esso permette all'utente di amministrare le proprie coppie *chiave pubblica - chiave privata* e di associare ad esse dei certificati con firma digitale, così da poterle utilizzare come certificazione di autenticità presso altri utenti/servizi o come garanzia di integrità dei dati. Inoltre il *keytool* consente all'utente di salvare le chiavi private ed i certificati degli altri utenti con cui si viene in contatto: questi vengono salvati nel *keystore*, un file che protegge chiavi private e certificati loro associate, per l'autenticazione della corrispondente chiave pubblica. Il *keystore* viene creato ogniqualvolta si usi il comando `keytool -genkey` oppure `keytool -import`; esso è implementato, come abbiamo già detto, come un file e se non ne viene specificato il nome e il path, viene memorizzato nel file di default *.keystore* presente nell'home directory dell'utente. Inoltre, il *keystore* protegge le chiavi private con delle password, che vengono impostate al momento della creazione.
Il *keystore* ha due tipi differenti di entry:

- **key entry**: contiene la chiave privata, memorizzata in un formato protetto per prevenire accessi non autorizzati, accompagnata dalla catena di certificati per la corrispondente chiave pubblica;

- **trusted certificate entry**: contiene il certificato con la chiave pubblica appartenente ad un'altra entità "affidabile". Questi certificati vengono chiamati "trusted certificate".

Tutte le entry del *keystore* sono accessibili mediante nomi univoci, gli *alias*, i quali vengono specificati nel momento della creazione di una coppia di chiavi (pubblica e privata) o dell'aggiunta di un certificato all'elenco dei certificati "affidabili".

Un **certificato** è un documento elettronico dotato di *firma digitale* che garantisce che una chiave pubblica appartenente ad un'entità (una persona, una società, ecc.), sia associata alla vera identità del soggetto che la rivendica come propria. In altre parole, un certificato è un "attestato" che garantisce un'associazione di tipo *chiave pubblica - entità*. Quando si ricevono dati firmati digitalmente,la firma funge da strumento di controllo per la verifica di integrità ed autenticità. Per *integrità* si intende che i dati non siano stati manomessi o modificati, *autenticità* significa invece che i dati provengono effettivamente dall'ente che afferma di aver creato e firmato tali dati.

Il **jarsigner** utilizza le informazioni di un *keystore* per verificare l'attendibilità degli archivi Java (i file *.jar*). Lo strumento *jarsigner* verifica le firme digitali dei file *.jar*, utilizzando i certificati che le accompagnano (contenute in file all'interno dell'archivio JAR) e poi controlla se la chiave pubblica di quel certificato sia "affidabile" o meno, cioé se sia contenuta nel *keystore* specificato.

Gli strumenti *keytool* e *jarsigner* sostituiscono il tool *javakey* fornito nel JDK 1.1. Questi due nuovi applicativi ampliano le potenzialità del *javakey*, includendo

- la possibilità di proteggere il *keystore* e le chiavi private mediante password;
- la possibilità di verificare le firme e di generare coppie di chiavi e certificati.

La nuova architettura del *keystore* rimpiazza il vecchio database che veniva creato e gestito dal *javakey*. È possibile comunque importare informazioni da un database di identità in un *keystore*, utilizzando il comando `keytool -identitylib`.

Per poter avere accesso completo alle risorse del client JNLP il file *.jar* distribuito deve essere "firmato". Nello specifico caso dell'applicazione che ho personalmente creato nel lavoro di tesi, la provenienza e l'identità di chi l'ha scritta è certa (il sottoscritto), pertanto per testare la tecnologia JWS è stato sufficiente utilizzare un certificato auto prodotto. Vediamo quali sono i passi da seguire per firmare un file con estensione *.jar* con i *security tool* messi a disposizione dalla JDK.

Una volta creato il file *DesktopApplication.jar* che contiene tutte le classi e le librerie necessarie al funzionamento del programma, si devono generare un *keystore* ed una coppia di chiavi. Per far questo, utilizziamo il seguenti comando:

Questo comando crea un nuovo keystore chiamato "mykeystore" e situato in "C:/Users/Michele" a cui viene assegnata la password "mykeystorepass"; se si fosse

```
keytool -genkey -dname ‘‘cn=Michele Pantano, o=E-project, st=PD
c=IT’’ -alias desktopApp -keypass desktopApp -validity 180
 -keystore C:/Users/Michele/mykeystore -storepass mykeystorepass
```

omessa questa voce, sarebbe stato utilizzato il keystore di default *.keystore* situato nell'home directory dell'utente). Inoltre viene generata la coppia di chiavi per l'entità il cui nome è "Michele Pantano", l'azienda è "E-project", la provincia è Padova e lo Stato è "IT"; le chiavi son generate con algoritmo di generazione *DSA* e lunghezza delle chiavi di 1024 byte.
Viene creato in aggiunta anche un certificato *auto firmato* (utilizzando l'algoritmo di firma "SHA1withDSA") che include la chiave pubblica e le informazioni riguardanti l'entità generatrice di tale chiave (quelle passate con il comando `-dname`). Questo certificato sarà valido per 180 giorni ed è associato alla chiave privata presente nel keystore a cui ci si riferisce con l'alias "desktopApp"; la password associata alla chiave privata è "desktopApp". Il comando avrebbe potuto essere anche significativamente più corto (ssarebbe stato ufficiente `keytool -genkey`) se fossero state accettate le impostazioni di default (validità del certificato per 90 giorni, alias "mykey", keystore ".keystore"). Alcune informazioni sarebbero state comunque richieste all'utente, come la password del keystore, la password per la chiave privata e le informazioni sull'entità generatrice delle chiavi (che sono dette *distinguished name information*).

Finora si è ottenuto un certificato auto-firmato: un certificato risulta essere accettato con più sicurezza se firmato da una **Certification Authority** (CA). Per ottenere una firma si deve quindi prima di tutto generare una *Richiesta di Firma del Certificato* (CSR) attraverso il comando

```
keytool -certreq -file CertSignReq.csr -alias desktopApp
```

per mezzo del quale per l'entry "desktopApp" del keystore viene creata una CSR e messa nel file CertSignReq.csr. In un secondo momento si dovrà inoltrare tale file ad una Certification Authority (ad esempio *VeriSign, Inc*) che provvederà all'autenticazione e ritornerà un certificato, firmato da essi stessi, che autentica la chiave pubblica creata. L'ultimo passo è quello di rimpiazzare il proprio certificato auto firmato con quello appena ottenuto dalla CA, cosicché gli archivi JAR firmati risultino essere "affidabili" agli occhi degli utilizzatori finali. Per fare ciò è sufficiente il comando

```
keytool -import -trustcacerts -file SignedCert.cer
```

Giunti a questo punto, si hanno tutti gli strumenti necessari per firmare con un certificato "sicuro" il proprio archivio *.jar*. Il comando seguente a partire dal file *desktopApp.jar* produce l'archivio firmato *desktopAppSigned.jar* con il certificato associato all'entry del keystore *desktopApp* (che è l'entry per cui si era richiesta la certificazione alla CA *VeriSign, Inc*)

```
jarsigner -signedjar desktopAppSigned.jar desktopApp.jar desktopApp
```

Si supponga ora che l'archivio JAR firmato sia stato distribuito e che il cliente che voglia utilizzare tale applicazione, desideri prima verificare l'autenticità della firma. Per fare ciò egli dovrà, innanzitutto, importare il certificato della chiave pubblica nel proprio keystore come una entry "affidabile". Chi distribuisce l'applicazione dovrà quindi procedere ad esportare il proprio certificato tramite il comando

```
keytool -export -alias desktopApp -file MichelePantano.cer
```

che esporta il certificato relativo all'entry desktopApp nel file MichelePantano.cer e lo fornirà all'utente finale che si preoccuperà di autenticare la firma con l'aiuto del tool *jarsigner* usando il comando:

```
jarsigner -verify desktopAppSigned.jar
```

5.3 Il Protocollo JNLP

Java Network Launching Protocol and API (JNLP) è un protocollo che specifica le modalità con cui un'applicazione può essere eseguita su una macchina che funge da client, utilizzando risorse ospitate da un Web server remoto: il software Java Web Start è considerato un client JNLP, perché può eseguire tali applicativi.
I file JNLP sono documenti XML che includono alcune informazioni come il nome ed il path dell'archivio *.jar*, la versione del software JRE necessario per l'esecuzione dell'applicazione, il nome della classe contenente il metodo `main` dell'applicazione, oltre ad altri parametri di sistema. Un browser configurato correttamente scarica i file JNLP e li passa ad un Java Runtime Environment (JRE), il quale a sua volta scarica l'applicativo sulla macchina dell'utente e lo manda in esecuzione.
La struttura di un file JNLP è molto semplice: trattandosi, come già detto, di un file XML esso presenta la classica struttura definita dai *tag*. Per far sì che possa essere interpretato correttamente dai client JNLP esso deve soddisfare le regole

dal protocollo; le specifiche definiscono infatti una lista di *Elementi* e di *Attributi* che stabiliscono la struttura del file. Si analizza ora parte del codice scritto nel file JNLP che lancia l'applicazione prodotta nel lavoro di tesi.

Listing 5.1: Struttura base di un documento JNLP

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp
    spec="1.0+"
    codebase="http://localhost:8080/DesktopApplication/"
    href="desktopApplication.jnlp">
    <information>
        <title>Desktop Application</title>
        <vendor>E-project s.r.l.</vendor>
        <description>Desktop Application</description>
        <offline-allowed/>
    </information>
    <security>
        <all-permissions/>
    </security>
    <resources >
        <j2se version="1.6+"/>
        <jar href="lib/desktopApp.jar"/>
  <jar href="lib/cookxml/cookswing-1.5.1.jar"/>
  <jar href="lib/cookxml/cookxml-3.0.2.jar"/>
  <jar href="lib/aspriseTIFF.jar"/>
  <jar href="lib/jdom-1.1.2.jar"/>
  <nativelib href="lib/windows/twainlibs.jar"/>
    </resources>
    <application-desc main-class="DesktopApplicationMain"/>
</jnlp>
```

L'elemento *radice* del file JNLP è **jnlp**. Esso è l'elemento padre per tutti gli altri elementi del file e serve a specificare che si tratta di un file JNLP; è un elemento obbligatorio: senza di esso il file JNLP non viene riconosciuto appartenere a questo protocollo. Esso presenta 3 attributi:

- **spec**: denota la versione minima delle specifiche JNLP con cui quel file è compatibile. Il suo valore può essere 1.0, 1.5, o 6.0 oppure, se compatibile con tutte le versioni superiori alla 1.0, si usa scrivere la dicitura 1.0+;

- **codebase**: indica il percorso di base per tutti i path relativi specificati negli attributi *href* del file JNLP;

- **href**: è l'attributo che riporta il percorso del file JNLP stesso. Questo attributo è obbligatorio ed è richiesto da Java Web Start per far sì che l'applicazione venga inclusa nell'*Application Manager*.

L'elemento **information** è figlio dell'elemento padre **jnlp** e riporta i seguenti sotto-elementi:

- **title**: è il nome dell'applicazione. È un elemento obbligatorio;

- **vendor**: indica il nome del produttore del programma;

- **description**: è una breve descrizione dell'applicazione. Il sotto-elemento descrizione è opzionale e può specificare diversi attributi: *one-line*, *short* e *tooltip*. Per la spiegazione di questi si rimanda alle specifiche di protocollo;

- **offline-allowed**: specifica se l'applicazione possa essere utilizzata offline. Nel caso non sia presente questo elemento, l'applicazione non verrà lanciata dall'*Application Manager*: esso infatti esegue le applicazioni solo se il sistema risulta connesso alla rete Internet. La presenza o meno di questo parametro influisce anche sul download degli aggiornamenti del programma: un'applicazione per cui è consentita l'esecuzione anche offline non è detto che stia eseguendo l'ultima versione disponibile del codice. Se il client infatti non è connesso alla rete Internet, Java Web Start non potrà controllare la presenza di eventuali aggiornamenti e l'applicazione verrà eseguita utilizzando i file salvati da JWS nella propria cache.

L'elemento **security** è figlio anch'esso dell'elemento padre *jnlp*. Ogni applicazione viene, di default, eseguita in un ambiente di esecuzione "ristretto", la *sandbox*, per cui questo elemento non è obbligatorio. L'elemento *security* può essere utilizzato per richiedere un'esecuzione con pieno accesso alle risorse del client.
Se l'attributo **all-permissions** è specificato, il programma avrà un accesso totale alle risorse macchina del client ed alla rete locale. In questo particolare caso, tutti gli archivi JAR devono essere firmati (per quanto riguarda la firma di un file *.jar* si veda 5.2) e all'utente finale sarà richiesto di accettare un certificato la prima volta che si manderà in esecuzione tale programma.

Figura 5.4: Accettazione di un certificato

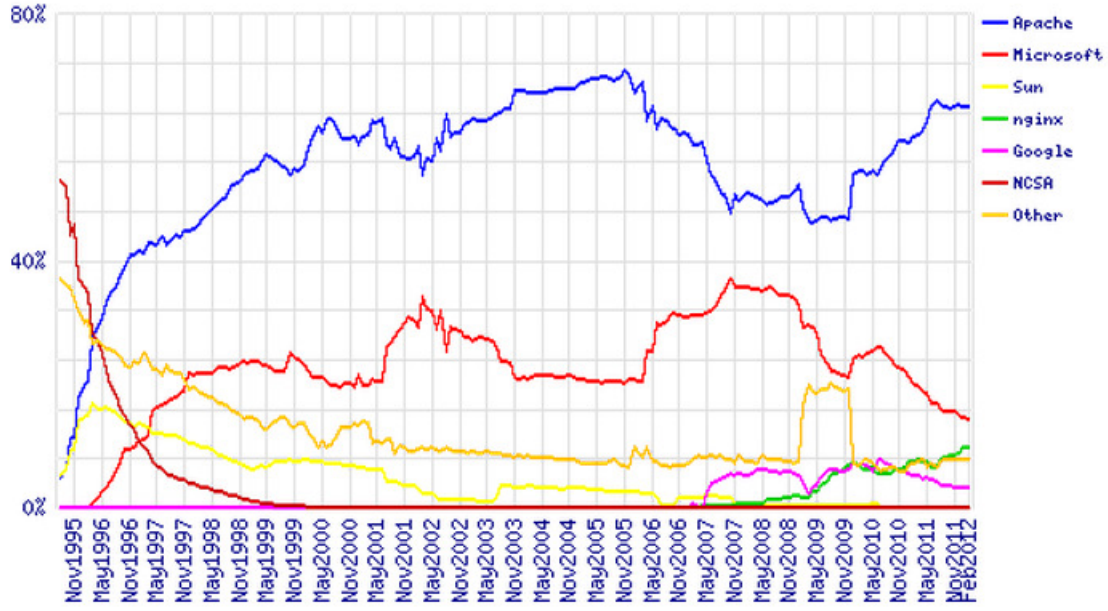
L'elemento **resources** è utilizzato per specificare tutte le risorse, come le classi Java, le librerie e le proprietà di sistema che sono parte dell'applicazione. Gli elementi *resources* hanno sei possibili sotto-elementi: **jar**, **nativelib**, **j2se**, **package**, **property** e **extension**. Gli elementi *package* e *extension* non vengono trattati, per il loro significato si rimanda alle specifiche del protocollo JNLP (scaricabili da `http://java.sun.com/javase/technologies/desktop/javawebstart/download-spec.html`).
L'elemento **jar** specifica un file JAR (il cui percorso è definito nell'argomento *href*) che fa parte dell'applicazione. Il file JAR tipicamente include delle classi Java utili per quella particolare applicazione, ma possono anche contenere altre risorse, come icone o file di configurazione, che sono ottenibili attraverso il meccanismo `getResource`.
nativelib è l'elemento che specifica un archivio JAR (il cui percorso è definito nell'argomento *href*) contenente le librerie native, cioè quelle librerie dipendenti dalla piattaforma (in Windows sono le *.dll, in Linux/Solaris sono le lib*.so) che devono essere caricate nel processo in esecuzione attraverso il metodo `System.loadLibrary`. In genere, le risorse *jar* e *nativelib* sono scaricate e disponibili localmente, per la JVM che esegue il programma, già prima dell'avvio del programma stesso. Si possono poi definire delle regole sui "tempi" di download: una risorsa con attributo **download="lazy"** non verrà scaricata fino a che l'applicazione non verrà eseguita, di contro l'attributo **download="eager"** specifica che il download deve essere eseguito prima dell'avvio dell'applicazione.
L'elemento **j2se** specifica la versione di Java 2 SE Runtime Environment suppor-

tata dall'applicazione. L'attributo *version* si riferisce, di default, alla verione 2 della piattaforma Java.
L'elemento **application-desc** indica che il file JNLP esegue un'applicazione (in opposizione all esecuzione di un'applet). Esso ha un attributo opzionale, **main-class**, che può essere utilizzato per indicare il nome della classe Java che contiene il metodo `main`; se il primo JAR specificato nel file JNLP contiene un file *MANIFEST.MF* in cui viene dichiarata la classe Java che contiene il metodo `main`, allora l'attributo può essere omesso. Se si vogliono/devono passare alcuni parametri al metodo `main`, essi possono essere specificati applicando all'elemento *application-desc* il sotto-elemento *argument* e specificando quindi il parametro desiderato.
Per l'analisi degli elementi e degli attributi prinicpali, si rimanda all'indirizzo `http://docs.oracle.com/javase/tutorial/deployment/deploymentInDepth/jnlpFileSyntax.html`.

5.4 Apache HTTP Server

Dopo aver visto come creare un archivio JAR contenente l'applicazione da distribuire mediante Java Web Start, come firmare tale archivio e come scrivere un file JNLP seguendo le specifiche del protocollo, l'ultimo passo è quello di creare una pagina HTML con un link che punti al file JNLP creato in precedenza e pubblicare poi tale pagina su un server, in modo che sia raggiungibile da un browser. Normalmente i server risiedono su sistemi hardware dedicati, ma per questa esperienza è stato utile eseguirlo in locale, così da sperimentare personalmente il funzionamento della tecnologia Java Web Start. Per fare ciò ci si è serviti di un *server HTTP Apache*. Il server Apache è un server HTTP *open source* compatibile con la grande maggioranza dei sistemi operativi esistenti tra cui UNIX/Linux, Microsoft Windows e Mac OS/X. L'obiettivo del progetto Apache è di implementare un server sicuro, personalizzabile ed efficiente che fornisca servizi HTTP rispettando gli standard HTTP. Il server HTTP Apache è, fin dal 1996, il web server HTTP più utilizzato[5]. Per farsi un'idea della sua diffusione, si osservi l'immagine riportata sotto.

[5]dati riportati da **Netcraft** e visualizzabili su `http://news.netcraft.com/archives/2012/02/07/february-2012-web-server-survey.html`



Developer	January 2012	Percent	February 2012	Percent	Change
Apache	378,267,399	64.91%	397,867,089	64.92%	0.01
Microsoft	84,288,985	14.46%	88,210,995	14.39%	-0.07
nginx	56,087,776	9.63%	60,627,200	9.89%	0.27
Google	18,936,381	3.25%	19,394,196	3.16%	-0.09

Figura 5.5: Diffusione dei web server da novembre 1995 a febbraio 2012

La procedura per installare sulla propria macchina un *server Apache* è davvero semplice: in ambienti Windows l'installer guida l'utente passo passo nell'installazione e non sono presenti quindi passaggi particolarmente difficili; viene data la possibilità di impostare alcune informazioni come il nome della propria macchina all'interno della rete, il nome del server e l'indirizzo email dell'amministratore del server. Nel mio caso specifico, poichè il sistema non deve fungere da vero e proprio Web server, alla voce Network Domain e Server Name ho inserito "localhost" e per quanto riguarda l'indirizo email, un indirizzo e-mail qualsiasi. È possibile inoltre cambiare la porta in cui il Web server starà in ascolto: quella di default è la porta 80, ma è possibile sceglierne un'altra nel caso la stessa venga già utilizzata da un altro servizio. Tutti questi dati sono comunque modificabili, accedendo al file *httpd.txt* conenuto nella cartella di configurazione del server.

L'ultimo passo è la creazione di una pagina HTML: la pagina è assolutamente minimale ed il suo aspetto poco curato; l'unica sua utilità è quella di fornire

un link al file JNLP che lancia l'applicazione da me creata. Come si evince dal codice riportato, essa contiene solamente un'intestazione di primo livello (definita dal tag *<h1>*) con del testo (che indica il titolo della pagina) ed un link al file JNLP che, come si può vedere dal path riportato nell'attributo *href* è contenuto nella cartella *app*. Dal path si può inoltre notare che la porta di ascolto del web server è stata modificata: al posto di quella "standard" si è scelto la porta 8081.

Listing 5.2: Codice per una pagina HTML

