UNIVERSITÀ DEGLI STUDI DI BERGAMO

SCUOLA DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica

Classe delle lauree in ingegneria dell'informazione

Classe n. L-8

LaTeX acronyms proofreading

Relatore: Chiar.mo Prof. Stefano Paraboschi

Prova finale di Isaac Maffeis

NOME COGNOME

Matricola n. 1041473

ANNO ACCADEMICO

2019/2020

Sommario

Nella frenesia della vita quotidiana abbiamo contratto la cattiva abitudine di scrivere nella maniera più veloce possibile, questo vizio è stato accentuato dagli smartphone, per mezzo dei quali componiamo ogni giorno decine, se non centinaia, di messaggi informali con lo scopo di inviarli senza farci distrarre troppo su quello a cui eravamo concentrati, così il rischio di compiere un refuso è dietro l'angolo. Qualsiasi tastiera di ogni smartphone offre la funzione di correzione, così da correggere queste sviste, di conseguenza ci siamo abituati a questo modo frenetico di scrivere davanti a uno schermo. Se quindi per i messaggi informali composti da smartphone un refuso non costituisce un problema, lo può invece rappresentare in ambito professionale su PC, dove non è sempre possibile disporre di strumenti di correzione automatica.

Lo scopo di questa tesi è quello di fornire uno strumento di correzione di bozze (proofreading) per documenti professionali scritti in linguaggio LaTeX, nello specifico tratta la realizzazione di un plugin per individuare e correggere gli acronimi del documento in questione.

Indice

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione

1.1 WYSIWYG vs WYSIWYM

L'acronimo What You See Is What You Get (WYSIWYG), letteralmente: quel che vedi è ciò che ottieni, si riferisce principalmente agli elaboratori di testo tradizionali che mostrano istantaneamente sullo schermo il risultato finale del documento a cui si sta lavorando, oltre ad essere immediati e semplici da usare, rendono facili le modifiche all'aspetto del testo, come ad esempio l'inserimento di righe vuote o di spazi vuoti aggiuntivi tra le parole. L'editor di testo più popolare che si basa su questo concetto è Microsoft Word. Una variante dell'acronimo è What You See Is All You Get (WYSIAYG), coniato per segnalare che gli utenti più esperti nell'editing sono limitati dall'interfaccia di modifica messa a loro disposizione.

L'acronimo What You See Is What You Mean (WYSIWYM), letteralmente: quel che vedi è ciò che intendi, è l'opposto di WYSIWYG e si riferisce agli editor che non mostrano a schermo l'impaginazione finale, rendendo così l'utilizzo poco immediato a favore di una maggiore comprensione della struttura del documento. Una variante dell'acronimo è You Asked For It You got it (YAFIYGI), ovvero "Hai chiesto questo, lo hai ottenuto". Il linguaggio che più estremizza questo concetto è il linguaggio di markup LaTeX, che permette di sviluppare un testo concentrandosi soltanto sul contenuto e sulla struttura, incaricando l'editor di sviluppare la forma, che comunque potrà essere modificata a piacimento in fase di impaginazione e rendering senza alterare la struttura. Un editor molto diffuso che si serve di questo linguaggio è TeXstudio.

Quando si ha la necessità di redigere un documento professionale come può essere un articolo, una relazione, una tesi di laurea, un libro o anche solo una lettera è buona norma usare un editor WYSIWYM, che oltre a rendere più leggibile e bello esteticamente il documento permette all'autore di risparmiare molto tempo.

1.2 Linguaggio LaTeX

LaTeX consente di produrre documenti professionali con un alto grado di qualità, generare riferimenti incrociati, indici e bibliografie con grande efficienza e flessibilità, gestire in modo ottimale la composizione tipografica di formule matematiche, semplificando notevolmente il lavoro dell'autore.

Gli acronimi WYSIWYG e WYSIWYM poc'anzi (paragrafo 1.1) ci hanno permesso di capire la differenza sostanziale tra un editor come Microsoft Word e uno come TeXstudio, ora offrono uno spunto per dimostrare l'efficienza del linguaggio LaTeX. Prendiamo in considerazione un acronimo in generale, LaTeX ne permette la definizione in un'apposita sezione, successivamente nel testo la prima volta che lo si inserisce viene stampato il suo nome per esteso, seguito dall'acronimo fra parentesi, mentre le volte successive viene stampato solo l'acronimo. Questa semplice, ma utile funzione, non è l'unica arma di LaTeX a nostra disposizione, infatti tra altre funzioni ci permette anche di creare una nota a piè di pagina con il significato dell'acronimo, seguire una convenzione tipografica nello stampare gli acronimi con un corpo più piccolo rispetto al testo, creare un elenco degli acronimi, con quelli effettivamente usati nel testo e con allegato il numero di pagina dove vengono menzionati.

Un documento professionale può contenere svariati acronimi, gestirli con LaTeX è sicuramente la soluzione più ottimale, ma seppur questo linguaggio ci offra molte utili funzioni a riguardo, pecca di un tool che verifica e corregga eventualmente il nome di questi acronimi.

1.3 Acronimi

Definizione di acronimo dalla Treccani [1]:

> Acrònimo s. m. [comp. di acro- e -onimo]. – Nome formato unendo le lettere o sillabe iniziali di più parole, come per es. radar, dall'ingl. ra (dio) d (etection) a (nd) r (anging), o come molti nomi di enti, società, organizzazioni, ditte, prodotti commerciali (ASL, CGIL, FIAT, ecc.); è comunem. detto sigla, rispetto a cui ha sign. più ristretto (la sigla può non costituire un vero e proprio nome, e talora non ha neanche la possibilità di essere letta come parola). Per estens., si chiamano acronimi anche i nomi formati con le sillabe estreme di due parole, come per es. motel, da mo (to-) e (ho) tel.

[1] Vocabolario Treccani on line. Istituto della Enciclopedia Italiana fondata da Giovanni Treccani S.p.A. URL: Https://www.treccani.it/vocabolario/acronimo/

Gli acronimi ricoprono un ruolo importante nella scrittura scientifica, infatti velocizzano la lettura e facilitano la comprensione del contenuto di un argomento, essi inoltre sono facili da ricordare in tutte le lingue.

Molti acronimi sono diventati così comuni da entrare nel nostro linguaggio abituale e vengono usati generalmente dimenticandosi l'origine. E' il caso di laser (light amplification by stimulated emission of radiation, ovvero amplificazione della luce mediante emissione stimolata della radiazione) o di radar (radio detection and ranging, in italiano radiorilevamento e misurazione di distanza). In un utilizzo professionale tuttavia è consigliato usarli con parsimonia, poiché il loro uso eccessivo può confondere e alienare.

Quando scegliamo di utilizzare gli acronimi all'interno di un testo, dobbiamo considerare attentamente quali utilizzare e come definirli, in modo tale che i nostri lettori comprendano appieno il riferimento. Questo è importante perché gli acronimi possono avere più significati, ad esempio ATM può significare Automated Teller Machine oppure Azienda Trasporti Milanesi. Chiaramente, l'uso di acronimi senza una definizione adeguata può creare confusione. Una buona norma per evitare ridondanze è quella di scrivere il nome dell'acronimo per esteso seguito dalla versione breve posta tra parentesi, ad esempio: Virtual Private Network (VPN); in questo modo, è chiaro a chi sta leggendo cosa significano esattamente le lettere che lo compongono.

1.4 Proofreading

Il termine "proofreading" viene dall'ambito scientifico, più precisamente della biologia molecolare e indica un processo che permette di rilevare e successivamente rimuovere l'appaiamento di un nucleotide scorretto all'interno del DNA. Tuttavia questa parola sta diventando sempre più comune per indicare l'individuazione e la correzione di errori formali durante il controllo precedente alla pubblicazione, si tratta di un esame piuttosto superficiale del testo, ma è sempre necessario. Da non confondere con la fase di editing, nella quale lo scopo è quello di migliorare la scorrevolezza delle frasi, cambiando talvolta soltanto l'ordine delle parole o preferendo l'utilizzo di un certo termine rispetto a un altro. Il processo di proofreading è quindi successivo a quello di content editing e consiste nella revisione e correzione di bozze finalizzata all'identificazione di precise categorie di errori, in particolare di refusi ed errori di battitura, errori grammaticali, errori sintattici, omissioni e frasi saltate, errori nelle numerazioni o nei nomi di luoghi, persone e società, problemi di formattazione ed errori di punteggiatura. Durante questa fase non si effettua un controllo della terminologia salvo in caso di errori eclatanti, ma soprattutto non si interviene sullo stile con cui sono presentati i contenuti.