```
1 <html>
2     <body>
3         <h1>Pagina di prova – Desktop Application</h1>
4         <a href="http://localhost:8081/app/DesktopApplication.
            jnlp">DOWNLOAD</a>
5     </body>
6 </html>
```

Ora non resta altro da fare che avviare il Web server, digitare nel proprio browser l'indirizzo della pagina HTML e fare clic sul link contenuto nella pagina per scaricare il file JNLP. Con un doppio clic sul file scaricato si avvierà l'applicazione.

Capitolo 6

I form “dinamici” e l’XML

Uno dei aspetti che mi ha particolarmente interessao in questo progetto è stato quello riguardante la creazione di **form “dinamici”**. Un **form** (letteralmente “modulo”) è il termine utilizzato per indicare quella particolare interfaccia che consente l’inserimento di dati da parte dell’utente finale. Di norma, i form vengono utilizzati nei siti Internet per collezionare i dati inseriti dagli utenti (ad esempio per la creazione di un account utente, per conservare informazioni riguardanti l’acquisto di prodotti, ecc.) ed inviarli ad un database. L’aggettivo **dinamico** invece assume una sfumatura leggermente diversa rispetto a quello che è il suo significato classico; se un form dinamico, soprattutto in ambito Web, è una “griglia” d’inserimento dei dati che cambia il proprio aspetto a seconda che alcuni suoi campi siano compilati o meno o a seconda di particolari selezioni operate mediante bottoni radio, in questo specifico progetto di tesi il termine assume un’accezione diversa. Il form è inteso *“dinamico”* perché esso può essere modificato in qualsiasi istante senza dover procede alla ricompilazione del codice. Chiaramente, quello che andrà modificato non sarà il codice Java che esegue il programma, ma quello di alcuni file d’“appoggio” che vengono letti per la creazione dei form. Il vantaggio principale derivante da questo tipo di approccio è che, non essendoci la necessità di ricompilare il codice, non vi è nemmeno la necessità di procedere con una conseguente fase di redistribuzione del codice aggiornato.
Nelle prossime sezioni vedremo gli strumenti utilizzati per l’implementazione di questa caratteristica all’interno del programma che ho sviluppato, soffermandoci sugli aspetti più interessanti ed analizzando alcuni frammenti di codice particolarmente significativi.

6.1 Il linguaggio di markup XML

La creazione dei *form "dinamici"* si basa sulla lettura di file XML definiti da una particolare grammatica. A questo proposito, si procede introducendo brevemente il *metalinguaggio* XML.

XML è l'acronimo per e**X**tensible **M**arkup **L**anguage ed è uno standard supportato dall'organizzazione *W3C*[1] per il markup dei documenti, che definisce la sintassi generica utilizzata per contrassegnare (*mark up*) i dati mediante *tag* semplici e leggibili. La sintassi XML rappresenta un formato standard per i documenti utilizzati nei computer: esso risulta infatti sufficientemente flessibile da poter essere personalizzato per l'uso nei domini più diversi come le pagine Web, lo scambio di dati fra database, la grafica vettoriale e molti altri.

XML è un linguaggio di *meta-markup* per documenti di tipo testuale. I dati vengono quindi inclusi all'interno di documenti XML sotto forma di stringhe di testo, racchiusi dai *tag* di markup che li descrivono. Una particolare unità di dati viene detto *elemento*. Le specifiche XML descrivono con precisione l'esatta sintassi delle stringhe di testo: il modo in cui gli elementi sono delimitati dai tag, l'aspetto dei tag, i nomi accetttabili per gli elementi, il posizionamento degli attributi, ecc. Superficialmente il markup di un documento XML assomiglia a quello di un HTML (a cui si è accennato in 5.4), tuttavia esistono alcune differenze tra i due.

L'elemento di distinzione più importante consiste nel fatto che XML è un linguaggio di *meta-markup*: non presenta quindi un insieme prefissato di tag e di elementi che siano utilizzati universalmente per tutte le applicazioni. XML permette infatti agli sviluppatori di definire gli elementi di cui hanno bisogno, nella forma in cui risultano più leggibili ed efficaci. Nonostante la flessibilità a livello di definizione degli elementi, XML stabilisce alcune regole molto rigide per quanto riguarda la grammatica dei documenti: il posizionamento dei tag, i nomi "legali" che è possibile assegnare loro, il modo in cui gli attributi vengono associati agli elementi, ecc. Questa grammatica è ragionevolmente rigida da permettere lo sviluppo di *parser* XML per la lettura e la comprensione di qualsiasi documento XML: i documenti che soddisfano alle regole della grammatica XML sono gene-

[1] Il **World Wide Web Consortium** (W3C) è una comunità internazionale che sviluppa *protocolli* e *standard open* per assicurare una crescita a lungo termine del Web e per sfruttarne a pieno le sue caratteristiche.

ralmente detti *well formed*.

All'interno del progetto da me realizzato, ho fatto uso di documenti XML sia per la creazione di form, sia per la creazione di file di log e di configurazione: vedremo quindi gli strumenti utilizzati per la creazione ed il parsing dei documenti XML e la sintassi in essi utilizzata.

6.2 Le librerie CookXml e CookSwing

Il progetto a cui ho preso parte prevede di destinare alle banche il software prodotto, per facilitarle nella scansione di documenti che attestino operazioni bancarie. Se, poniamo il caso, l'operatore di sportello volesse trasferire in formato digitale un bonifico, è richiesto che inserisca alcuni dati relativi al bonifico in un form, prima di effettuare la scansione del documento.
Si immagini adesso che, per qualche motivo, venga effettuata una modifica alla normativa che regola le informazioni da riportare sui bonifici, introducendo in questo un nuovo campo obbligatorio. Il codice da me prodotto non subirebbe alcuna modifica; basterebbe infatti andare ad agire sul documento XML che definisce il form relativo al Bonifico Bancario; tale documento viene letto ad ogni esecuzione del programma ed è "interpetato" dalle librerie CookSwing.

CookXml e **CookSwing** sono due librerie Java per la creazione di interfacce grafiche a partire dalla lettura di un documento XML.
CooxXml è, tra le due, la libreria più "potente" perché "decodifica" gli oggetti definiti in un XML utilizzando un approccio interpretativo. Essa infatti carica una libreria di tag definiti dall'utente per convertire un file XML in oggetti Java. Il risultato ottenuto è un parser XML altamente configurabile (e, purtroppo, anche altamente complesso). La libreria CookSwing invece risulta essere di più semplice utilizzo e meno dispendiosa (come numero di righe di codice prodotto). Essa infatti dispone già di una libreria di tag che contiene i principali oggetti Java del package *Swing*, evitando così la noiosa procedura di definizione della TagLibrary.

6.2.1 CookXml

CookXml utilizza una serie di oggetti Java (Creator, Setter, Adder, Converter) per interpretare gli XML; tali oggetti formano una libreria di tag.

Per utilizzare CookXml basta creare una *TagLibrary*, che controlli come gli oggetti vengono creati, modificati e collegati fra loro. In particolare si fa uso di:

- **factory object** per la creazione degli oggetti Java;
- funzioni **setter** per l'assegnazione degli attributi agli oggetti creati;
- funzioni **adder** se si vogliono aggiungere degli oggetti al proprio oggetto padre (ad esempio, per aggiungere un panel ad un frame).

Nel mio progetto ho dapprima cercato di realizzare i form di inserimento dei dati mediante i metodi messi a disposizione dalla libreria *CookXml*[2], salvo poi ripiegare nell'uso della più "comoda" *CookSwing*. Ad ogni modo, con un breve esempio si possono apprezzare quelle che son le potenzialità di CookXml.

I passi per il parsing di un documento XML sono i seguenti:

Passo 1: Creazione di una TagLibrary
Il primo passo consiste nella creazione di una libreria di tag che conterrà tutte le definizioni. Poiché si farà uso di un unico *namespace*, ci si servirà dell'oggetto SingleNSTagLibrary.

Passo 2: Tag Creator
I *creator* sono degli oggetti costruttori (*factory object*) che "spiegano" a *CookXml* come creare un oggetto quando si incontra un particolare tag nel documento XML. Per esempio, si può mappare il tag <menubar> con l'oggetto *JMenuBar*, <menu> con *JMenu* e <menuitem> con *JMenuItem*. Queste classi (*JMenuBar*, *JMenu*, *JMenuItem*) sono tutte dotate di un costruttore di default: verrà quindi utilizzata la classe *DefaultConstructor* per generare i *creator*.

Passo 3: Setter
I **setter** sono utilizzati per modificare gli oggetti mediante attributi. Ad esempio, il tag <menu> ha attributi di testo. Se si utilizza il `DefaultSetter`, allora CookXml cerca delle variabili di testo e dei metodi Java che attribuiscano delle stringhe a degli oggetti. Poiché `JMenu` ha delle funzioni "setText" (il costruttore della classe accetta infatti come parametro una stringa `JMenu(String s)` per

[2]Le API della libreria sono disponibili all'indirizzo `http://cookxml.yuanheng.org/apidoc/index.html`

definire il testo che comparirà nel menù), si farà uso della classe *DefaultSetter*. In caso Java non preveda questa funzione, si dovrà aggiungere qualche riga di codice, pertanto è sempre consigliato l'uso di attribuiti XML che possano essere poi gestiti da metodi o variabili Java.

Passo 4: Adder

Gli **Adder** sono usati per aggiungere gli oggetti Java creati da un tag *figlio* agli oggetti *padre*. CookXml mette a disposizione dell'utente la classe DefaultAdder che si occupa di cercare delle funzioni per aggiungere oggetti figli ai propri padri. In questo caso, per esempio, si sarà in grado di aggiungere l'oggetto corrispondente al tag <menu> al proprio padre <menubar> e l'oggetto corrispondente a <menuitem> a <menu>; Java infatti mette a disposizione i metodi `insert(JMenuItem mi, int pos)` ed `add(JMenu m)` per aggiungere gli oggetti Jmenu e JMenuItem ai rispettivi padri.

Nell'esempio, di cui è riportato sia il codice Java che il relativo documento XML, si realizza un semplice *JFrame* Java con una barra di menù.
Il documento XML è facilmente interpretabile e la sua struttura suggerisce le relazioni padre/figlio tra gli oggetti Java. L'indentatura che si può notare in questo file, ad esempio, indica che il <*menuitem*> con attributo *text="XML"* è figlio di <*menu*> con attributo *text="Open"*.

Listing 6.1: Documento XML per la definizione di una JMenuBar

```
1  <?xml version="1.0"?>
2  <menubar>
3      <menu text="File">
4    <menu text="Open">
5        <menuitem text="XML"/>
6        <menuitem text="Java"/>
7    </menu>
8        <menuitem text="Exit"/>
9      </menu>
10     <menu text="Edit">
11   <menuitem text="Cut"/>
12   <menuitem text="Copy"/>
13   <menuitem text="Paste"/>
14     </menu>
15     <menu text="Help">
16   <menuitem text="About"/>
17     </menu>
```

```
18 </menubar>
```

Con seguente codice Java, invece, ci si occupa di definire dapprima una TagLibrary e poi di effettuare il parsing del documento XML.

Listing 6.2: Codice Java per il parsing di un documento XML utilizzando le librerie *CookXml*

```
1  public class CookxmlMenuBar {
2      private static SingleNSTagLibrary singTagLibrary;
3
4      static {
5    //Passo 1
6    SingleNSTagLibrary tagLibrary = new SingleNSTagLibrary();
7    // Passo 2
8    tagLibrary.setCreator("menubar", DefaultCreator.getCreator(
       JMenuBar.class));
9    tagLibrary.setCreator("menu", DefaultCreator.getCreator(
       JMenu.class));
10   tagLibrary.setCreator("menuitem", DefaultCreator.getCreator(
       JMenuItem.class));
11   // Passo 3
12   tagLibrary.setSetter("menu", "text", DefaultSetter.
       getInstance());
13   tagLibrary.setSetter("menuitem", "text", DefaultSetter.
       getInstance());
14   // Passo 4
15   tagLibrary.setAdder("menubar", DefaultAdder.getInstance());
16   tagLibrary.setAdder("menu", DefaultAdder.getInstance());
17
18   singTagLibrary = tagLibrary;
19     }
20
21     public static void main(String[] args) {
22   JMenuBar menuBar;
23   try{
24       //Il DocumentBuilder è utilizzato per il parse del
         documento XML
25       DocumentBuilder builder = DocumentBuilderFactory.
         newInstance().newDocumentBuilder();
```

```
26        CookXml cookXml = new CookXml (builder, singTagLibrary,
            (Object)null);
27        menuBar = (JMenuBar)cookXml.xmlDecode("C:/xml/menubar1.
            xml");
28    }
29    catch (Exception e) {
30       logger.error("Exception in main", e);
31    }
32    JFrame f = new JFrame("CookXml MenuBar");
33    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
34    f.setJMenuBar(menuBar);
35    f.pack();
36    f.setLocationRelativeTo(null);
37    f.setVisible(true);
38      }
39 }
```

CookXml è uno strumento dalle grandi potenzialità, ma la definizione di una libreria dei tag gioca notevolmente a suo sfavore, essendo una lavoro abbastanza noioso e non sempre semplice da effettuare; è questo il caso in cui si necessita dell'implementazione dei *Converter*, per realizzare quei metodi che nativamente non vengono messi a disposizione dalle librerie Java standard.

6.2.2 CookSwing

CookSwing[3] è una libreria che fornisce classi e metodi per la costruzione di una Graphical User Interface (GUI) Java *Swing* a partire dalla lettura di un documento XML. Come anticipato, esso dispone già di una libreria di tag che gestisce i componenti *Swing*, ma non solo: si possono utilizzare anche i *LayoutManager* ed i *Listener* definiti in *Awt* e *Swing*, le classi *Border* e molti altri oggetti (icone, stringhe, Vettori, LinkedList, ecc).
Non dovendo definire una *TagLibrary*, il suo utilizzo è di gran lunga meno articolato; si riporta un frammento di codice per poter notare la differenza che lo distingue dal listato 6.2 (tralasciando la parte che implementa l'oggetto Java *JFrame*, che in questo secondo caso viene descritto nel file XML)

Listing 6.3: Esempio di utilizzo di CookSwing

```
1 CookSwing cookSwing = new CookSwing();
```

[3]Le API della libreria sono disponibili all'indirizzo `http://cookxml.yuanheng.org/apidoc/index.html`

```
2 cookSwing.render(System.getProperty("user.home")+"\\
    DesktopApplication\\Xml\\Bonifico.xml").setVisible(true);
```

Si riporta inoltre una parte del contenuto del file *Bonifico.xml* che definisce un form per l'inserimento di alcune informazioni riguardanti il Bonifico Bancario di cui si vuole effettuare la scansione.

Listing 6.4: Form di inserimento dati per un Bonifico Bancario

```
1  <frame id="Bonifico" title="Bonifico" size="1024, 768"
     defaultcloseoperation="EXIT_ON_CLOSE">
2      <panel>
3    <borderlayout>
4        <constraint location="North">
5      <panel>
6          <gridbaglayout>
7        <gridbagconstraints id="left" anchor="EAST" insets="
           3,3,3,3">
8        </gridbagconstraints>
9        <gridbagconstraints id="grid2" anchor="WEST" insets="
           3,3,3,3">
10       </gridbagconstraints>
11       <gridbagconstraints id="right" weightx="1.0" anchor="
           WEST" gridwidth="REMAINDER" insets="3,3,3,5">
12       </gridbagconstraints>
13           <idref ctor="left">
14         <label text="Data Valuta (GG-MM-AAAA)" labelfor="
             id:Data Valuta" font=" , ,18"/>
15           </idref>
16           <idref ctor="right">
17         <formattedtextfield columns="20" id="DataValuta" font=
             " , ,16">
18             <dateformatter>
19           <simpledateformat pattern="dd-MM-yyyy"/>
20             </dateformatter>
21             <date setas="value" value="current"/>
22         </formattedtextfield>
23           </idref>
24           <idref ctor="left">
25         <label text="ABI" labelfor="Numero ABI" font=" , ,18"/
             >
26           </idref>
```

```
27            <idref ctor="grid2">
28          <formattedtextfield columns="5" id="ABI" font=" , ,16"
              >
29              <maskformatter mask="*****" validcharacters="
                  0123456789"/>
30              <string setas="value"/>
31                </formattedtextfield>
32            </idref>
33            <idref ctor="grid2">
34          <label text="Provincia" labelfor="id:Provincia" font="
               , ,18"/>
35            </idref>
36            <idref ctor="right">
37          <formattedtextfield columns="2" id="Provincia" font="
              , ,16">
38              <maskformatter mask="**" validcharacters="
                  ABCDEFGHILMNOPQRSTUVZWXYJK"/>
39              <string setas="value"/>
40          </formattedtextfield>
41            </idref>
42        </gridbaglayout>
43          </panel>
44      </constraint>
45            <constraint location="South">
46          <panel>
47        <button id="okButton" text="Ok" actionlistener="saveText
            " />
48        <button id="backButton" text="Torna indietro" mnemonic="
            VK_T" actionlistener="formBackButtonAction" />
49        <button id="cancelButton" text="Annulla" actionlistener=
            "exitAction" />
50          </panel>
51      </constraint>
52    </borderlayout>
53      </panel>
54 </frame
```

Analizzando il file XML, saltano all'occhio subito alcune particolarità che è bene illustrare. Per prima cosa, risulta evidente che il documento XML definisce un `JFrame` Java, il cui titolo e le cui dimensioni sono specificate come attributi nel tag <*frame*>. Nel `JFrame` è applicato un contenitore di tipo JPanel (indicato dal

tag <*panel*>) su cui è applicato un layout di tipo `BorderLayout`[4]. Questo particolare layout suddivide l'area del panel in cinque zone: *north*, *south*, *west*, *east* e *center*. Nella sezione *south* c'è un *JPanel* contenente tre oggetti di tipo `JButton`, in quella *north* è a sua volta annidato un altro `JPanel`, questo però con layout di tipo *GridBagLayout*[5]. La classe `GridBagLayout` definisce un layout molto flessibile, che allinea i componenti verticalmente ed orizzontalmente, indipendentemente dalla loro dimensione. Ciascun oggetto gestito da un `GridBagLayout` è associato ad un'istanza di *GridBagConstraints*[6]. Tali oggetti specificano dove un componente debba essere visualizzato sulla griglia e come debba essere posizionato all'interno dell'area di visualizzazione. Inoltre, in aggiunta a questi vincoli, `GridBagConstraints` tiene in considerazione le dimensioni minime e quelle preferite da ciascun componente. Questi dettagli implementativi sono specificati negli attributi *weightx*, *gridwidth*, *insets* ed *anchor*.

(a) BorderLayoout (b) GridBagLayout

Figura 6.1: Java LayoutManager

È curioso vedere come vengono gestiti i campi di testo: essi sono degli oggetti di tipo `JFrormattedTextField` ed in quanto tali, l'inserimento di informazioni al loro interno può essere definito da alcune regole. Per spiegare meglio questa affermazione, si guardi il campo di testo che conterrà la data del bonifico bancario (indicato con l'id "DataValuta"): esso accetterà esclusivamente date nella forma *dd-MM-yyyy* e qualsiasi altro formato (ad esempio utilizzando punti o barre al posto dei trattini, o invertendo il mese con il giorno) sarà rifiutato. I `JFormattedTextField` possono così accettare solo numeri (come per il campo *ABI*, che richiede cinque caratteri numerici), solo lettere maiuscole (è il caso del

[4] `http://docs.oracle.com/javase/6/docs/api/java/awt/BorderLayout.html`
[5] `http://docs.oracle.com/javase/6/docs/api/java/awt/GridBagLayout.html`
[6] `http://docs.oracle.com/javase/6/docs/api/java/awt/GridBagConstraints.html`

campo *Provincia*) e così via, purché sia definita per ciascuno di essi una regola. Infine, focalizzando l'attenzione sugli attributi dei tag <*button*> per la costruzione di oggetti Java `JButton`, si può notare che per ciascun `JButton` è data la possibilità di definire una scorciatoia da tastiera (definita con l'attributo *mnemonic*) ed un oggetto `ActionListener` per l'esecuzione di una particolare azione ogniqualvolta si registri una "pressione" del bottone.

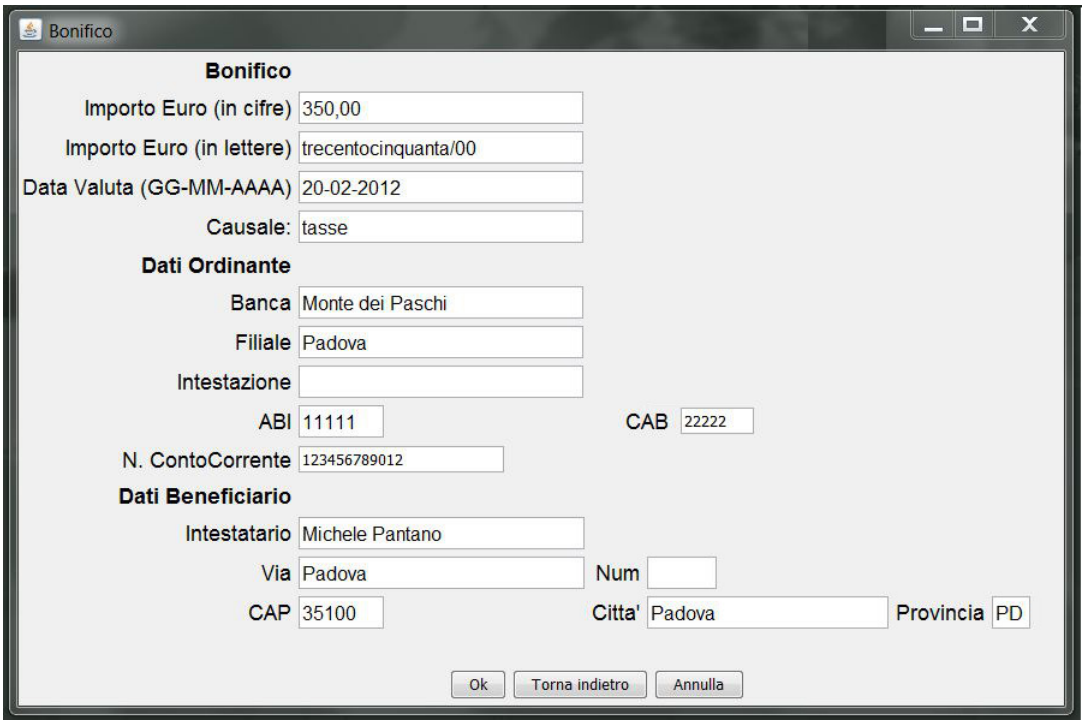


Figura 6.2: Form di inserimento dati per un Bonifico Bancario

Per ulteriori dettagli riguardanti i Tag della libreria CookSwing ed i loro attributi, si rimanda all'indirizzo Web `http://cookxml.yuanheng.org/cookswing/tagdoc/index.html` in cui si trova una buona documentazione della libreria.

6.3 La libreria JDOM

JDOM è il nome di un *Document Object Model*[7] per i documenti XML, open source, scritto in Java e sviluppato specificatamente per le piattaforme Java. Il

[7]**Document Object Model** (DOM): è lo standard ufficiale del *W3C* per la rappresentazione di documenti strutturati in maniera da risultare indipendenti sia dal linguaggio che dalla piattaforma. Per ulteriori informazioni si rimanda a 6.3.1 o alle specifiche al sito`http://www.w3.org/DOM/`.

suo obiettivo è quello di fornire una soluzione completa per la lettura, la manipolazione e la scrittura di file XML. JDOM deve il suo successo all'integrazione tra *DOM* e *SAX*[8]; i vantaggi di un parser SAX rispetto ad uno DOM sono

- la possibilità di poter trattare un sottoinsieme delle informazioni,
- la possibilità di interrompere il parsing del documento quando necessario;
- la possibilità di poter costruire una propria struttura dati a partire dalle informazioni ricavate da un documento XML (a differenza di DOM che genera una struttura di dati ad albero di tutto il documento);
- la possibilità di poter analizzare documenti di qualsiasi dimensione (in quanto non richiede grandi risorse di memoria);
- la semplicità;

mentre gli svantaggi sono

- l'impossibilità di accedere casualmente al documento (è consentito soltanto un accesso sequenziale);
- l'impossibilità di modificare il documento XML (è un sistema di sola lettura);
- la difficoltà di implementazione di interrogazioni complesse al documento.

JDOM è un progetto tuttora attivo ed in evoluzione, la cui ultima versione (1.1.2) è stata rilasciata il 23 ottobre 2011.
Si analizzano in questa sezione quelle che sono le caratteristiche principali di JDOM, mostrando infine l'utilizzo di questa libreria all'interno del lavoro di tesi, sia per quanto riguarda il processo di creazione e scrittura di un file XML, sia per la lettura da un documento già esistente.

Un documento XML è rappresentato come un'istanza della classe *org.jdom.Document*; la prima caratteristica di JDOM che lo sviluppatore incontra è il poter creare gli oggetti direttamente col proprio costruttore (particolarità che richiama la programmazione in stile Java), senza la necessità di utilizzare metodi factory (come accade, per esempio, in *CookXml*). Per creare un oggetto

[8]**Simple API for XML** (SAX): è un parser ad accesso sequenziale per documenti XML; fornisce un meccanismo per la lettura di dati contenuti in un file XML ed è usato, solitamente, come alternativa ai DOM.

`org.jdom.Element` che rappresenta un tag *<Document>* è quindi sufficiente scrivere `Element rootElement = new Element("Document");`. Se poi si desidera che l'oggetto di tipo `Element` creato assuma il ruolo di elemento *root*, basterà utilizzare il metodo `setRootElement (rootElement);`. Con estrema semplicità è possibile inoltre creare dei *tag figli* da associare ai propri padri: sia *rootElement* il tag padre dell'elemento *pages*, allora si scriverà `rootElement.addContent(pages)`.

Per il parsing di documenti XML, JDOM fornisce i *builder*, oggetti che costruiscono un documento a partire da diverse sorgenti dati (file, `InputStream` ed `InputSource`). Sono definiti due tipi di builder, `DOMBuilder` e `SAXBuilder`. Come è evidente dal nome `DOMBuilder` carica un documento a partire da un oggetto `org.w3c.dom.Document` mentre *SAXBuilder* sfrutta un parser *SAX* ed è quindi più leggero e performante. Nel progetto mi sono servito esclusivamente di *builder* di tipo `SAXBuilder`.
Per la scrittura dei documenti XML invece è fornita una classe apposita, *org.jdom.output.XMLOutputter*, che si occupa appunto di questo ed ha un pratico metodo `output(Document d, Writer w)` che accetta come argomenti in input un *org.jdom.Document* ed un `java.io.Writer`[9]. Per la creazione di un file XML si osservi il listato 6.5, in cui si è fatto uso di un oggetto *FileWriter* (sottoclasse di *Writer*) per la creazione del file XML, a cui poi è stata assegnata la classica visualizzazione a "tag annidati" con il comando `xmlOutput.setFormat (Format.getPrettyFormat())` (la formattazione di default scrive tutti i tag sulla medesima riga). L'ultima caratteristica che mi preme evidenziare è l'accesso agli elementi di un documento. Il modello a oggetti di JDOM permette di "esplorare" la struttura di un file XML in maniera molto semplice; ad esempio dato un `Document` è possibile accedere all'elemento radice con: `Element root = doc.getRootElement()`. A questo punto si può accedere alla lista di tutti i nodi figli `List childrenList = root.getChildren()` o alla lista dei nodi figli aventi lo stesso tag `List childrenList = root.getChildren("pagine")`. Per accedere al contenuto di un elemento, si usa invece il metodo più generico `getContent()` o in alternativa uno più specifico, `getText()`, se si è certi che si tratti una stringa. In maniera del tutto analoga, per ciascun elemento si possono ottenere anche i valori degli attributi `e.getAttributeValue(name)`; tuttavia non ho fatto uso di questa particolare funzione nel mio codice.
I comandi analizzati sono davvero banali, ma allo stesso tempo dimostrano quanto JDOM sia funzionale e facile da comprendere per uno sviluppatore Java.

[9] `http://docs.oracle.com/javase/6/docs/api/java/io/Writer.html`

Listing 6.5: Creazione di un file XML