1.5 LaTeX Acronyms Proofreading

Abbiamo appena visto su cosa si basa il linguaggio LaTeX (paragrafo 1.2), il concetto di acronimo (paragrafo 1.3) e cosa si intende per proofreading (paragrafo 1.4), ora concentriamo queste 3 nozioni per introdurre l'obiettivo di questa tesi: un correttore di bozze per acronimi presenti in un documento professionale scritto utilizzando il linguaggio LaTeX.

Questo utile strumento nasce con lo scopo di aiutare l'autore di un testo ad effettuare il processo di proofreading, controllando automaticamente tutti gli acronimi presenti nel documento e mostrando all'utente quelli ritenuti errati, per questi il tool espone la tipologia di errore e offre una o più possibili correzioni. La scelta finale spetta comunque all'autore che può decidere di correggere l'acronimo con la soluzione proposta, sfogliare le varie correzioni, aggiungere l'acronimo nel dizionario oppure ignorare l'errore. In aggiunta a queste funzioni è presente l'operazione di correzione automatica, che si incarica di correggere automaticamente il testo con le correzioni ritenute più opportune.

LaTeX Acronyms Proofreading mira a ridurre le lacune degli editor WYSIWYM, offrendo funzionalità comuni a molti editor WYSIWYG.

Capitolo 2

Caratteristiche

2.1 Funzionalità

LaTeX Acronyms Proofreading è un plugin che viene chiamato in causa una volta portato a termine il documento in linguaggio LaTeX con formato .tex. Subito dopo averlo aperto il programma mostra una lista di acronimi ritenuti errati, se ce ne sono, valutati sulla base di un dizionario. L'utente può così prendere visione degli errori commessi in fase di scrittura e selezionarli nella lista, la quale pone gli acronimi in ordine cronologico di come sono definiti nel documento e li classifica in base al acronimo stesso assieme al suo nome breve, se presente. Quando un acronimo viene selezionato il plugin mostra all'utente il suo nome esteso, il problema riscontrato e una lista di soluzioni proposte. Di default il programma seleziona la correzione ritenuta più appropriata, ma l'utente può sempre preferirne un'altra selezionandola dalla lista delle correzioni. A questo punto l'utilizzatore può decidere se correggere l'errore con la correzione selezionata, ignorare l'errore oppure aggiungere l'acronimo nel dizionario, in modo tale che il programma non rilevi lo stesso errore in utilizzi successivi su altri documenti. In seguito a una di queste scelte l'acronimo sparisce dalla lista degli errori e l'utente può passare a controllare il successivo. Una menzione particolare va fatta per gli acronimi considerati errati perché non trovati nel dizionario, infatti in questo caso il plugin non propone alcuna correzione e, oltre all'opzione di ignorare l'errore, offre solo la possibilità di aggiungere l'acronimo nel dizionario. In qualsiasi istante l'utilizzatore può passare a visualizzare la lista completa degli acronimi, che contiene tutti gli acronimi presenti nel documento, corretti, errati o corretti dal plugin, in questo modo può tenere traccia delle correzioni eseguite o rimuovere acronimi erroneamente aggiunti nel dizionario.

Per velocizzare ulteriormente la procedura di proofreading, il tool mette a disposizione la possibilità di correggere automaticamente tutti gli acronimi ritenuti errati con la correzione reputata più opportuna per ognuno, l'utente può usufruire di questa funzionalità in

ogni momento, anche dopo aver corretto autonomamente parte degli acronimi errati, che non verranno perciò presi in carico dal correttore automatico.

2.2 Interfaccia grafica

L'interazione utente-plugin si svolge totalmente tramite un'intuitiva interfaccia grafica. La lista degli errori (o la lista degli acronimi completa in base a cosa si è selezionato), i campi descrittivi e i pulsanti sono ben visibili e semplici da interagirci. Messaggi pop-up se necessario informano l'utente circa l'esito delle operazioni e in caso di errore ne specificano la natura.

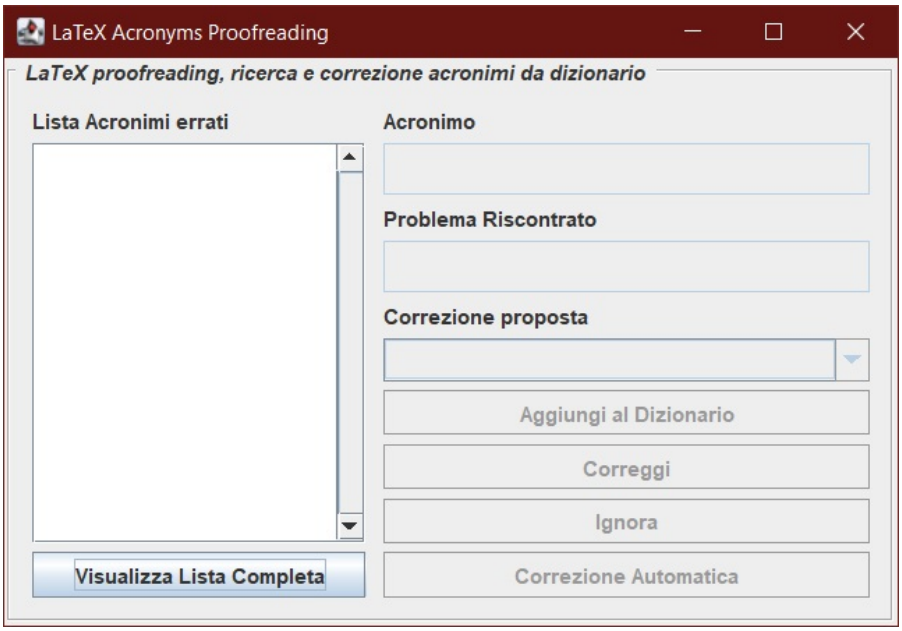

Figura 2.1: Interfaccia grafica vuota senza acronimi

La Figura 2.1 mostra come è formata questa interfaccia grafica, attualmente senza alcun acronimo, per questo motivo le liste (dettaglio nel paragrafo 2.2.1) e i campi (dettaglio nel paragrafo 2.2.2 nella pagina successiva) sono vuoti inoltre i pulsanti "Aggiungi al dizionario", "Correggi", "ignora" e "Correzione Automatica" sono disabilitati (dettaglio nel paragrafo 2.2.3 a pagina 14). È facile notare i campi descritti in modo generale nel paragrafo 2.1, ora andremo ad analizzarli tutti quanti nello specifico.

2.2.1 Liste

Tramite l'interfaccia grafica è possibile prendere visione e lavorare con due liste, la lista degli errori e la lista degli acronimi completa.

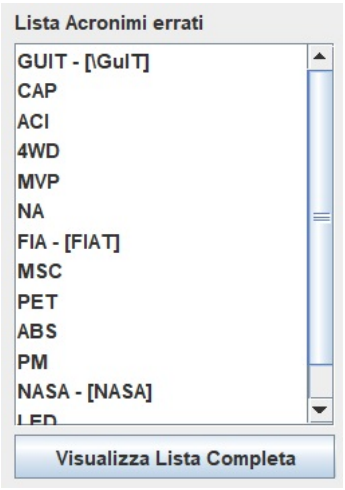

Figura 2.2: Lista degli errori

Lista degli errori è quella selezionata di default e racchiude tutti gli acronimi considerati errati dal tool, classificati in base all'acronimo stesso assieme al suo nome breve, se presente. È tramite questa lista che è possibile selezionare un acronimo semplicemente cliccandoci sopra con il mouse o spostandosi con le frecce direzionali. Nella Figura 2.2 è rappresentato un esempio dimostrativo di lista degli errori.