```
1  public static void createSourceXML(String selectedSource){
2      try{
3          File xmlFile = new File(pathConfigXMLFile);
4    if (!xmlFile.exists())
5        xmlFile.createNewFile();
6
7          org.jdom.Element rootElement = new org.jdom.Element("
             devices");
8          org.jdom.Document document = new org.jdom.Document(
             rootElement);
9          document.setRootElement(rootElement);
10
11         org.jdom.Element rootNode = document.getRootElement();
12
13         org.jdom.Element date = new org.jdom.Element("source")
             .setText(selectedSource);
14         rootNode.addContent(date);
15
16         XMLOutputter xmlOutput = new XMLOutputter();
17         xmlOutput.setFormat(Format.getPrettyFormat());
18         FileWriter writer = new FileWriter(pathConfigXMLFile);
19         xmlOutput.output(document, writer);
20         writer.flush();
21         writer.close();
22     }
23     catch (IOException e) {
24   logger.error("createConfigXML(String)"+ " IOException IO", e
       );
25     }
26 }
```

Listing 6.6: Parsing di un file XML

```
1  public String getNameXML(String filePath){
2      SAXBuilder builder = new SAXBuilder();
3      String name;
4      try{
5    org.jdom.Document document = builder.build(new File(filePath
       ));
```

```
6     Element root = document.getRootElement();
7     Element nameElement = root.getChild("name");
8     name = nameElement.getText();
9       }
10      catch (Exception e) {
11    logger.error("getUsername()", e);
12      }
13      return name;
14  }
```

Listing 6.7: File XML prodotto con l'utilizzo della libreria JDOM

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <scansione>
3      <date>16/01/2012</date>
4      <time>23:12:59</time>
5      <pages>4</pages>
6      <path>C:/DesktopApplication/Storico/</percorso>
7      <file>16012012_231259.tiff</file>
8  </scansione>
9  }
```

6.3.1 Gestione dei dati inseriti tramite form

Le specifiche di progetto richiedono che il documento digitalizzato sia inviato al back office, accompagnato da alcuni dati. Tali dati sono informazioni riguardanti il documento scansionato e servono per favorirne la gestione, l'archiviazione e la ricerca all'interno del sistema documentale. I dati allegati vengono inseriti dall'operatore durante la compilazione dell'apposito form; vediamo quindi come sia possibile la realizzazione di questo compito con l'aiuto del componente CookSwing, che obbliga l'utente a servirsi (seppur in maniera limitata) delle specifiche dello standard DOM.

Il **Document Object Model** è uno standard che definisce una serie di funzionalità per accedere ed aggiornare il contenuto, la struttura e lo stile di un documento (XML, HTML o XHTML[10]) rappresentandolo tramite una struttura ad albero. In base a questo approccio, la struttura del documento preso in

[10]XHTML: e**X**tensible **H**yper**T**ext **M**arkup **L**anguage. Per approfondimenti si veda `http://www.w3.org/TR/xhtml1/`

considerazione viene creata nella memoria della macchina come una gerarchia di oggetti, ciascuno dei quali rappresenta un diverso elemento del documento. DOM mette a disposizione delle API per rappresentare tali elementi ed interagire con essi e definisce inoltre la struttura logica dei documenti. Utilizando DOM si può quindi costruire un documento, navigare attraverso la sua struttura, aggiungere, modificare o cancellare elementi.
L'implementazione ad albero, tipica di DOM ha però una controindicazione: richiede che l'intero contenuto di un documento venga analizzato e salvato in memoria. DOM è utilizzato principalmente per recuperare informazioni da documenti con una strutturazione non standard, cioè dove gli elementi devono essere trovati in modo casuale. Per le applicazioni basate su XML che usano un processo di lettura e scrittura sequenziale, DOM presenta un grande spreco di memoria; per questo tipo di applicazioni si consiglia di usare il modello SAX.

Le specifiche dello standard DOM elaborate dal W3C sono suddivise in tre livelli, ma non andremo a studiare nei dettagli quali siano le loro caratteristiche. Ci basta ricordare che i livelli ricalcano quelle che sono le "evoluzioni" compiute dal DOM: con il passare degli anni, con l'evolversi delle potenzialità dei linguaggi e degli strumenti utilizzati nel Web, si è dovuto adattare ed aggiornare anche DOM. Il livello 1, completato nel 1998, fornisce il supporto per *XML 1.0* e *HTML 4.0* e definisce gli elementi DOM di base con interfacce contenenti i metodi e gli attributi di uso più comune. Il livello 2 (2000, W3C Reccomendation da gennaio 2003) ha esteso il primo livello, introducendo diverse funzionalità, tra cui le *viste filtrate* ed i *namespace*. Infine il terzo ed ultimo livello (W3C Reccomendation da aprile 2004), che ha introdotto nuovi metodi ed interfacce per una navigazione più rapida nel documento e per la validazione di esso. Nell'immagine è riportato lo schema con i moduli delle funzioni definite dalle specifiche, così come appare dopo il raggiungimento del terzo ed ultimo livello.

Nel frammento di codice che segue (listato 6.8), viene mostrata quale sia la procedura da seguire per ricavare le informazioni immesse con la compilazione dei campi di testo del form.
Le classi Java coinvolte nell'elaborazione mediante DOM sono contenute nei seguenti package:

- **javax.xml.parsers** contiene la classe DocumentBuilder che permette di creare un parser DOM ed utilizzare quest'ultimo per ottenere una struttura ad albero rappresentativa del contenuto di un documento XML;

- **org.w3c.dom** definisce tutte le interfacce coinvolte nella rappresentazione di un documento XML.

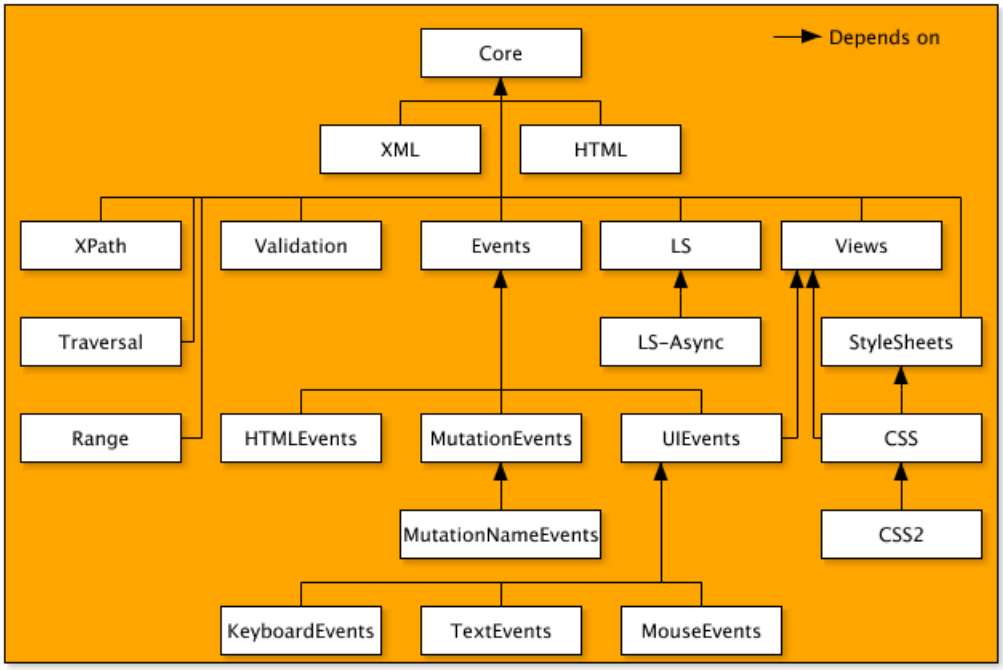


Figura 6.3: Document Object Module - Vista di tutti i moduli

La parte riguardante la creazione del file XML di output (quella che riporta tutti i dati inseriti nei campi di testo) è tralasciata perché analoga a quanto già visto in precedenza (listato 6.5).
Il procedimento di estrazione delle informazioni inserite nel form non è particolarmente complicato. I passi da seguire sono i seguenti:

- creare un oggetto *builder* di tipo *DocumentBuilder*. Il metodo `CookSwing.getSwingDocumentBuilder();` restituisce il builder di default per la creazione di un oggetto di tipo `CookSwing`;

- eseguire il *parse* del documento XML (nella fattispecie, del file che definisce la struttura del form relativo al Bonifico Bancario (listato 6.4);

- inizializzare una `org.w3c.dom.NodeList` che punti a tutti gli elementi del documento il cui tag è *<formattedtextfield>*;

- creare un elemento (`org.w3c.dom.Element`) per ciascun *item* della `NodeList`;

- per ciascun elemento, creare una stringa che sarà inizializzata con il valore dell'attributo "id";
- ottenere, con `(JTextField)cookSwing.getId(string).object`), un oggetto di tipo `JTextField`. L'istruzione riportata esegue il cast a `JTextField` del riferimento all'oggetto CookSwing che ha per "id" la stringa specificata (quella creata al punto precedente);
- ottenere, infine, il testo inserito dall'utente tramite il form, richiamando il metodo `getText()` sull'oggetto JTextField appena creato.

Listing 6.8: Cattura dei dati inseriti nel form per un Bonifico Bancario

```
1  DocumentBuilder builder = CookSwing.getSwingDocumentBuilder();
2  File f = new File("C:\\DesktopApplication\\Xml\\Bonifico.xml")
     ;
3  try{
4      Document document = builder.parse(f);
5
6      NodeList textFieldList = document.getElementsByTagName("
         formattedtextfield");
7
8      for(int i = 0; i<textFieldList.getLength(); i++) {
9          Element textFieldElement = (Element) textFieldList.
             item(i);
10         String textFieldString = textFieldElement.getAttribute
             ("id");
11         JTextField jtextfield = (JTextField) cookSwing.getId(
             textFieldString).object;
12         String inputText = jtextfield.getText();
13     }
14 }
```

Effettuare il parsing di documenti tramite DOM, come si è potuto constatare da questo esempio, è una procedura sostanzialmente facile, che non richiede al programmatore grandi sforzi. È richiesto qualche sforzo in più alla macchina e per questo, ove possibile, è preferibile usare i parser SAX.

Capitolo 7

Manuale Utente

Il progetto assegnatomi nel tirocinio svolto presso *E-project s.r.l.* prevede la realizzazione di un progetto pilota riguardante il tema della dematerializzazione e della gestionce documentale che verrà distribuito in ambito bancario. Lo scopo principale di questo applicativo è quello di fornire uno strumento per la scansione dei documenti bancari e l'invio di questi ad un sistema documentale che possa poi gestirli. Nonostante l'utilizzo dei computer sia divenuto assolutamente indispensabile anche per gli Istituti di Credito, non tutti gli operatori hanno la stessa dimestichezza quando si trovano di fronte ad una macchina. Si deve inoltre pensare che l'impiegato che voglia eseguire una scansione di un documento, non si soffermerà ad analizzare e valutare gli aspetti estetici del programma, ma gradirà che questo sia veloce e pratico da utilizzare, per poter completare quanto prima il processo di acquisizione. Per questa serie di motivi, in questi mesi di lavoro non ci si è concentrati tanto sugli aspetti esteriori del software (che dovranno essere chiaramente migliorati prima di un eventuale rilascio ufficiale), quanto piuttosto sulla realizzazione di un prodotto che fosse leggero, intuitivo e semplice da utilizzare. Segue ora una descrizione del funzionamento del programma, analizzando passo dopo passo le possibili scelte.

La prima schermata dell'applicazione è riportata nell'immagine 7.1 e pone l'utente di fronte a tre possibili scelte:

- procedere con la scansione del programma;
- consultare l'archivio delle scansioni già effettuate;
- uscire dal programma.

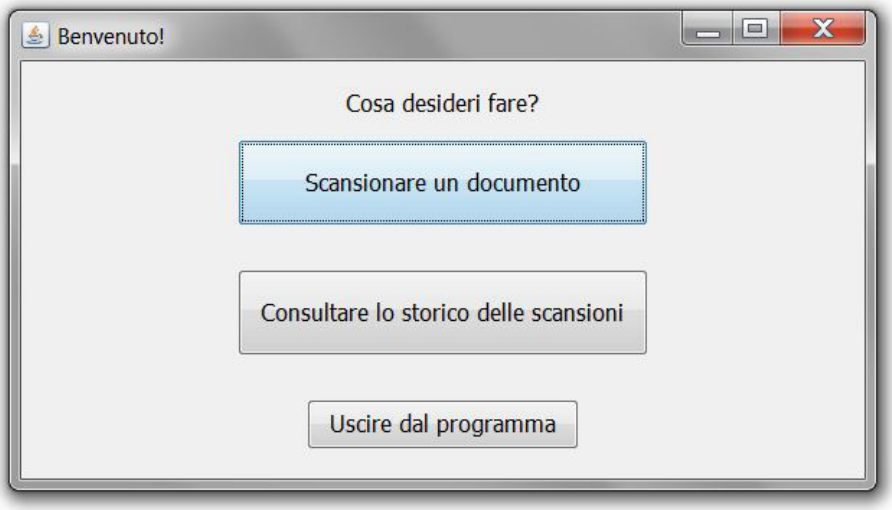


Figura 7.1: Schermata di benvenuto

Nel caso si voglia lasciare l'applicazione, viene richiesto di confermare la scelta attraverso una finestra (una *dialog box* appartenente alla classe Java *JOptionPane*) che avvisa che si sta per abbandonare il programma (figura 7.2).

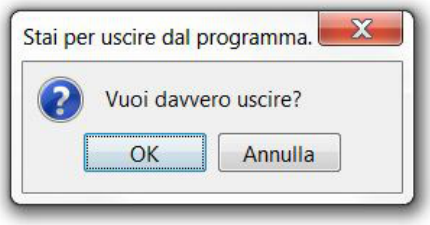


Figura 7.2: Dialog box per la conferma dell'uscita dal programma

Se si sceglie di consultare lo storico dei documenti scansionati, si apre una nuova finestra che visualizza un'elenco con i dettagli sulle scansioni effettuate. L'elenco è visualizzato mediante una tabella (un oggetto della classe Java JTable) suddivisa su più pagine, che riporta dieci voci per ciascuna pagina. La tabella presenta una colonna per ognuna delle informazioni riguardanti le scansioni effettuate ed è creata a partire dalla lettura dei log di scansione. Ogniqualvolta si utilizza uno scanner, o un altro strumento dedito all'acquisizione delle immagini, per importare documenti in formato digitale, viene prodotto un file XML di log che conserva alcune informazioni riguardanti quel particolare processo di importazione: il nome del file, la data e l'ora di scansione, il numero di pagine del documento scansionato, il nome dell'operatore che ha effettuato tale scansione. Le voci sono inserite nella tabella in ordine alfabetico rispetto al nome del file,

ma possono poi essere ulteriormente ordinate per colonna (facendo un clic con il mouse sull'header della colonna da ordinare). Nella figura sotto riportata viene mostrato l'ordinamento secondo la colonna "data".

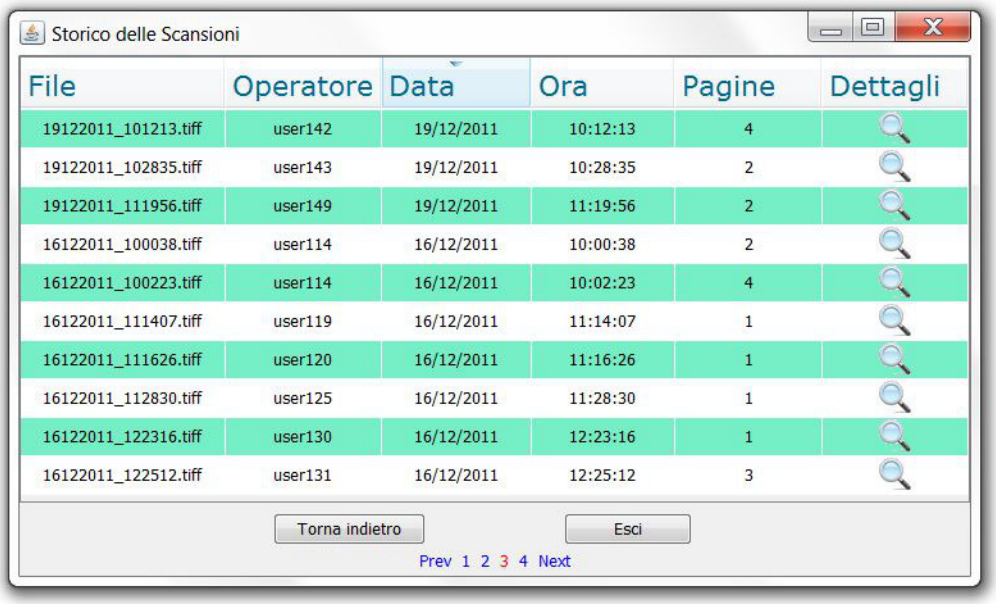


Figura 7.3: Tabella per la visualizzazione dello storico delle scansioni

Come si può notare dall'immagine 7.3, l'ultima colonna non contiene delle voci di testo, bensì tante piccole figure di lenti di ingrandimento: con un clic del mouse sopra tali icone, si aprirà un'altra finestra (figura 7.4) che riporta tutti i dettagl di scansione, in un formato più "leggibile" rispetto a quello fornito dalla tabella. In futuro non è escluso che questa finestra possa incorporare anche un'anteprima del file scansionato. Se dalla vista della tabella dovesse venir premuto il bottone "Torna indietro", si ritorna alla schermata iniziale.

Se si sceglie la voce "Scansionare un documento" si aprirà la finestra che permette di selezionare il tipo di documento che si desidera importare. Al momento compaiono tre singole voci (Bonifico, Fattura, Curriculum) a scopo puramente dimostrativo; difficilmente infatti l'operatore di sportello di una banca dovrà effettuare la scansione di un curriculum. Questo `JFrame` viene realizzato con il supporto delle API di `CookSwing` (6.2.2) e quindi nel caso vi fosse la necessità di inserire nuove tipologie di documento, tale modifica potrebbe essere apportata esclusivamente agendo sul file XML che realizza questo `JFrame`, senza dover aggiungere una sola riga di codice Java.

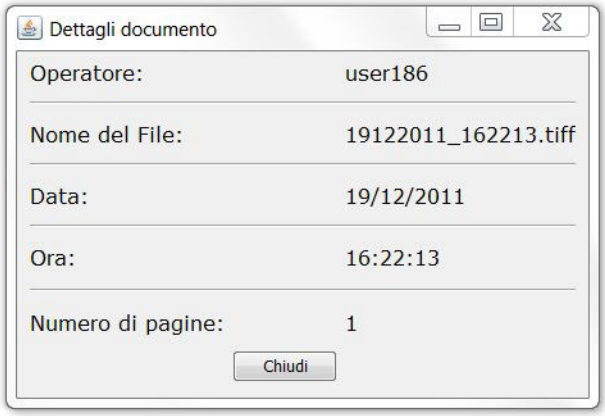


Figura 7.4: Dettagli riguardanti uno dei file scansionati

La schermata di selezione del tipo di documento è riportata in figura 7.5. Una

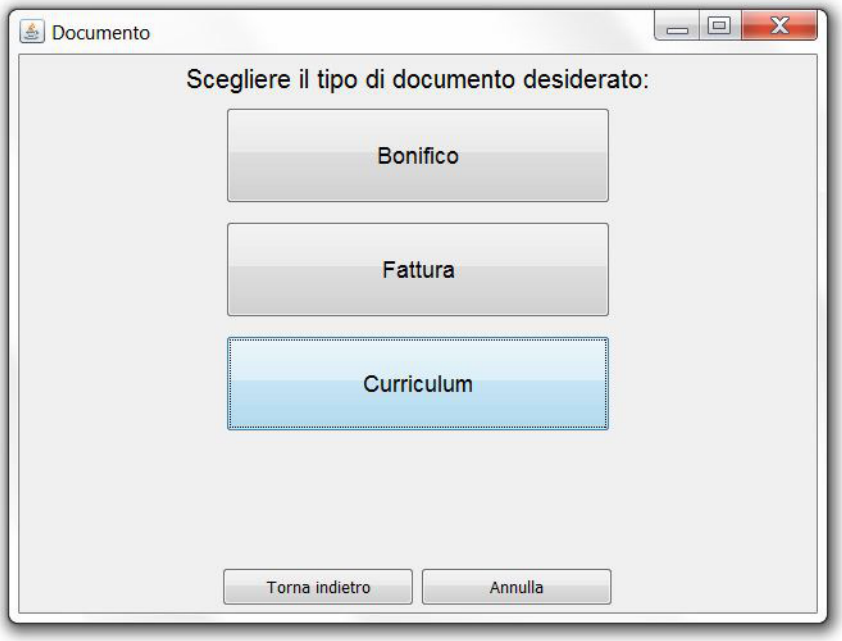


Figura 7.5: Selezione del tipo di documento da importare

volta selezionato il tipo di documento di cui si vuole effettuare la scansione, viene mostrata la finestra che presenta il form di inserimento dei dati; nell'immagine 7.6 viene illustrato quello relativo ad un documento di tipo "Fattura". Per ulteriori informazioni riguardo la creazione e la gestione del form, si rimanda ai paragrafi 6.2.2 e 6.3.1.

Dopo aver compilato i campi del form, l'utente può iniziare la fase di acquisizione delle immagini. Premendo il tasto "Ok" verrà salvato un file XML con tutti i

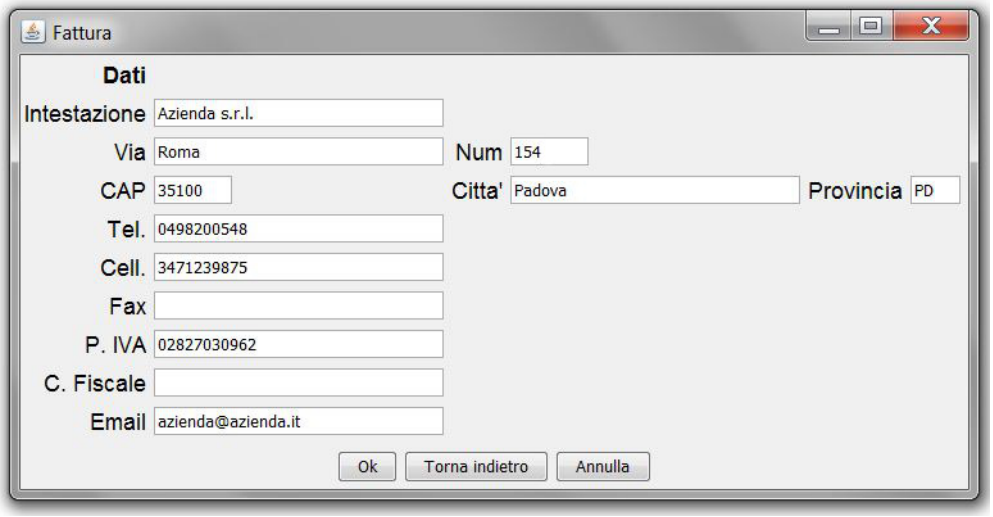


Figura 7.6: Form di inserimento dati

dati inseriti dall'utente e sarà chiuso il form per dar spazio ad una nuova finestra. La vista che ci si troverà di fronte è quella riportata in figura 7.7 e si compone di tre elementi principali:

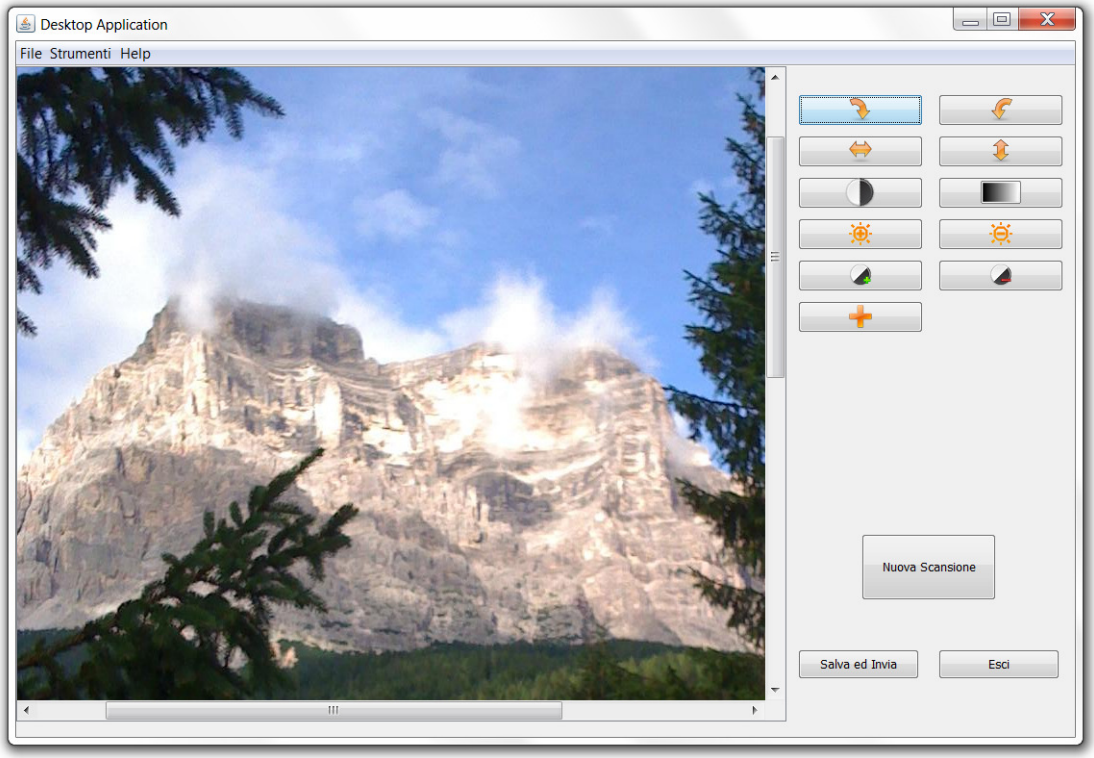


Figura 7.7: Finestra di acquisizione dell'immagine

1. nella parte superiore è presente la barra del menù. Dispone dei comandi per iniziare la scansione, salvare l'immagine ed inviarla al sistema documentale, chiudere il programma, cambiare il dispositivo di default per la scansione delle immagini ed effettuare modifiche di tipo grafico all'immagine acquisita;

2. nella parte centrale è collocato il pannello che consentirà di visualizzare il documento acquisito;

3. nel lato sinistro è posizionato il pannello con gli strumenti necessari alla modifica delle immagini (rotazione, ribaltamento, contrasto, luminosità, modifica dei colori), all'inizio di una nuova scansione, al salvataggio e successivo invio del documento acquisito ed all'uscita dal programma.