Un acronimo viene considerato errato, di conseguenza finisce in questa lista, se viene rilevata una discrepanza con il dizionario, quindi nel caso in cui il nome esteso sia stato inserito in modo scorretto, oppure se non viene rilevata alcuna corrispondenza, caso in cui l'acronimo non sia presente nel vocabolario.

Lista degli acronimi completa Sotto la lista degli errori, come si evince dalla Figura 2.2, si trova il pulsante per cambiare la visualizzazione dalla lista degli errori alla lista degli acronimi completa, una volta passata in visualizzazione la lista completa è possibile visionare tutti gli acronimi presenti nel documento, che questi siano corretti, incorretti oppure corretti dall'utente tramite il plugin. Può essere uno strumento utile per tenere traccia delle modifiche attuate. In ogni momento è possibile tornare a visualizzare la lista degli errori tramite lo stesso pulsante che ci ha permesso du visualizzare quella degli acronimi completa, che avrà cambiato nome in "Visualizza Acronimi errati".

2.2.2 Campi

I campi permettono di visualizzare informazioni sull'acronimo selezionato necessarie per fare il proofreading, in totale i campi utilizzati sono tre, nello specifico:

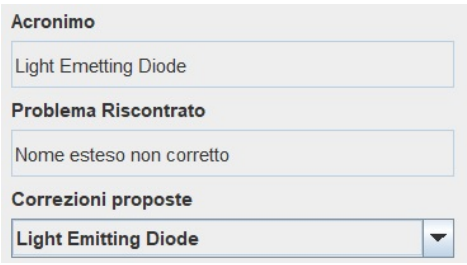

Figura 2.3: Campi informativi

Acronimo Il primo campo espone il nome esteso completo dell'acronimo scritto nel documento.

Problema Riscontrato Il secondo campo informa circa la correttezza dell'acronimo. In particolare informa se è corretto, se c'è una scorrettezza nel nome esteso, se non è stato trovato nel dizionario, se è stato ignorato precedentemente dall'utente oppure se è stato aggiunto nel dizionario manualmente.

Correzioni proposte Se il problema riscontrato è dovuto alla scorrettezza del nome esteso allora il terzo campo si attiva e propone una o più soluzioni per correggere l'errore, altrimenti rimane disabilitato. Questo campo è costituito da una lista drop-down, anche detta lista a cascata, e contiene appunto una lista di correzioni per correggere l'acronimo errato, di default è selezionato l'elemento ritenuto come correzione più plausibile dal plugin, ma l'utente può sempre visionare la lista completa e preferire la correzione che ritiene più adatta.

Nell'esempio mostrato in Figura 2.3 nella pagina precedente, è possibile capire come funzionano questi campi. In questo caso dalla lista degli errori è stato selezionato l'acronimo Light Emitting Diode (LED) tramite il codice LED, nel campo Acronimo è stampato il nome esteso non corretto dell'acronimo, il campo Problema Riscontrato notifica che il nome esteso non è corretto e la lista drop-down propone la correzione più adatta.

2.2.3 Pulsanti

I pulsanti permettono di interagire direttamente con il tool, in totale questi sono cinque, ma abbiamo già visto nel paragrafo 2.2.1 a pagina 12 il funzionamento del "Visualizza Lista Completa", concentriamoci ora sugli altri quattro. Questi pulsanti sono disponibili sia per la lista degli errori che per la lista degli acronimi completa.

Aggiungi al Dizionario pulsante che si attiva ogniqualvolta un acronimo viene considerato errato, permette l'omonima funzione di aggiungere l'acronimo nel dizionario e di toglierlo dalla lista degli errori, da questo momento in poi per tutti gli utilizzi successivi quest'ultimo verrà considerato corretto dal tool. Il plugin tiene traccia degli acronimi inseriti nel dizionario in questo modo e da la possibilità di ripensarci, infatti dalla lista degli acronimi completa è possibile selezionare l'acronimo aggiunto manualmente nel dizionario e con lo stesso pulsante con cui lo abbiamo aggiunto, che avrà cambiato nome in "Rimuovi dal dizionario", eliminarlo dal vocabolario. In questo modo l'acronimo torna nella lista degli errori pronto per essere corretto nuovamente. Sono presenti dei messaggi pop-up che informano l'esito dell'aggiunta/rimozione dal dizionario

Figura 2.4: Pulsanti attivi

Correggi questo pulsante permette di sovrascrivere nel documento in questione l'acronimo errato con la correzione scelta e di rimuoverlo quindi dalla lista degli errori. Un messaggio pop-up notifica la riuscita o meno dell'azione, specificando, in quest'ultimo

caso, il problema riscontrato. Si attiva solamente se è presente almeno una correzione per l'acronimo considerato, ovvero se quest'ultimo ha nome esteso non corretto.

Ignora ogni volta che un acronimo viene considerato errato questo pulsante diventa visibile e permette di ignorare l'errore, rimuovendo l'acronimo dalla lista degli errori. Nella lista degli acronimi completa un acronimi ignorato compare con l'etichetta di errore ignorato. Negli utilizzi successivi l'acronimo tornerà ad essere considerato errato.

Correzione Automatica diventa disponibile se è presente almeno un acronimo nella lista degli errori, consente di sovrascrivere tutti gli acronimi con nome esteso non corretto del documento con la correzione ritenuta più plausibile dal tool. Non va ad influire, invece, sugli acronimi non trovati nel dizionario, che saranno ancora presenti nella lista degli errori una volta terminata l'autocorrezione. Anche in questa procedura l'esito della correzione viene notificato tramite messaggio pop-up.

Nella Figura 2.4 nella pagina precedente è possibile prendere visione di come sono disposti questi pulsanti, in questo caso sono tutti attivi, questo perché è stato selezionato un acronimo con nome esteso non corretto.

Esempio funzionale nel capitolo 5 a pagina 47.

2.3 Dizionario

Un elemento di fondamentale importanza per il proofreading automatizzato è il dizionario, qui viene immagazzinato un elevato numero di acronimi che il tool consulta ogni volta che trova un acronimo nel documento, per verificarne così la correttezza.

```
WYSIWYG What You See Is What You Get
WYSIWYG Quel che vedi è ciò che ottieni
WYSIWYM What You See Is What You Mean
WYSIWYM Quel che vedi è ciò che intendi
WWF World Wildlife Fund
WWF Fondo mondiale per la natura
WWW World Wide Web
WWW Ragnatela mondiale
```

Figura 2.5: Sezione del dizionario

Come si evince dalla sezione presa in esempio nella Figura 2.5 gli acronimi inseriti nel dizionario devono avere un formato preciso, ossia: acronimo + spazio + nome esteso dell'acronimo. Non c'è distinzione tra maiuscole e minuscole, ma è preferibile scrivere l'acronimo in maiuscolo, le iniziali delle parole del nome esteso che compaiono nell'acronimo in maiuscolo e il resto in minuscolo. Per quasi ogni acronimo in lingua straniera è presente nel dizionario la versione tradotta in italiano, solo con la prima iniziale del nome esteso in maiuscolo, perché queste generalmente non corrispondo all'acronimo.

Il dizionario, come anticipato nei paragrafi precedenti, può essere manipolato dall'utente, al quale è consentito aggiungere altri acronimi oltre a quelli già presenti.

Capitolo 3

Composizione

3.1 Linguaggi di programmazione

Il plugin è stato realizzato totalmente in linguaggio Java, ma per comprenderlo è necessario conoscere anche le basi del linguaggio LaTeX.

3.1.1 Java

Java è un linguaggio di programmazione ad alto livello progettato per essere il più possibile indipendente dalla piattaforma di esecuzione. La forza di questo linguaggio è quella di basarsi sul paradigma della programmazione ad oggetti, il quale permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi. Questo tipo di programmazione è particolarmente adatto nei contesti in cui si possono definire delle relazioni di interdipendenza tra i concetti da modellare, un ambito che più di altri ne riesce a sfruttare i vantaggi è quello delle interfacce grafiche. Vediamo ora nel dettaglio particolari strutture utilizzate per la realizzazione del plugin:

Array

Un array, detto anche vettore o matrice a seconda della mono o bi dimensionalità, è una struttura dati statica. Si può definire come un contenitore che permette di gestire una sequenza di lunghezza fissa di elementi tutti del medesimo tipo. La lunghezza del array deve essere dichiarata al momento della sua allocazione e non può essere cambiata.
Metodi alla pagina Class Array di Oracle.

Liste

Una lista è una struttura di dati astratta che indica una collezione omogenea o container di dati. Ogni volta che si aggiunge un elemento o lo si rimuove dalla lista essa cambia di

dimensione, perciò è una struttura dinamica. Per accedere a un elemento in una qualsiasi posizione bisogna scandire sequenzialmente tutti gli elementi che lo precedono, tranne ovviamente per il primo elemento che gode di un accesso diretto.
Metodi alla pagina Interface List<E> di Oracle.

ArrayList

Un ArrayList è un'importante implementazione delle liste, anche chiamato vettore dinamico o ridimensionabile, permette di avere una struttura dati array con dimensione variabile. Gli array dinamici beneficiano di molti vantaggi degli array, inclusa una buona località di riferimento e l'utilizzo di cache dati, un basso utilizzo di memoria e accesso casuale, concedendo solo un piccolo sovraccarico addizionale (overhead) per memorizzare le informazioni su dimensione e capacità.
Costruttori, campi e metodi alla pagina Class ArrayList<E> di Oracle.

HashMap

Una Mappa rappresenta il tipo astratto che definisce una struttura dati in grado di memorizzare elementi nella forma di coppie chiave-valore. Ogni elemento all'interno di una mappa è identificato da una determinata chiave, essa consente non solo di recuperare ed inserire un elemento in una mappa, ma anche di definire un ordinamento per le implementazioni che prevedono questa funzionalità. È molto utilizzata nei metodi di ricerca nominati hashing, i quali cercano di accedere agli elementi nella tabella in modo diretto tramite operazioni aritmetiche che trasformano le chiavi in indirizzi della tabella.
Costruttori, campi e metodi alla pagina Class HashMap<K,V> di Oracle.

Regex

Le espressioni regolari (regular expression abbr. regexp o regex) sono una sequenza di simboli che identificano un insieme di stringhe. Un'espressione regolare specificata come stringa, per poter essere utilizzata in Java, deve prima venire compilata in un'istanza della classe Pattern. Il modello risultante può quindi essere utilizzato per creare un oggetto Matcher che può abbinare sequenze di caratteri arbitrarie all'espressione regolare.
Vediamo nel dettaglio alcuni costrutti delle espressioni regolari utilizzati nel progetto:

Classi di caratteri predefinite:

. Qualsiasi carattere
\d Carattere numerico: [0-9]
\D Carattere diverso da un numero: [^0-9]
\s Carattere spazio: [\t\n\x0B\f\r]

\S Carattere diverso dallo spazio: [^\s]
\w Carattere alfanumerico: [a-zA-Z_0-9]
\W Carattere costituito da simboli speciali: [^\w]
dove il simbolo ^ indica insieme negato.

Quantificatori:
X? X, zero o una volta
X* X, zero or più volte
X+ X, una o più volte
Xn X, esattamente n volte
Xn, X, almeno n volte
Xn,m X, almeno n volte, ma non più di m
con X una qualsiasi regex.
Per ulteriori informazioni riguardo le espressioni regolari e la classe Pattern visualizzare la guida Class Pattern di Oracle.

Swing

Swing è un framework per Java, ossia un'architettura logica di supporto, che mette a disposizione del programmatore una serie di classi e interfacce per la realizzazione di applicazioni grafiche (Grafic User Interface (GUI)). E' a tutti gli effetti una libreria e viene utilizzata appunto come libreria ufficiale per la realizzazione delle GUI in Java.

Per realizzare un'interfaccia grafica è necessario definire i vari componenti che la compongono, come ad esempio liste, campi di testo, bottoni ed esplicitare il comportamento delle interazioni eseguite sull'interfaccia (listener), per esempio la pressione di un bottone.

I componenti Swing gestiscono in modo autonomo la propria grafica e il proprio comportamento, anche se il rendering finale deve essere affidato al Sistema Operativo (SO), i componenti Swing sono responsabili della loro stessa renderizzazione grazie ai wrapper, che delegano il disegno ai widget nativi del SO.
Libreria in dettaglio alla sezione Package javax.swing di Oracle.

3.1.2 LaTeX

Nel paragrafo 1.2 a pagina 8 abbiamo già visto su cosa si basa il linguaggio LaTeX e le funzioni offerte per gestire gli acronimi, vediamo ora nello specifico come definirli.

Acronimi

Gli acronimi del documento vengono definiti mediante l'ambiente acronym con l'istruzione \acro{acronimo}[nome breve]{nome esteso}, dove: acronimo indica l'acronimo stesso

(es. GUIT), nome breve, opzionale, include solitamente dei comandi LaTeX richiesti dall'acronimo (es. \GuiT), nome esteso indica il nome per esteso dell'acronimo (es. Gruppo Utilizzatori Italiani di TeX e LaTeX). Nel caso dell'esempio proposto l'acronimo Gruppo Utilizzatori Italiani di TeX e LaTeX (GUIT) è stato introdotto con l'istruzione
\acro{GUIT} [\GuIT] {Gruppo Utilizzatori Italiani di \TeX e \LaTeX}.

Una volta definito l'acronimo, per scriverlo nel testo si usa il comando \ac{acronimo}, come già anticipato la prima volta che si inserisce un acronimo viene stampato il suo nome per esteso, seguito dall'acronimo fra parentesi, mentre le volte successive viene stampato solo l'acronimo.
Ulteriori informazioni su come utilizzare gli acronimi con il linguaggio LaTeX nella guida di Lorenzo Pantieri LaTeXpedia.

3.2 Ambienti

Per la realizzazione del progetto è stato richiesto un numero modesto di ambienti, nello specifico:

IntelliJ IDEA è un Ambiente di sviluppo integrato (IDE) specializzato per il linguaggio di programmazione Java. E' stato scelto per sviluppare il codice di questo plugin poiché oltre ad essere intuitivo, rapido e reattivo, supporta nativamente la gestione del framework swing.

TeXstudio editor per LaTeX multi piattaforma open-source, presenta un'interfaccia chiara e offre molte utili funzioni, ma va benissimo qualsiasi editor che consenta di lavorare con il linguaggio LaTeX e generare un file in formato .tex.

3.3 Scelta procedurale

Per realizzare questo progetto è stato scelto un processo incrementale, questo significa che dopo un'iniziale fase di specifica è stata svolta una fase di progettazione architetturale, seguita poi in modo graduale da progettazione di dettaglio, realizzazione e test incrementale dei componenti, partendo da quelli più critici e urgenti sino ad arrivare a quelli meno essenziali.

In modo più dettagliato si è realizzato inizialmente una versione senza interfaccia grafica, con delle funzionalità limitate, quali solo la ricerca degli acronimi e la verifica della loro correttezza, per poi aggiungere gradualmente con nuove versioni tutte le altre

funzionalità, partendo da quella di correggere l'errore, passando poi ad integrare un'interfaccia grafica dedicata, per poi finire includendo la possibilità di aggiungere/rimuovere l'acronimo nel dizionario, ignorare l'errore, suggerire automaticamente la correzione più adatta.

Capitolo 4

Funzionamento

Il funzionamento è descritto tramite la rete di Petri in Figura 4.1 nella pagina successiva, realizzata con un basso livello di astrazione e assumendo che tutte le transizioni siano realizzate in linguaggio Java, tutte sullo stesso server, compreso il blocco di Visualizzazione e Interazione.
Le risorse persistenti File.tex e Diz.txt si riferiscono rispettivamente al documento in formato .tex (in linguaggio LaTeX) e al dizionario.