Con un clic sul tasto "Salva ed invia" verrà salvato il file in locale e ne verrà inviata una copia al sistema documentale Alfresco. Se l'operazione va a buon fine ed il file viene inviato correttamente, apparirà una finestra di dialogo che avvertirà l'utente che l'azione è stata eseguita correttamente, in caso contrario sarà mostrato un messaggio d'errore ed il file risulterà non inviato. Se alla chiusura del programma vi saranno alcuni file che non è stato possibile trasmettere al back office, al successivo riavvio del software verrà segnalata, attraverso una dialog box (figura 7.8), la presenza di immagini da inviare.

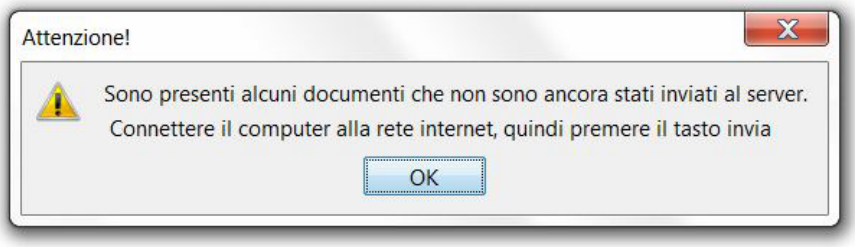


Figura 7.8: Avviso della presenza di file da inviare

Sarà data all'operatore la possibilità di selezionare i documenti da spedire, attraverso una finestra di navigazione del file system, realizzata per mezzo dell'oggetto Java *JFileChooser* (figura 7.9).

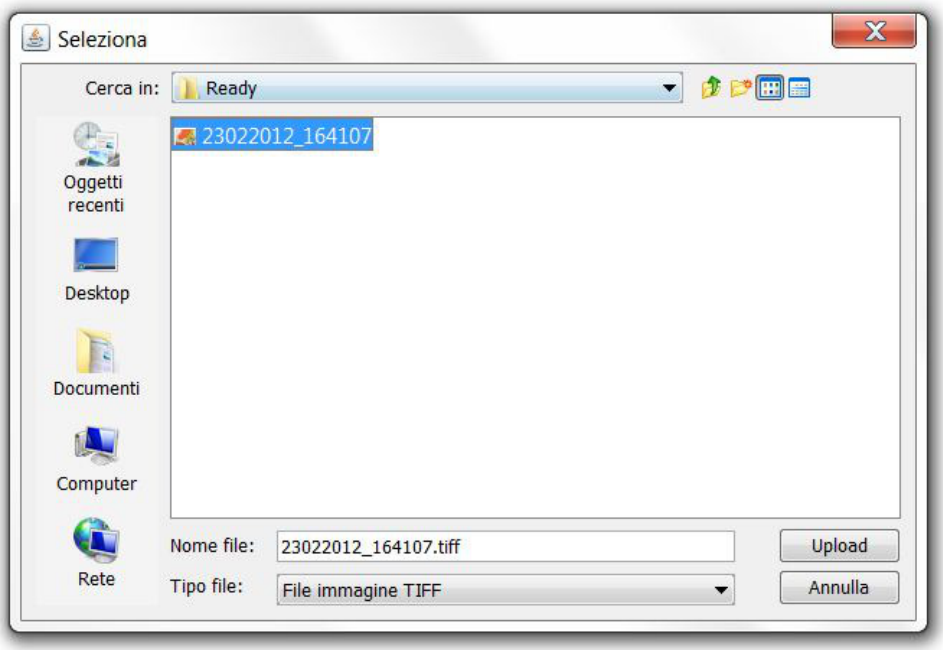


Figura 7.9: FileChooser per la selezione di un file

Capitolo 8

Manuale Tecnico

In questo capitolo verrà presentata un'analisi dettagliata del codice scritto per la realizzazione del programma; sarà illustrata ogni singola classe e, per ciascuna di esse, si cercherà di analizzarne il contenuto, magari illustrando nello specifico quelli che sono i metodi più significativi. Saranno inoltre motivate le scelte implementative effettuate, analizzandone vantaggi e svantaggi. Tale capitolo, puramente tecnico, ha l'unico obiettivo di creare una sorta di manuale per coloro i quali volessero apportare delle modifiche o delle migliorie al codice.

8.1 La classe AppMain

La classe principale del programma è `AppMain` e contiene al suo interno il solo metodo `main`. Il metodo `main` svolge quattro semplici azioni:

- assegna un particolare *look and feel* (aspetto grafico) all'applicazione;
- crea le cartelle necessarie al programma per salvare i log, le immagini scansionate ed i file di configurazione (tale azione è eseguita solo alla prima esecuzione assoluta);
- crea un'icona nella *System Tray* per interagire con l'applicazione;
- avvia la schermata di benvenuto dell'applicativo.

Tutte le cartelle di appoggio per il programma vengono create all'interno della *home* directory dell'utente, il cui *path* è restituito come stringa dal metodo Java `System.getProperty("user.home")`. Nei sistemi *Windows* tale directory

si trova all'indirizzo *<root>/Documents and Settings/<username>* (in Windows XP) o *<root>/Users/<username>* (Vista e 7), nei sistemi *Unix* si trova in *<root>/home/<username>*, nei sistemi *MAC OS* in */Users/<username>*. Le cartelle necessarie sono:

- *DesktopApplication*, è la root folder e contiene tutte le sottocartelle del software;
- *Config*, che contiene i file di configurazione per l'applicativo;
- *Log*, che conserva tutti i Log di scansione. Questa cartella è fondamentale per la creazione dello storico, perché l'elenco con i dettagli di ogni scansione viene creato proprio dalla lettura dei file di log;
- *Storico*, contiene al suo interno la sottocartella *Ready* (in cui vengono salvati i documenti prima di essere inviati) e *Sent* (che conserva i file TIFF inviati al documentale);
- *Temp_image*, cartella temporanea che viene ricreata ad ogni sessione di scansione. In essa vengono posti i file TIFF contenenti le singole pagine scansionate, prima che essi siano "aggregati" in un unico file multipagina;
- *Xml*, che ha al suo interno i file XML che vengono processati dagli oggetti della libreria CookSwing.

Per quanto riguarda l'aspetto esteriore delle finestre (in termine tecnico *look and feel*), l'applicazione fa uso di quello proprio del sistema su cui viene eseguita `UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());` è possibile comunque impostare uno dei *look and feel* predefiniti, messi a disposizione dalle librerie Java (Nimbus, Metal, ecc.), oppure crearne uno *ad hoc* per le proprie esigienze ed i prori gusti, personalizzando le varie componenti (bottoni, pannelli, scroll bar, barra del menù, ecc).

L'oggetto Java `SystemTray`[1], che identifica la system tray del sistema, permette la visualizzazione della `java.awt.TrayIcon`[2]. La TrayIcon non è solo un'immagine che viene mostrata durante l'esecuzione dell'applicazione, ma consente di

[1] `http://docs.oracle.com/javase/6/docs/api/java/awt/SystemTray.html`
[2] `http://docs.oracle.com/javase/6/docs/api/java/awt/TrayIcon.html`

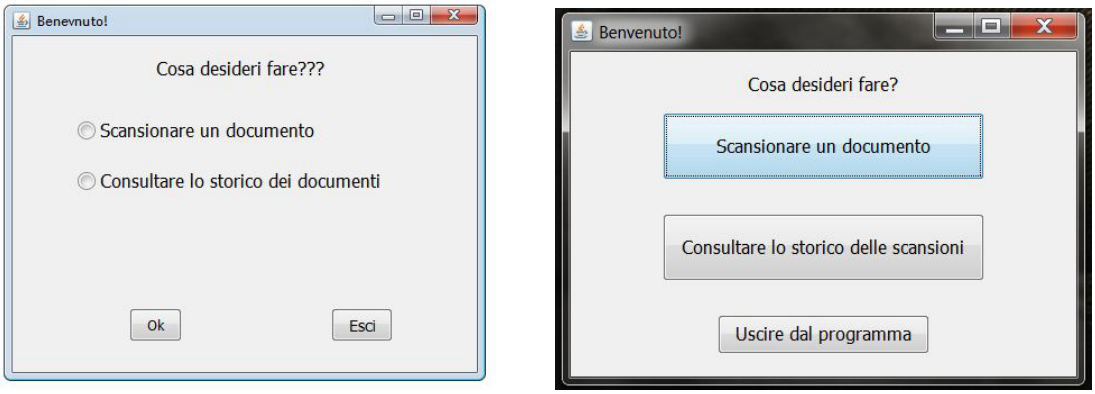
- interagire con il programma, grazie ad un pop-up menù personalizzabile (implementato di defualt ed attivabile con un clic del tasto destro del mouse);

- mostrare messaggi all'utente (visualizzati all'interno di "fumetti" che escono dall'icona) attraverso il metodo `displayMessage(String title, String text, MessageType messageType)`;

- generare dei `MouseEvent` e supportare l'aggiunta di *listener*, per gestire gli eventi generati dal mouse.

L'ultima istruzione eseguita nel `main` è quella che crea un'istanza della classe ***WelcomeFrame***, il cui costruttore ha il compito di visualizzare sullo schermo la finestra di benvenuto del programma. Tale finestra era stata inizialmente progettata per mostrare un pannello contenente, al centro, dei bottoni radio (`JRadioButton`) e nella parte bassa i due classici `JButton` "Ok" ed "Esci"; i radio button sono però leggermente meno "user-friendly" rispetto ai pulsanti standard ed il motivo è presto detto. Se il pulsante standard richiede un semplice clic per essere "selezionato", il radio button ne richiede due: uno per la selezione del bottone (che avendo un'area di selezione molto piccola, necessità di una maggior precisione dell'utente), uno per la conferma (pulsante "Ok"). Ispirati dunque dall'esperienza introdotta dai sistemi touchscreen (che presentano, di norma, interfacce minimali e "pulite" e pulsanti grandi), in cui l'utente con un semplice tocco esegue l'azione desiderata, si è pensato di sostituire la vecchia interfaccia con una contenente tre soli `JButton`. Due hanno dimensioni più grandi e sono legati alle due azioni principali consentite dal programma (effettuare una scansione e visualizzare lo storico dei documenti scanionati), il terzo è leggermente più piccolo e permette la chiusura dell'applicazione.

A ciascuno dei pulsanti è associato un oggetto dell'interfaccia `ActionListener` che serve a "catturare" gli eventi che si verificano sui bottoni (i clic del mouse, ad esempio). L'implementazione dell'interfaccia avviene direttamente nella firma del metodo `addActionListener(ActionListener listener)`, tuttavia sarebbe formalmente più corretto che questa avvenisse in un'appostita classe che implementi `ActionListener`; la soluzione che ho proposto è tuttavia, a mio avviso, più "efficace" (l'ho utillizzata spesso all'interno del codice) perché consente di vedere subito quale azione sia legata a quel `JButton`, senza dover andare in cerca della classe all'interno del codice.



(a) Prima soluzione

(b) Soluzione attuale

Figura 8.1: Finestra di benvenuto

Listing 8.1: Assegnazione di un ActionListener ad un JButton

```
1 JButton exitButton = new JButton();
2 exitButton.addActionListener(new ActionListener() {
3     public void actionPerformed(ActionEvent e) {
4         Tools.doExit(0);
5     }
6 });
```

La classe `WelcomeFrame` non contiene altri elementi rilevanti: il suo costruttore crea un oggetto Java di tipo `JFrame` ed vi aggiunge un `JPanel` coi relativi `JButton`, ottenuti dall'invocazione del metodo `createWelcomePanel()`.
Dalla classe `WelcomeFrame`, tramite il bottone "Consultare lo storico delle scansioni" si invoca il costruttore della classe `AcquiredItemsTable()`, che verrà ora analizzata.

8.2 La classe AcquiredItemsTable

La classe ***AcquiredItemsTable()*** è la classe preposta alla costruzione della tabella in cui viene mostrato l'elenco dei documenti scansionati. Il costruttore della classe inizializza un nuovo oggetto di tipo `JFrame` su cui è definito un layout di tipo `BorderLayout` (6.1). In corrispondenza di `BorderLayout.CENTER` è posizionato il `JPanel` che contiene la `JTabel`, mentre nella sezione `BorderLayout.SOUTH` è situato un `Box` contenente dei `JRadioButton` che fungono da collegamento alle

pagine della tabella.
Il costruttore della classe `AcquiredItemsTable` invoca il metodo `createPanel()`, che restituisce un oggetto di tipo `JPanel` su cui sono collocati due *container*: uno di tipo `JScrollPane` che contiene l'oggetto `JTable` ed uno di tipo `JPanel` con i bottoni "Torna indietro" ed "Esci".
Il metodo `createTable()` è preposto alla creazione della tabella: colora le righe, imposta il font ed il colore di background per l'header, centra gli elementi all'interno delle celle. Per assegnare un colore di sfondo alle righe della tabella, è stato necessario sovrascrivere il metodo `prepareRenderer(TableCellRenderer tcr, int row, int column)` della classe `javax.swing.JTable` che, al renderer[3] di default delle celle, applica alcune regole per far si che gli sfondi risultino essere di colore alternato.
Il metodo `showPages(int itemsPerPage, int currentPageIndex)` ha il compito di visualizzare la pagina corrente e gestire i collegamenti alle altre pagine; richiede in input il numero di elementi per pagina e l'indice della pagina su cui ci si trova. Al suo interno vengono richiamati due metodi:

- `JRadioButton createLinks(int itemsPerPage, int current, int target)`, che è il vero artefice della creazione dei collegamenti alle singole pagine;

- il metodo `JRadioButton createRadioButtons(int itemsPerPage, int target, String title)`, che crea i link alla pagina "Precedente" e "Successiva".

Ciascuno dei `JRadioButton`, creati con uno dei metodi succitati, implementa `actionPerformed(ActionEvent e)` richiamando il metodo `showPages(int itemsPerPage, int currentPageIndex)`.
All'interno di `createLinks(int itemsPerPage, int current, int target)` viene sovrascritto inoltre il metodo `void fireStateChanged()` appartenente alla classe `AbstractButton`[4], che assegna una colorazione diversa a seconda che i collegamenti siano abilitati/disabilitati, premuti o selezionati.
Per concludere si trova il metodo `RowFilter filter(final int itemsPerPage, final int target)`: è l'artefice della paginazione della tabella e sovrascrive il metodo `boolean include (Entry entry)` che indica quali item devono essere

[3]Il renderer può essere pensato come ad un oggetto che si occupa della visualizzazione degli aspetti grafici degli elementi di una cella

[4]La classe `AbstractButton` definisce i comportamenti dei pulsanti e degli elementi dei menù

visualizzati (in questo specifico caso, mostra 10 elementi per pagina). Gli oggetti della classe `RowFilter` sono utilizzati proprio per filtrare quegli elementi, appartenenti a liste od elenchi, che non devono essere visibili. Quando un filtro di tipo `RowFilter` è associato ad una tabella, tale filtro è applicato ad un'intera riga (e può essere indotto da regole definite sui contenuti di una qualsiasi cella).

8.3 Il package desktopApp.table

Nel package **it.eproject.desktopApp.table** sono state collocate le classi di appoggio per la costruzione della tabella. La classe `RadioBttonUI` definisce gli aspetti grafici dei bottoni radio che fungono da collegamento alle varie pagine: nello specifico, tale classe si preoccupa di creare l'effetto "sottolineato" ai radio button, quando il mouse passa sopra il bottone (`ButtonModel.isRollover()` rileva il passaggio del mouse sul pulsante).
La classe `CustomModel` estende la classe `DefaultTableModel`, che a sua volta implementa l'interfaccia `TableModel`[5]. Il motivo per cui si è deciso di estendere la `DefualTableModel` è puramente pratico: anziché dover implementare tutti i metodi dell'interfaccia `TableModel`, a cui non ero interessato, è risultato più semplice sovrascrivere il costruttore della classe `DefualTableModel`, in modo che questo si occupasse di inserire nella `JTable` i dati relativi alle scansioni effettuate. L'acquisizione delle *entry* da importare nella tabella è fatta da un'istanza della classe `XMLParser` (di cui si analizzerà il contenuto più avanti nel capitolo, ma si è già accennato anche in 6.3), che si occupa di processare i file XML di log per ricavarne informazioni.
All'interno del package `it.eproject.desktopApp.table` vi sono poi altre due classi: `ButtonEditor` e `ButtonRenderer`. La prima estende `DefaultCellEditor`, che è la classe di default per la gestione delle celle delle `JTable`; `ButtonEditor` ne sovrascrive il costruttore per far sì che venga creato, nella colonna dei dettagli, un bottone (un `JCheckBox`) a cui aggiungere un listener che risponda ai clic del mouse. Alla ricezione del clic, esso crea un'istanza della classe `ShowDetails`, che realizza un `JFrame` che riporta i dettagli della scansione. `ButtonRenderer` estende invece la classe `JButton` ed implementa `TableCellRenderer`, che è l'interfaccia che definisce i metodi richiesti da ciascun oggetto che voglia essere inserito nelle celle di una `JTable`. Per assegnare un allineamento (centrato) ed un'icona (la lente di ingrandimento) ai bottoni della colonna "Dettagli" viene realizzato quin-

[5] L'interfaccia `TableModel` specifica i metodi con cui la `JTable` interroga i dati in essa inseriti.

di il metodo `getTableCellRendererComponent`.

8.4 La classe SelectDocsType

Qualora alla prima schermata si fosse scelto di "Scansionare un documento", il software avrebbe coinvolto la classe `SelectDocsType` ed a seguire la classe `ImageEditor`, che è di fatto cuore del programma.
La classe `SelectDocsType` è particolarmente interessante perché fa uso della libreria *CookSwing*, di cui si è parlato in 6.2 e 6.2.2. Il costruttore crea un'oggetto di tipo CookSwing che esegue il render del file *DocsType.xml*[6]. La finestra visualizzata come risultato dell'invocazione del costruttore della classe `SelectDocsType` presenta cinque pulsanti: tre sono grandi e consentono la selezione di un particolare tipo di documento (Bonifico, Fattura, Curriculum), gli altri due, più piccoli e situati più in basso, consentono di tornare indietro o di abbandonare il programma. A ciascuno dei tre bottoni più grandi, è associata un'azione collegata al clic del tasto sinistro del mouse; tale azione, che consente l'apertura del form relativo al documento prescelto, è definita nel file *DocsType.xml* dal parametro associato all'attributo *actionlistener*. Il codice Java per la gestione di tale comando è del tutto simile a quello già visto nel listato 8.1 e consiste, ancora una volta, nell'implementare il metodo `actionPerformed(ActionEvent e)` dell'interfaccia `ActionListener`.
Restando all'interno di `SelectDocsType`, possiamo trovare ancora i metodi dedicati all'uscita dal programma ed al salvataggio dei dati inseriti nel form, il cui codice è stato precedentemente analizzato e compare nel listato 6.8.
Una volta compilati tutti i campi d'inserimento del form, si può procedere con l'acquisizione del documento, che verrà analizzato nel dettaglio nella prossima sezione.

8.5 La classe ImageEditor

La classe **ImageEditor** è senza dubbio la più "importante" del progetto, perché attraverso essa si compiono le due azioni per cui l'intero software è stato pensato: la scansione e l'invio dei documenti acquisiti al sistema documentale. È la prima classe su cui ho iniziato a lavorare ed è quella che ha richiesto gli sforzi maggiori,

[6]Tale file deve essere realizzato seguendo le regole indotte da *CookSwing*

poiché realizza una serie di meccanismi che necessitano di essere trattati con attenzione.
Il costruttore della classe svolge alcune azioni preliminari:

- inizializza un contatore (per il conteggio delle pagine scansionate);
- seleziona la periferica di scansione predefinita (attraverso il metodo `Source setSource(String path)`, il cui nome è scritto sul file *source.xml*);
- istanzia un esemplare della classe `JFrame` per la visualizzazione della barra del menù, del pannello dei pulsanti e di quello che conterrà l'immagine scansionata;
- controlla, attraverso il metodo `boolean checkEmptyFolder(String path)`, che la cartella dei file da inviare (*DesktopApplication/Storico/Ready*) sia vuota. In caso contrario avvisa l'utente con un messaggio, visualizzato in un frame creato dal metodo `void warningFrame(String title, String message)` (si faccia riferimento alla figura 7.8).

Il metodo `void addComponentsToMainFrame()` realizza l'intera interfaccia grafica: nella parte superiore del frame è posizionata la barra del menù, a sinistra c'è il pannello con i pulsanti per la modifica dell'immagine e nella parte centrale c'è il pannello che conterrà l'immagine del documento acquisito (è di tipo `JScrollPane`); nell'angolo in basso a destra si trovano i pulsanti di scansione, invio, uscita. I componenti (pannelli e pulsanti) sono disposti secondo un layout di tipo **FormLayout**; esso non è fornito dalle librerie standard Java, ma viene introdotto nel progetto grazie alle librerie open source *jgoodies*. Il layout `FormLayout` è stato ideato per essere uno strumento preciso, flessibile e potente, ma allo stesso tempo facile da comprendere ed applicare. Il codice che viene riportato (listato 8.2), a titolo d'esempio, posiziona un oggetto di tipo `JButton` su un pannello a cui viene applicato il `FormLayout`.
Il costruttore della classe `FormLayout` definisce per prima cosa una griglia, specificandone il numero e le dimensioni di colonne e righe; in seconda battuta viene aggiunto un componente alla griglia, in una posizione specifica.

Listing 8.2: Pannello con layout di tipo FormLayout

```
1 buttonPanel.setLayout(new FormLayout("30dlu, 30dlu, 10dlu, 30
    dlu, 30dlu",
2         "40dlu, 25dlu, 14dlu, 25dlu, 19dlu, 6dlu"));
```

```
3
4  scanButton.setText("Nuova Scansione");
5  scanButton.addActionListener(new ActionListener() {
6      public void actionPerformed(ActionEvent evt) {
7          scanAction(evt);
8      }
9  });
10
11 buttonPanel.add(scanButton, CC.xywh(3, 3, 1, 2, CC.CENTER, CC.
     CENTER));
```

Analizzando il codice, osserviamo che vengono create 5 colonne e 7 righe, le cui dimensioni sono tutte specificate con l'unità di misura *dlu* (*Dialog Unit*). Le *Dialog Unit* sono unità "particolari" e sono una caratteristica del `FormLayout`: esse hanno il compito di preservare le proporzioni qualora venisse modificata la risoluzione del componente che implementa tale layout. Ciascun componente gestito da un `FormLayout` è associato ad un'istanza di `CellConstraints` che specifica dove esso debba essere visualizzato e come debba essere allineato.
Creata la griglia, il bottone può essere aggiunto con il metodo `add(Component c, CellConstraints cc)` nella posizione specificata dai parametri dell'oggetto `CellConstraints`. Nel codice riportato si noterà che viene fatto uso della classe CC, che è una classe *factory* per la creazione di oggetti `CellConstraints`. L'alternativa sarebbe stata quella di definire un nuovo oggetto `CellConstraints cc = new CellConstraints()` e poi su di esso applicare i valori che determinano il posizionamento del componente all'interno del layout. Il metodo `CC.xywh(3, 3, 1, 2, CC.LEFT CC.CENTER)` indica che il bottone deve essere posizionato nella colonna 3, alla riga 3, occupa una sola colonna e si estende per due righe. Il suo allineamento poi prevede che sia posizionato a sinistra verticalmente e sia centrato orizzontalmente.

Nella classe `ImageEditor` sono contenuti inoltre i metodi definiti per la realizzazione delle funzioni associate ai pulsanti presenti nella finestra principale del programma (ovviamente le stesse azioni sono eseguibili anche tramite barra del menù). Si riporta quindi una lista dei metodi disponibili, preoccupandosi poi di analizzare quelli più significativi.

- *scanAction(ActionEvent e)* - esegue la scansione del documento;
- *exitAction(ActionEvent e)* - esce dal programma;

- *addPageAction(ActionEvent e)* - aggiunge una pagina al set corrente di scansioni;
- *changeDevAction(ActionEvent e)* - cambia il dispositivo da utilizzare per la scansione;
- *horizontalFlipAction(ActionEvent e)* e *verticalFlipAction(ActionEvent e)* - richiamano i metodi per esegire il ribaltamento (orizzontale o verticale) dell'immagine scansionata;
- *rotateRightAction(ActionEvent e)* e *rotateLeftAction(ActionEvent e)* - richiamano i metodi per ruotare di 90 gradi a destra o a sinistra l'immagine;
- *grayscaleAction(ActionEvent e)* - invoca il metodo che trasforma l'immagine in scala di grigi;
- *blackWhiteAction(ActionEvent e)* - invoca il metodo per la trasformazione dell'immagine in bianco e nero;
- *brightenAction(ActionEvent e)* e *darkenAction(ActionEvent e)* - richiamano i metodi per aumentare o diminuire la luminosità della scansione;
- *contrastIncrAction(ActionEvent e)* e *contrastDecrAction(ActionEvent e)* - invocano i metodi per aumentare o diminuire il contrasto dell'imagine acquisita;
- *resizeAction(ActionEvent e)* - effettua un ridimensionamento dell'immagine;
- *uploadAction(ActionEvent e)* - salva ed invia l'immagine.

Il metodo `scanAction(ActionEvent e)` realizza la scansione del documento. Esso viene invocato alla pressione del bottone "Nuova scansione" ed ha il compito di iniziare una nuova acquisizione. Questo pulsante dà inizio ad un nuovo set di scansioni pertanto, per prima cosa, si occupa di cancellare eventuali immagini importate in acquisizioni precedenti (presenti nella cartella *Temp_ image*); azzera quindi il contatore delle pagine e procede con la scansione. Il codice relativo alla scansione è già stato illustrato in 3.4; a quanto già detto va aggiunto un solo dettaglio e riguarda il formato con cui il file viene salvato: si è deciso di inviare al back office documenti di tipo TIFF, poiché questo particolare formato consente il salvataggio di file multipagina. Per ottenere i file TIFF, che non sono nativamente supportati dalle librerie Java, si è fatto uso della libreria **aspriseTIFF** che

è nata a corredo della libreria `JTwain` (presentata in 3.3), proprio con l'intento di supportare questo particolare formato di file. In particolare, si è fatto uso del metodo `createTIFFFromImages(BufferedArray array,int conv, int comp, File f)` che richiede come parametro un array di `BufferedImage` (che saranno trasformate in un unico file TIFF), due oggetti di tipo `int` che specifichino la conversione cromatica ed il tipo di compressione da applicare alle immagini ed il file di output.

Il metodo `changeDevAction(ActionEvent e)` richiama l'interfaccia utente già realizzata dalla libreria `JTwain` per la selezione della periferica di acquisizione delle immagini (`SourceManager.instance().selectSourceUI()`) e salva il nome del dispositivo selezionato direttamente nel file *source.xml*; tale file viene letto ad ogni avvio del programma per selezionare la periferica da utilizzare.

Per finire analizziamo il metodo `uploadAction(ActionEvent e)` che esegue il salvataggio e l'invio al back office dei documenti acquisiti. Il metodo può essere richiamato in due istanti diversi: prima dell'esecuzione della scansione (nel caso siano rimasti file da inviare alla precedente chiusura del programma) o dopo l'acquisizione del documento.
Si immagini che, mentre si sta eseguendo una scansione, per un guasto tecnico sulla rete non sia possibile inviare il file d'immagine appena creato, che è rimasto salvato nella cartella *DesktopApplication/Storico/Ready*. Premendo il pulsante "Salva ed invia" si invoca `uploadAction(ActionEvent e)` che per prima cosa effettua un controllo nella cartella *Ready* alla ricerca di eventuali file non ancora inviati. Se la cartella non risulterà vuota, allora verrà visualizzato un `FileChooser` che consentirà di selezionare il file da inviare (riferimento all'immagine 7.9). A seguire due listati: il primo mostra la realizzazione del `FileChooser`, il secondo svela una parte di codice del metodo `uploadAction(ActionEvent e)`.

Listing 8.3: Finestra di selezione dei file da inviare

```
1 JFileChooser fileUploaderFrame() {
2     JFileChooser fileUploader = new JFileChooser();
3     fileUploader.setDialogTitle("Seleziona");
4     tiffFilterCreator(fileUploader);
5     fileUploader.setApproveButtonMnemonic(KeyEvent.VK_U);
6     fileUploader.setCurrentDirectory(new File(basePath + "\\
        Storico\\Ready\\"));
7     fileUploader.setAcceptAllFileFilterUsed(false);
```

```
8       fileUploader.setApproveButtonToolTipText("Scegl uno o pi?
          file da caricare");
9       return fileUploader;
10  }
```

Listing 8.4: Invio dei file salvati

```
1   if (Tools.checkEmptyFolder(System.getProperty("user.home")+ "
      \\DesktopApplication\\Storico\\Ready\\")) {
2       Frame frame = new Frame();
3       JDialog.setDefaultLookAndFeelDecorated(true);
4       JFileChooser fileUploader = Tools.fileUploaderFrame();
5       int returnVal = fileUploader.showDialog(frame, "Upload");
6       if (returnVal == JFileChooser.APPROVE_OPTION) {
7     File file = fileUploader.getSelectedFile();
8     Tools.connectToAlfresco(file.getName(), file.getPath());
9       }
10  }
```

Se invece il metodo `uploadAction(ActionEvent e)` è premuto a seguito di una scansione, verranno eseguite le seguenti azioni l'una dopo l'altra:

- si crea un array di BufferedImage (che sarà passato come parametro al metodo `TIFFWriter.createTIFFFromImages(BufferedImage a, File f)`) in cui sono posti i riferimenti agli oggetti di tipo BufferedImage, creati a partire dai file TIFF.
- si salva il file TIFF creato, con il nome "data_ora.tiff";
- si crea il log di salvataggio;
- si invia il file.

Il seguente frammento riporta la seconda parte del metodo `uploadAction (ActionEvent e)`.