In questo capitolo sono descritte tutte le transizioni e le risorse che compongono la rete di Petri, facendo riferimento alle classi utilizzate per realizzare le varie funzioni. Specificatamente le classi che vedremo sono: la classe Acronimo (4.1 a pagina 25), la classe Gestione Documento (4.2.1 a pagina 29), la classe Gestione Acronimi (4.3.1 a pagina 31), la classe Mappa Acronimi (4.4.1 a pagina 33), la classe Screen (4.6.1 a pagina 36) e la classe MainPlugIn.

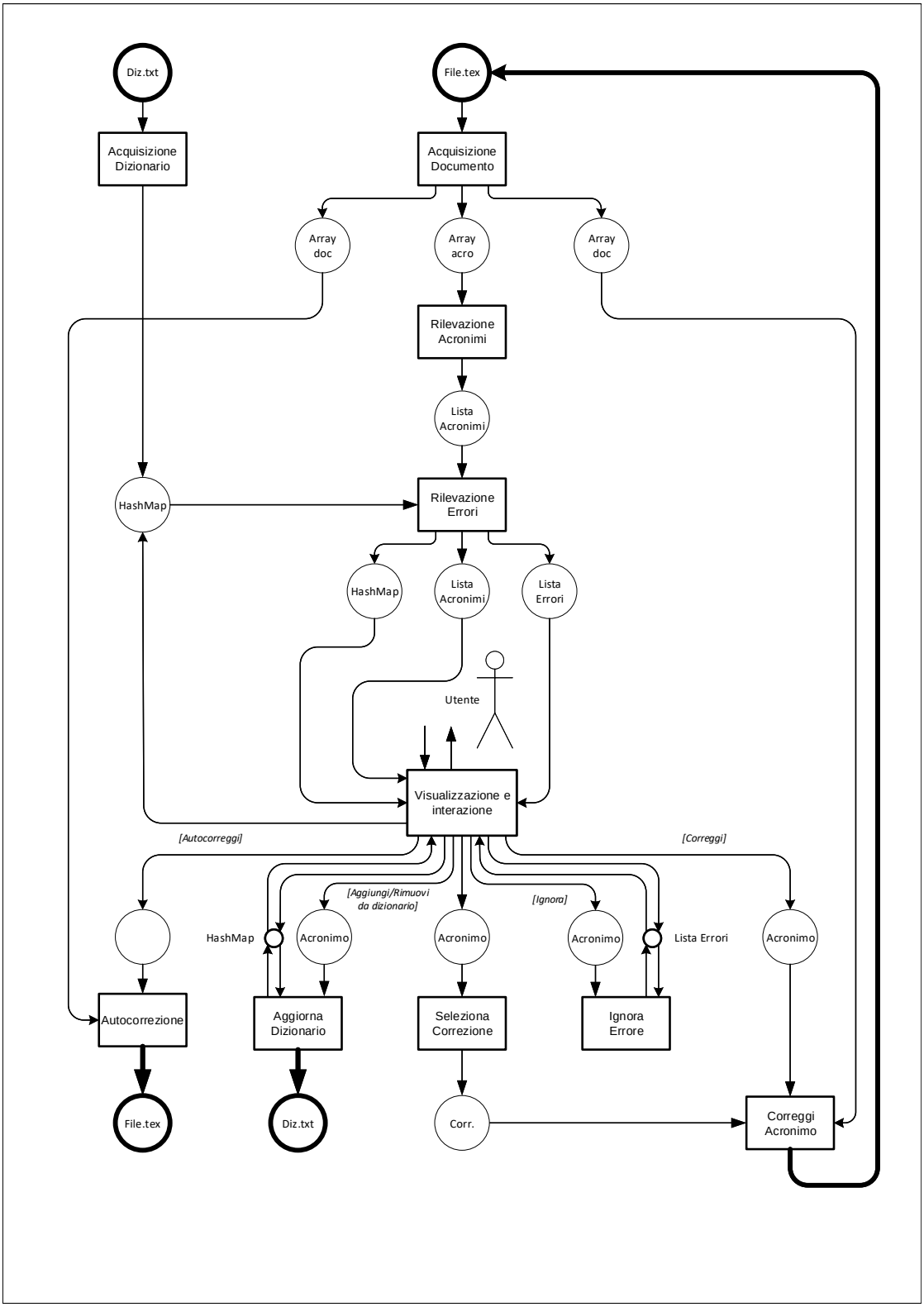

Figura 4.1: Rete di Petri

4.1 La Classe Acronimo

La classe Acronimo permette di creare un oggetto acronimo con le caratteristiche citate nel paragrafo 3.1.2 a pagina 19.
Le sezioni di codice presenti in questo capitolo appartengono al plugin e sono riportate con la numerazione originale che hanno all'interno della classe.

4.1.1 Attributi

```
25  private final String acronym;
26  private final String shortName;
27  private String extendedName;
28  private int code;
29  private List<String> correction;
```

Codice 4.1: Attributi della classe Acronimo

Come possiamo evincere dal codice 4.1 la classe Acronimo dispone di cinque attributi, precisamente:
acronym è una stringa contenente l'acronimo (es.GUIT);
shortName è una stringa contenete il nome breve (es. \GuIT), opzionale;
extendedName è una stringa contenete il nome esteso (es. Gruppo Utilizzatori Italiani di TEXe LATEX);
code è un numero intero che permette di capire la correttezza dell'acronimo, può assumere i seguenti valori 0=non valutato, 1=corretto, 2=non trovato, 3=nome esteso non corretto, 4=errore ignorato, 5=aggiunto manualmente al dizionario;
correction è una lista di stringhe contenente, se l'acronimo ha nome esteso non corretto, le correzioni proposte. (approfondimento sulle liste nel paragrafo 3.1.1 a pagina 17).

4.1.2 Costruttore

La classe acronimo dispone di tre costruttori (4.2 nella pagina seguente), in particolare quello di riferimento permette di inizializzare tutti gli attributi della classe, gli altri due facendo ovviamente riferimento al primo, consentono di creare l'oggetto Acronimo solo con l'acronimo, il nome esteso ed eventualmente il nome breve (altrimenti settato a null), impostando il codice di correttezza a zero (cioè non ancora valutato) e lasciando vuota la lista delle correzioni.

```
31  public Acronimo(String acro, String brev, String ests, int cod, List<
     String> corr){
32    this.acronym = acro;
33    this.shortName = brev;
34    this.extendedName = ests;
35    this.code = cod;
36    this.correction = corr;
37  }
38
39  public Acronimo(String acronym,String shortName,String extendedName){
40    this(acronym,shortName,extendedName,0,null);
41  }
42
43  public Acronimo(String acronym,String extendedName){
44    this(acronym,null,extendedName,0,null);
45  }
```

Codice 4.2: Costruttori della classe Acronimo

4.1.3 Metodi

Oltre ai consueti getter e setter di una classe, sono degni di nota i seguenti metodi:

L'ovverride del metodo **equals** che permette di considerare equivalenti due oggetti Acronimo solo se hanno lo stesso acronimo e lo stesso nome esteso, confrontati senza il carattere speciale per LaTeX "\", senza gli spazi prima e dopo la stringa e senza distinzione tra maiuscolo e minuscolo. Quest'ultima considerazione è richiesta ad esempio per considerare equivalenti gli acronimi Gruppo Utilizzatori Italiani di TeX e LaTeX (con le parole speciali inserite in questo modo \TeX e \LaTeX) e Gruppo Utilizzatori Italiani di Tex e Latex.

```
109 @Override
110 public boolean equals(Object o) {
111   if (this == o) return true;
112   if (o == null || getClass() != o.getClass()) return false;
113   Acronimo acronimo = (Acronimo) o;
114   return  acronym.equals(acronimo.acronym) &&
115   extendedName.trim().toLowerCase().replaceAll("\\\\", "").equals(
     acronimo.extendedName.trim().toLowerCase().replaceAll("\\\\", ""));
116 }
```