Listing 8.5: Salvataggio ed invio del file

```
1   try {
2       // converto i singoli file tiff in oggetti BufferedImage
3       BufferedImage[] bufArray = new BufferedImage[countPages];
4       for (int i = 0; i < countPages; i++) {
5     bufArray[i] = ImageEditorTools.TIFFtoBUFF(new File(path+ "
        img_" + i + ".tiff"));
```

```
6       }
7
8       //Salvo un file tiff multipagina
9       File f = new File(System.getProperty("user.home")+ "\\
          DesktopApplication\\Storico\\Ready\\" + dataScansione[1]
          + "_" + oraScansione[1] + ".tiff");
10      Tools.MultiPageTIFF(bufArray, f);
11      tiffName = f.getName();
12      tiffPath = f.getPath();
13      String savingLogName = dataScansione[1] + "_" +
          oraScansione[1]+ ".xml";
14      Tools.createLogXML(userName, dataScansione[0],
          oraScansione[0],"" + countPages, tiffPath, tiffName,
          savingLogName);
15      Tools.connectToAlfresco(tiffName, tiffPath);
```

8.6 Le classi Tools ed ImageEditorTools

Le classi **Tools**, **ImageEditorTools** ed **XMLParser** sono tre classi ausiliarie, che forniscono una serie di metodi utili alla realizzazione del programma. Analizzeremo quindi il contenuto di ciascuna classe e vedremo nel dettaglio quelli che sono i metodi più significativi.

Fanno parte della classe **XMLParser** tutti quei metodi che vengono utilizzati per la lettura dei file XML e l'acquisizione delle informazioni in essi contenute. Essa presenta due costruttori: il primo è creato *ad hoc* per la lettura dei file di log delle scansioni e richiede come parametro d'ingresso una stringa che specifichi il percorso della cartella che contiene tali file; il secondo invece è stato ideato per la lettura del file *source.xml* (che contiene il nome del dispositivo di scansione da utilizzare) e come parametro di input richiede l'oggetto Java `File` che punta al file *source.xml*. I metodi definiti in `XMLParser` sono i seguenti:

- `getCount()` - calcola il numero totale di file xml di log;
- `getUsername()` - restituisce un array di stringhe, ciascuna delle quali riporta il valore contenuto nel tag <nome>;
- `getDate()` - restituisce un array di stringhe, ciascuna delle quali riporta il valore contenuto nel tag <data>;

- `getTime()` - restituisce un array di stringhe, ciascuna delle quali riporta il valore contenuto nel tag <ora>;
- `getPages()` - restituisce un array di stringhe, ciascuna delle quali riporta il valore contenuto nel tag <pagine>;
- `getFilename()` - restituisce un array di stringhe, ciascuna delle quali riporta il valore contenuto nel tag <file>;
- `getSource()` - restituisce una stringa contenente il nome del dispositivo.

Riporto, a titolo di esempio, il codice che realizza il costruttore della classe `XMLParser` ed il codice del metodo `String[] getDate()` che esegue un ciclo `for` su tutti i file di log ed acquisisce, dalla lettura di essi, il valore contenuto nel tag <data>, servendosi della libreria JDOM (presentata in 6.3).

Listing 8.6: Acquisizione dei valori contenuti nel tag <date>

```
1  public XMLParser(String filePath) {
2      SAXBuilder builder = new SAXBuilder();
3      path = filePath;
4      File dir = new File(path);
5      String[] list = dir.list();
6  }
7
8  public String[] getDate() {
9      String[] date;
10     try {
11   date = new String[list.length];
12   for (int i = 0; i < list.length; i++) {
13       Document document = builder.build(new File(path + list[i
           ]));
14       Element root = document.getRootElement();
15       Element description = root.getChild("data");
16       date[i] = description.getText();
17   }
18     } catch (JDOMException e) {
19   logger.error("getUsername()", e);
20     } catch (IOException e) {
21   logger.error("getUsername()", e);
22     }
23     return date;
```

```
24 }
```

La classe **ImageEditorTools** contiene i metodi che sono richiamati da quelli presenti nella classe `ImageEditor` e si preoccupano di effettuare fisicamente le operazione di trasformazione sulle immagini acquisite. I metodi presenti in `ImageEditorTools` infatti eseguono rotazioni, ribaltamenti, aumenti o diminuzioni della luminosità, conversioni cromatiche, conversioni di file TIFF in oggetti di tipo `BufferedImage`. Riporto il codice per la rotazione di un'immagine.

Listing 8.7: Rotazione di un'immagine

```
1  public static BufferedImage rotateLeft(BufferedImage
     sourceImage) {
2      int width = sourceImage.getWidth();
3      int height = sourceImage.getHeight();
4      BufferedImage rotatedImage = new BufferedImage(height,
         width, sourceImage.getType());
5      //costruisce l'immagine, pixel per pixel
6      for (int i = 0; i < w; i++) {
7    for (int j = 0; j < h; j++) {
8        //setRGB imposta un pixel del BufferedImage con
9              //il valore specificato da getRGB.
10             rotatedImage.setRGB(j, w - 1 - i, sourceImage.
                 getRGB(i, j));
11   }
12     }
13     return rotatedImage;
14 }
```

Nel seguente listato (8.8), invece, si converte un'immagine a colori in un'immagine in scala di grigi. Si presti attenzione all'utilizzo della classe `Graphics2D`[7], fondamentale per il rendering di forme, testi ed immagini bidimensionali. Il metodo `createGraphics()` crea un oggetto di tipo `Graphics2D` che può essere utilizzato per disegnare nella `BufferedImage` su cui tale metodo è invocato, mentre il metodo `drawImage(sourceImage, 0, 0, null)` è lo strumento che effettua la colorazione, a partire dal vertice in alto a sinistra dell'immagine (le cui coordinate nello spazio sono indicate dalla coppia di interi 0,0).

[7] `http://docs.oracle.com/javase/6/docs/api/java/awt/Graphics2D.html`

Listing 8.8: Conversione di un'immagine, da colori a scala di grigi

```
1  public static BufferedImage convertToGrayscale(BufferedImage
     sourceImage) {
2      int width = sourceImage.getWidth();
3      int height = sourceImage.getHeight();
4      //si crea una BufferedImage il cui modello di colore
5      // è la scala di grigi
6      BufferedImage grayScaledImage = new BufferedImage(width,
         height, BufferedImage.TYPE_BYTE_GRAY);
7      //Si crea un oggetto di tipo Graphics2D, che può esssere
         utilizzato per
8      // disegnare nella grayScaledImage
9      Graphics2D g2d = grayScaledImage.createGraphics();
10     g2d.drawImage(sourceImage, 0, 0, null);
11     g2d.dispose();
12     return grayScaledImage;
```

Per terminare con l'analisi delle classi, si elencano i metodi contenuti nella classe di supporto **Tools** e si fornisce, per ciascuno di essi, una breve frase esplicativa della loro funzione.

- `createAppFolder()` e `createFolder(String path)` si occupano della creazione delle cartelle necessarie al software per salvare file di configurazione, log ed immagini scansionate;
- `createLogXML(String username, String date, String time, String numberOfPages, String path, String fName, String savingName)` e `createSourceXML(String sourceName)` creano i file XML di log ed il file *source.xml*;
- `date()` e `time()` sono usati per la raccolta di informazioni su data ed ora di scansione;
- `deleteAllFiles(String path)` e `deleteFolder(String path)` eseguono, rispettivamente, la cancellazione dei file presenti in una cartella e l'eliminazione di un'intera cartella;
- `createTrayIcon()` per la crazione dell'icona nella system tray;
- `setSource(String path)` seleziona per l'utilizzo il dispositivo specificato nel file, passatogli come parametro. In caso il file non esista o il dispositi-

vo riportato nel file non sia disponibile, consente una nuova selezione del dispositivo e vi scrive il nome su file;

- `MultiPageTIFF(BufferedImage[] array, File file)` e `saveToTIFF(int n, BufferedImage sourceImage)` concorrono alla creazione di un file TIFF multipagina (il primo) ed alla conversione da `BufferedImage` a file TIFF (il secondo);

- `showImage(BufferedImage sourceImage, JScrollPane scrollPane)` fa sì che la BufferedImage passata come parametro venga visualizzata nel pannello specificato;

- `sentMessageFrame(String message, String titleFrame, ImageIcon icon)` e `warningFrame(String message, String titleFrame)` sono invocati per la creazione, rispettivamente, di `JOptionFrame` ti tipo *INFORMATION_MESSAGE* e *WARNING_MESSAGE*.

- `doExit()` esegue l'uscita dal programma, mostrando un `JOptionPane` di tipo *OK_CANCEL_OPTION* che richiede una conferma prima di chiudere l'applicazione.

Non ritengo necessario riportare il codice di alcuno di questi metodi, poiché, salvo quelli di cui si è già trattato in precedenza, gli altri fanno uso di classi Java note ai più e presentano un'implementazione semplice e lineare, la cui comprensione non richiede particolari sforzi.

8.7 La libreria Log4j e lo strumento Log4E

Log4j è una libreria Java sviluppata dalla *Apache Software Foundation* che consente di implementare un ottimo sistema di logging per controllare il comportamento di un'applicazione in fase di sviluppo; è uno dei sistemi di logging più utilizzato per le applicazioni Java, tanto da essere considerato uno standard *de facto*. Senza dubbio, un file di log è inutile quando si tratta di poche righe di codice, ma diventa essenziale quando il codice diventa più complesso e, soprattutto, quando si lavora in un team. Si procede con la spiegazione dell'utilizzo di *Log4j* e del plugin per l'*IDE Eclipse Log4E*.

Innanzitutto, è necessario importare la libreria *Log4j* all'interno del proprio progetto: per istruzioni riguardanti questa procedura si rimanda alla lettura del paragrafo 5.1. Il secondo passo da compiere è quello di scrivere un *file di properties*

per configurare la libreria e utilizzarla all'interno di un'applicazione. Ciascun file di properties definisce tre entità che svolgono tre diverse funzioni:

- il **Logger** specifica di cosa si voglia tener traccia (quali classi, quali package, ecc.);
- l'**Appender** specifica l'output su cui si vuole visualizzare il log (la console, un file di testo, ecc.);
- il **Layout** specifica il formato di output del log (testo, file xml, file html, ecc.).

Una volta creato il *file di properties*, si può procedere con l'uso della libreria. Per prima cosa va importato il package con le classi necessarie alla creazione del log (`import org.apache.log4j.Logger;`). In seconda battuta si deve creare l'oggetto `Logger` per poterlo utilizzare all'interno del programma. Per far ciò è necessario inserire il seguente comando `private static final Logger logger = Logger.getLogger(MyClass.class)`, facendo attenzione ad impostare il nome della propria classe nel parametro del costruttore. A questo punto si può procedere con l'utilizzazione dell'oggetto *logger* per creare i log attraverso i metodi messi a disposizione dalla libreria. Per portare un esempio, una strategia da me stesso utilizzata è quella che vede l'inserimento, all'interno del blocco `catch`, di un comando del tipo `logger.error("Messaggio")`; tale istruzione, se eseguita, aggiunge al log una voce *ERROR* e segnala che è stata generata e catturata una particolare eccezione (il cui nome è di solito specificato nel parametro di input del metodo).

Si immagini ora di avere centinaia di metodi e, per ciascuno di essi, voler notificare quando vengano richiamati quando terminino la propria esecuzione. Inserire tutte queste righe di codice all'interno delle proprie classi può risultare estremamente noioso, specialmente se i progetti a cui si lavora sono di grandi dimensioni. La procedura poc'anzi descritta può essere automatizzata attraverso l'utilizzo di un plug-in per l'IDE Eclipse: **Log4E**. Esso si occupa di creare dei sitemi di log in maniera del tutto autonoma: cliccando infatti con il tasto destro in un punto specifico del codice sorgente, si può accedere al menù *Log4E* che consente opzioni di inserimento dei messaggi di log all'intera classe, in un punto specifico, ai metodi della classe ed altre opzioni.
L'utilizzo di tale strumento si è rivelato fondamentale in più occasioni, aiutandomi a comprendere in maniera molto rapida quali fossero i comportamenti del pro-

gramma ed in quale punto specifico venivano generate le eccezioni che mandavano in crash l'applicazione.

Conclusioni

Il tirocinio, svolto presso *E-project s.r.l.* nei mesi che vanno da ottobre 2011 a febbraio 2012, è stato decisamente arricchente, consentendomi di mettere in pratica le nozioni apprese nel corso degli anni di studio, oltre ad avermi fornito un primo approccio alla realtà lavorativa. Il progetto al quale ho lavorato è stato assolutamente gratificante perché mi ha permesso di procedere in autonomia per quanto riguarda alcune scelte implementative ed alcuni strumenti utilizzati, senza mai farmi mancare l'appoggio di colleghi competenti e preparati.
Gli obiettivi del tirocinio sono stati raggiunti, producendo una versione funzionante dell'applicazione, seppure ancora "scarna" per quanto riguarda la sua interfaccia grafica.
Eventuali estensioni future si dirigeranno sicuramente verso questo aspetto (il miglioramento della GUI) ed inoltre non è escluso che venga effettuato un adattamento del codice per far sì che il software possa essere eseguito anche su architetture Unix. Ulteriori sviluppi dovrebbero inoltre produrre un *porting* dell'applicazione verso le piattaforme mobili più utilizzate (Android e iOS) per consentire l'acquisizione ed il trattamento delle immagini tramite fotocamera ed il conseguente invio al documentale Alfresco e la progettazione e realizzazione di una *Web application*, lato server, per la consultazione del back-end Alfresco tramite Web service.

Bibliografia

[1] *Gestione documentale in banca*. ABILab, Milano, 2009.

[2] Cay S. Horstmann *Concetti di informatica e Fondamenti di Java - Terza Edizione*. Apogeo, 2005.

[3] Elliotte R. Harold, W. Scott Means *XML Guida di Riferimento*. Apogeo, 2001.

[4] *Java Web Start* `http://docs.oracle.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html`.

[5] *TWAIN* `http://www.twain.org/docs/TWAIN_2_1_Spec.pdf`.

[6] *CMIS* `http://wiki.alfresco.com/wiki/CMIS`.

[7] *Atom Publishing Protocol* `http://bitworking.org/projects/atom/rfc5023.html`.

[8] *Atom Publishing Protocol* `http://atompub.org/`

[9] *Web Services* `http://www.w3.org/2002/ws/`

[10] *WSDL* `http://www.w3.org/TR/wsdl`

Elenco delle figure

Listings

Ringraziamenti

Il primo pensiero va alle due persone a me più care, i miei genitori, supporto costante e fonti inesauribili di fiducia ed affetto. Ringrazio Marco, continuo stimolo per farmi tendere sempre al meglio. Ringrazio davvero di cuore Marica, che mi incoraggia, mi sostiene e, soprattutto, mi apprezza per quello che sono. Gli amici, senza i quali la vita avrebbe tutto un altro sapore: Stefano, Enrico, Davide, Nicola, Giovanni, Sarah, Paola, Talita, Maddalena, Valentina e Giulia. Ringrazio il relatore, Professor Michele Moro. Un grazie a Massimo Businaro, Francesco Businaro, Fabio Valeri, Manuel Spigolon e Simone Pinton. Per finire, un grazie all'U.S.D. Ferri, incredibile compagine di atleti stravaganti.