Codice 4.3: Override metodo equals della classe Acronimo

Override del metodo **toString** ottimizzandolo per restituire una stringa più chiara con le informazioni necessarie, distinguendo il caso dell'acronimo con o senza nome breve e differenziando il risultato per i vari tipi di codice.

getOptimalCorrection consente di selezionare automaticamente la correzione ottimale per l'errore in questione. Si basa sulla distanza di Levenshtein [1], la è quale nata per determinare l'analogia tra due stringhe (ad esempio A e B), è costituita dal numero minimo di modifiche elementari che consentono di trasformare la stringa A nella stringa B. È considerata modifica la cancellazione di un carattere, la sostituzione di un carattere con un altro oppure l'inserimento di un carattere. Ad esempio per trasformare "Italy" in "Italia" sono necessarie 2 modifiche, ovvero la sostituzione della 'y' con la 'i' (Italy → Itali) e l'aggiunta della 'a' (Italy → Italia). Nel codice 4.4 nella pagina successiva è mostrato l'algoritmo usato per calcolare questa distanza tramite l'uso di una matrice (la funzione levenshteinDistance(String a,String b) che restituisce un numero intero). Per lo scopo del progetto è necessario determinare la correzione ottimale tra una lista di stringhe, per cui viene calcolata la distanza di Levenshtein per ogni elemento della lista e poi selezionato l'elemento con una distanza minore (appunto la funzione getOptimalCorrection() che restituisce una stringa).

[1]Distanza di Levenshtein da Wikipedia, l'enciclopedia libera.
URL: https://it.wikipedia.org/wiki/Distanza_di_Levenshtein

```
150   public String getOptimalCorrection(){
151     if(correction.isEmpty())    return null;
152     else if (correction.size()==1)  return  correction.get(0);
153     else{
154       int j=0;    // j = posizione correzione ottimale
155       int k=levenshteinDistance(extendedName,correction.get(0));
                    // k = numero di modifiche minore
156       for(int i=1;i<correction.size();i++){
                    // i = posizione corrente
157         int h=levenshteinDistance(extendedName,correction.get(i));
                    // h = numero di modifiche posizone i-esima
158         if(h<k) { // correzione i-esima ha una correzione migliore
159           k = h;  // aggiorno numero di modifiche
160           j = i;  // salvo posizione migliore
161         }
162       }
163       return correction.get(j);
164     }
165   }
166
167   private int levenshteinDistance(String a,String b){
168     int i, j;
169     final int n = a.length(), m = b.length();
170     int D[][] = new int[n+1][m+1];
171     for (i=0; i<n+1; i++) {
172       for (j=0; j<m+1; j++) {
173         if (i==0 || j==0)
174           D[i][j] = maximum(i,j);
175         else
176           D[i][j] = minimum(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] +
    (a.charAt(i-1) != b.charAt(j-1) ? 1 : 0) );
177       }
178     }
179     return D[n][m];
180   }
181
182   private int maximum(int i, int j) {
183     if(i>j) return i;
184     else return j;
185   }
186
187   private int minimum(int i, int j, int k) {
188     int result = i;
189     if (j < result) result = j;
190     if (k < result) result = k;
191     return result;
192   }
```

Codice 4.4: Metodo per la correzione ottimale della classe Acronimo

4.2 Acquisizione Documento

L'acquisizione del documento in formato .tex è svolta totalmente dalla classe Gestione Documento.

4.2.1 La classe Gestione Documento

```
20  private final ArrayList<String> array;
21  private final ArrayList<String> arraymk2;
22  private String file;
23  public GestioneDocumento() {
24    this.array = new ArrayList<>();
25    this.arraymk2 = new ArrayList<>();
26  }
```

Codice 4.5: Attributi e costruttore della classe Gestione Documento

Gli attributi della classe, come si deduce dal codice 4.5, sono due ArrayList (approfondimento paragrafo 3.1.1 a pagina 18) di stringhe e un campo per includere la stringa contenente il percorso dove è presente il documento (**file**). Il costruttore permette semplicemente di istanziare i due array appena citati. Il primo (**array**) ha il compito di contenere e conservare il documento riga per riga, mentre il secondo (**arraymk2**) copia in primo luogo il primo, per poi essere modificato senza sovrascrivere e quindi perdere informazioni. Questa operazione è svolta dal metodo **readFile** (codice 4.6), che legge per mezzo del Buffer il documento e inserisce ogni sua riga in una posizione del array.

```
29  public void readFile(String file) throws IOException {
30    this.file=file;
31    FileReader fr = new FileReader(file);
32    BufferedReader bf = new BufferedReader(fr);
33    String line = null;
34    while((line=bf.readLine())!=null)
35    array.add(line);
36    bf.close();
37    arraymk2.addAll(array);
38  }
```

Codice 4.6: Metodo readFile della classe Gestione Documento

In seguito alla lettura del file è mandato in esecuzione il metodo **deleteComment** che rimuove dal array tutti i commenti, così da non andare a controllare acronimi non necessari per il documento ed evitare comportamenti indesiderati. Un commento in LaTeX è iniziato dal carattere speciale % e continua su tutta la riga da dopo la sua presenza, bisogna fare attenzione a non considerare come commento il carattere percentuale preceduto dal escape backslash (\%) che invece non commenta. Un'altra possibilità per commentare in LaTeX è

rappresentata dell'ambiente comment, inizializzato con l'espressione \begin{comment} e terminato da \end{comment}, tutto ciò che è compreso tra queste espressioni è commento e viene quindi rimosso dal array da questo metodo.

L'ultima operazione di questa transizione è svolta dal metodo **takeAcronymBlock**, il quale lascia nel array solo la parte dedicata alla dichiarazione degli acronimi, rimuovendo tutto il resto considerato non necessario. Il codice di questo metodo (4.7) utilizza le espressioni regolari (approfondimento paragrafo 3.1.1 a pagina 18) e inizialmente rimuove ogni riga fino a trovare l'ambiente per definire gli acronimi, lascia poi invariate tutte le altre fino a quando non individua l'istruzione per terminare il blocco acronimi, torna così a rimuovere righe in maniera ricorsiva, fino a che non incontra un altro blocco acronimi o finché non arriva in fondo al documento.

```
90  public void takeAcronymBlock() throws DocumentoNonValidoException {
91    if(arraymk2.size()==0)
92      throw new DocumentoNonValidoException("Documento vuoto");
93    Pattern patternBeginA = Pattern.compile("(.*)(\\\\begin\\s*\\{
      acronym})(.*)");  // creo il pattern con 3 gruppi --> gruppo1:(
      eventualmente qualsiasi carattere) gruppo2:(istruzione \begin{
      acronym} con una eventuale spaziatura indefinita tra \begin e {
      acronym}) gruppo3:(eventualmente qualsiasi carattere)
94    Pattern patternEndA = Pattern.compile("(.*)(\\\\end\\s*\\{acronym})
      (.*)");           // creo il pattern con 3 gruppi --> gruppo1:(
      eventualmente qualsiasi carattere) gruppo2:(istruzione \end{acronym
      } con una eventuale spaziatura indefinita tra \end e {acronym})
      gruppo3:(eventualmente qualsiasi carattere)
95    int i=0; // posizione nel array
96    while(arraymk2.get(i)!=null) {
97      Matcher matcherBeginA = patternBeginA.matcher(arraymk2.get(i));
98      while(!matcherBeginA.matches()){ //non presente \begin{acronym}
99        arraymk2.remove(i);
100       if(arraymk2.size()!=i)
101         matcherBeginA = patternBeginA.matcher(arraymk2.get(i));
102       else return;
103     }
104     Matcher matcherEndA = patternEndA.matcher(arraymk2.get(i));
105     while(!matcherEndA.find()) {  //non presente \end{acronym}
106       i++;
107       if(arraymk2.size()==i)
108         throw new DocumentoNonValidoException("Manca l'istruzione
          /end{acronym}");
109       matcherEndA = patternEndA.matcher(arraymk2.get(i));
110     }
111     i++;
112     if(i== arraymk2.size()) return;
113   }
114 }
```

Codice 4.7: Metodo takeAcronymBlock della classe Gestione Documento

4.3 Rilevazione Acronimi

L'obiettivo di questa transizione è quello di creare una lista contente tutti gli acronimi definiti nel documento, ogni acronimo deve costituire un oggetto acronimo. Questo compito spetta totalmente alla classe Gestione Acronimi.

4.3.1 La classe Gestione Acronimi

```
30  private final ArrayList<Acronimo> acronymList;
31  private final ArrayList<Acronimo> errorList;
32  private final ArrayList<Acronimo> ignoreList;
33  private final ArrayList<Acronimo> dictionaryList;
34
35  public GestioneAcronimi(){
36    this.acronymList = new ArrayList<>();
37    this.errorList = new ArrayList<>();
38    this.ignoreList = new ArrayList<>();
39    this.dictionaryList = new ArrayList<>();
40  }
```

Codice 4.8: Attributi e costruttore della classe Gestione Acronimi

La classe Gestione Acronimi presenta quattro ArrayList istanziati inizialmente senza oggetti dal costruttore della classe:
acronymList è una lista completa di tutti gli oggetti acronimo (4.1 a pagina 25);
errorList è lista di oggetti acronimo ritenuti errati;
ignoreList è una lista di oggetti acronimo ignorati dall'utente;
dictionaryList è una lista di oggetti acronimo aggiunti manualmente nel dizionario.

Il metodo incaricato di svolgere l'interazione Rilevazione Acronimi è **findAcronym**, il quale riceve dalla classe Gestione Documento l'array composto dalle sole dichiarazioni di acronimi. Questo metodo si serve delle espressioni regolari (3.1.1) così da estrapolare dal array i componenti necessari per formare l'oggetto acronimo, più precisamente un primo MatcherAcro si serve del PatternAcro, per ricavare dal gruppo 2 l'acronimo, successivamente esamina il gruppo 4, in prima battuta prova a trovare una corrispondenza con il patternShort, nel caso in cui l'acronimo presenti il nome breve, se la trova preleva il nome breve dal gruppo 2 e quello esteso dal gruppo 4, altrimenti cerca riscontro nel patternExt e trae il nome esteso dal gruppo 2. A questo punto crea l'oggetto acronimo e si serve del metodo isInList(Acronimo a, ArrayList<Acronimo> list) per verificare che questo non sia già presente nella lista, fornendogli l'oggetto acronimo appena creato e la lista degli acronimi, in caso di esito negativo aggiunge l'oggetto in lista (acronymList) e passa alla dichiarazione successiva.

```
43  public void findAcronym(ArrayList<String> arrayAcro) throws
     AcronimoNonValidoException {
44     Pattern patternAcro = Pattern.compile("(.*\\\\acro\\s*\\{)(\\w*)(})
    (.*)");    // creo il pattern con 4 gruppi --> gruppo1:(
    eventualmente qualsiasi carattere, istruzione \acro qualsiasi
    eventuale spaziatura e parentesi {) gruppo2:(qualsiasi carattere
    alfanumerico) gruppo3:(parentesi }) gruppo4:(qualsiasi carattere)
45     Pattern patternShort = Pattern.compile("(\\s*\\[)(.+)(]\\s*\\{)(.+)
    (}.*)");   // creo il pattern con 5 gruppi --> gruppo1:(ev.
    qualsiasi spaziatura e parentesi [) gruppo2:(qualsiasi carattere)
    gruppo3:(parentesi ] eventualmente qualsiasi spaziatura e parentesi
     {) gruppo4:(qualsiasi carattere) gruppo5:(parentesi })
46     Pattern patternExt = Pattern.compile("(\\s*\\{)(.+)(}.*)");
    // creo il pattern con 3 gruppi --> gruppo1:(eventualmente
    qualsiasi spaziatura e parentesi {) gruppo2:(qualsiasi carattere)
    gruppo3:(parentesi })
47     for(String s:arrayAcro) {
48       Matcher matcherAcro = patternAcro.matcher(s);
49       if(matcherAcro.matches()) { // match con l'istruzione \acro
50         String acronym=matcherAcro.group(2);
51         Matcher matcherShort = patternShort.matcher(matcherAcro.group
    (4));
52         Matcher matcherExt = patternExt.matcher(matcherAcro.group(4));
53         if(matcherShort.matches()) { // caso [nome breve]{nome esteso}
54           String shortName=matcherShort.group(2).trim();
55           String extendedName=matcherShort.group(4).trim();
56           Acronimo acronymObject = new Acronimo(acronym,shortName,
    extendedName);
57           if(!isInList(acronymObject,acronymList)) { // non in lista
58             acronymList.add(acronymObject);
59           }
60         }
61         else if (matcherExt.find()) { // caso solo {nome esteso}
62           String extendedName=matcherExt.group(2).trim();
63           Acronimo acronymObject = new Acronimo(acronym,extendedName);
64           if(!isInList(acronymObject,acronymList)) { // non in lista
65             acronymList.add(acronymObject);
66           }
67         }
68         else
69           throw new AcronimoNonValidoException("non e' stato dichiarato
     il nome esteso dell'acronimo: " + acronym);
70       }
71     }
72  }
```

Codice 4.9: Metodo findAcronym della classe Gestione Acronimi

4.4 Acquisizione Dizionario

Questa transizione si occupa di acquisire gli acronimi da un dizionario e di salvarli in modo efficace in una struttura dati. L'incarico appena citato è totalmente condotto dalla classe Mappa Acronimi.

4.4.1 La classe Mappa Acronimi

```
22  private final Map<String, List<Acronimo>> map;
23  private final String DictionaryFile;
```

Codice 4.10: Attributi della classe Mappa Acronimi

Nel codice 4.10 appena citato sono presentati gli unici due attributi della classe:
Map<String, List<Acronimo> > map HashMap con chiave il codice dell'acronimo e come valore una lista di oggetti acronimo (approfondimento HashMap 3.1.1 a pagina 18)
DictionaryFile percorso e nome del file del dizionario.

```
25  public MappaAcronimi() throws IOException {
26    this.DictionaryFile = MainPlugIn.getFileDizionario();
27    this.map = new HashMap<>();
28    FileReader fr = new FileReader(DictionaryFile);
29    BufferedReader bf = new BufferedReader(fr);
30    String line = null;
31    Pattern pattern = Pattern.compile("(\\w+)(\\s+)(.+)");
      //creo il pattern con 3 gruppi --> gruppo1:(caratteri alfanumerici)
       gruppo2:(spaziatura) gruppo3:(qualsiasi carattere)
32    Matcher matcher;
33    while((line=bf.readLine())!=null) {
34      matcher = pattern.matcher(line);
35      if(matcher.matches()) {
36        String acronymKey = matcher.group(1);
37        Acronimo acronimo = new Acronimo(acronymKey,matcher.group(3));
38        if(!map.containsKey(acronymKey)) {  // chiave vuota
39          List<Acronimo> list = new ArrayList<>();
40          list.add(acronimo);
41          map.put(acronymKey, list);
42        }
43        else {  // chiave gia' presente
44          List<Acronimo> list = map.get(acronymKey);
45          list.add(acronimo);
46          map.put(acronymKey,list);
47        }
48      }
49    }bf.close();
50  }
```

Codice 4.11: Costruttore della classe Mappa Acronimi

Il costruttore di questa classe è molto importante, infatti oltre ad istanziare la mappa svolge la funzione di acquisizione del dizionario (4.11 nella pagina precedente). Di seguito è mostrato il procedimento: tramite il Buffer è possibile leggere il file dizionario riga per riga, per ogni riga tramite le regex (3.1.1) viene estratto l'acronimo (gruppo 1) e il suo nome esteso (gruppo 3), viene così creato un oggetto acronimo senza il nome breve (non presente nel dizionario). Successivamente si passa ad aggiungere questo oggetto nella mappa, considerando come chiave di quest'ultima l'acronimo stesso, per prima cosa si controlla se non è presente la chiave in questione nella mappa, in questo caso si crea una lista nuova di acronimi con solo l'oggetto appena creato e la si inserisce nella mappa come valore per la chiave selezionata; altrimenti se la chiave è già presente nella mappa si preleva la lista corrispondente, si aggiunge l'oggetto acronimo e si rimette la lista nella mappa con ovviamente lo stesso valore della chiave. Per richiedere la mappa è disponibile il rispettivo metodo getMap che ritorna il riferimento alla HashMap.

4.5 Rilevazione Errori

Questa mansione è incaricata alla classe Gestione Acronimi (4.3.1 a pagina 31), richiede da se stessa la lista degli acronimi e dalla Mappa Acronimi (4.4.1 nella pagina precedente) la HashMap (dizionario degli acronimi).

```
74  public void acronymCheck(Map<String, List<Acronimo>> map) {
75    for(Acronimo a: acronymList) {
76      if(isInList(a,ignoreList)) // acronimo ignorato
77        a.setCode(4);            // codice 4 = acronimo errato ignorato
78      else if(isInList(a,dictionaryList))    // aggiunto nel dizionario
79        a.setCode(5); // codice 5 = aggiunto manualmente nel dizionario
80      else{       // acronimo non ignorato e non aggiunto al dizionario
81        acronymAndListSet(map,a);
82      }
83    }
84  }
```

Codice 4.12: Metodo acronymCheck della classe Gestione Acronimi

La funzione principale è compiuta dal metodo acronymAndListSet, ma esso viene diretto dal metodo **acronymCheck** (codice 4.12), il quale, sfogliando la lista degli acronimi, richiama il primo solo se l'acronimo non è presente nella lista degli acronimi ignorati o nella lista degli acronimi aggiunti manualmente nel dizionario, in modo da non aggiungere questi nelle lista degli errori.

Il metodo **acronymAndListSet** è adibito a verificare la correttezza dell'acronimo, riceve dal metodo acroymCheck la mappa degli acronimi (dizionario) e l'acronimo in questione, setta il giusto codice a quest'ultimo e in caso di rilevato errore, lo aggiunge

nella lista degli errori. Nello specifico cerca l'acronimo stesso nel dizionario, se non lo trova gli assegna il codice 2 (acronimo non presente nel dizionario), successivamente controlla la corrispondenza del nome esteso, in caso positivo gli assegna il codice 1 (acronimo corretto), altrimenti salva in una lista tutti i nomi estesi che hanno lo stesso codice di acronimo e la aggiunge nel campo correzioni dell'acronimo, settandogli a 3 il codice (nome esteso non corretto). Il metodo aggiunge l'acronimo nella lista degli errori quando il codice di errore è 2 e 3.

```
266  private void acronymAndListSet(Map<String, List<Acronimo>> map,
      Acronimo a){
267    String key = a.getAcronym();
268    if (!map.containsKey(key)) {  // non presente nel dizionario
269      a.setCode(2); // codice 2 = acronimo non presente nel dizionario
270      errorList.add(a);
271    } else {        // chiave acronimo presente nel dizionario
272      List<Acronimo> list = map.get(key);
273      boolean findIt = false;
274      List<String> corrections = new ArrayList<>(); // lista correzioni
275      for (Acronimo acro : list) {
276        if (a.equals(acro)) {
277          findIt = true;
278          break;
279        }
280        corrections.add(acro.getExtendedName());  // tutti i nomi
     estesi degli acronimi con la stessa chiave
281      }
282      if (findIt)     // acronimo presente nel dizionario
283      a.setCode(1);   // codice 1 = corretto
284      else {          // chiave presente ma non nome esteso
285        a.setCode(3); // codice 3 = nome esteso non corretto
286        a.setCorrection(corrections); // lista  correzioni
287        errorList.add(a);
288      }
289    }
290  }
```

Codice 4.13: Metodo acronymAndListSet della classe Gestione Acronimi

4.6 Visualizza e Interagisci

E' la transizione più complessa, si occupa di creare l'interfaccia grafica e di permettere all'utente di interagirci. In base alle scelte di quest'ultimo richiama altre transizioni in altrettante classi e gestisce il comportamento complessivo. E' implementata dalla classe Screen e richiede la lista degli acronimi completa, la lista degli errori e la HashMap.

4.6.1 La classe Screen

Classe creata con lo Swing UI Designer di IntelliJ (3.2 a pagina 20), il quale consente di sviluppare la forma dell'interfaccia grafica tramite, in questo caso, la classe Screen e modificando automaticamente il file .form abbinato che gestisce la forma finale della GUI (Approfondimento framework swing paragrafo 3.1.1 a pagina 19).

Per prima cosa la classe Screen estende la classe JFrame, così da poter generare componenti personalizzati, poi passa alla definizione di tutti questi vari componenti che compongono l'interfaccia, creando la struttura vista in figura 2.1 a pagina 12, (nel codice 4.14).

```
33 public class Screen extends JFrame {
34   private JPanel pannelMain;
35   private JList acronymList;
36   private JTextField textAcronym;
37   private JTextField textProblem;
38   private JButton aggiungiAlDizionarioButton;
39   private JLabel labelErrorList;
40   private JComboBox correctionsComboBox;
41   private JLabel labelAcronimo;
42   private JLabel labelProblem;
43   private JLabel labelCorrection;
44   private JButton automaticCorrectionButton;
45   private JButton removeErrorButton;
46   private JButton completeListButton;
47   private JButton correctionButton;
48   private JScrollPane listScrollPane;
49   private final DefaultListModel ListAcronymModel;
50   private final DefaultComboBoxModel CorrectionsModel;
51   private final ArrayList<Acronimo> acronimi;
52   private final List<String> correzioni;
53   private final GestioneAcronimi gestioneAcronimi;
54   private final GestioneDocumento gestioneDocumento;
55   private final MappaAcronimi mappaAcronimi;
56   private final String textErrorList = "Lista Acronimi errati";
57   private final String textErrorListButton = "Visualizza Acronimi
       errati";
58   private final String textAcronymList = "Lista completa Acronimi";
59   private final String textAcronymListButton = "Visualizza Lista
       Completa";
```

Codice 4.14: Attributi della classe Screen

Successivamente nel costruttore, oltre a creare il frame, vengono definiti i vari listener, ovvero il comportamento delle interazioni eseguite sull'interfaccia. Ogni listener, pubblico nel costruttore, quando viene chiamato in causa richiama un proprio metodo privato all'interno della classe. Per istanziare un oggetto Screen è necessario fornire gli oggetti delle classi Gestione Acronimi, Gestione Documento e Mappa Acronimi. E' presentato di seguito il codice che costituisce il costruttore (4.15 a fronte).

```
61  public Screen(GestioneAcronimi gestioneAcronimi, GestioneDocumento
      gestioneDocumento, MappaAcronimi mappaAcronimi) {
62    super("LaTeX Acronyms Proofreading");
63    this.setContentPane(this.pannelMain);
64    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
65    this.pack();
66    this.gestioneAcronimi = gestioneAcronimi;
67    this.gestioneDocumento = gestioneDocumento;
68    this.mappaAcronimi = mappaAcronimi;
69    aggiungiAlDizionarioButton.setEnabled(false);
70    removeErrorButton.setEnabled(false);
71    correctionButton.setEnabled(false);
72    correctionsComboBox.setEnabled(false);
73    acronimi = new ArrayList<>();
74    ListAcronymModel=new DefaultListModel();
75    acronymList.setModel(ListAcronymModel);
76    correzioni = new ArrayList<>();
77    CorrectionsModel=new DefaultComboBoxModel();
78    correctionsComboBox.setModel(CorrectionsModel);
79
80    acronymList.addListSelectionListener(new ListSelectionListener() {
81      @Override
82      public void valueChanged(ListSelectionEvent e) {
83        ListAcronymSelection(e);
84    }});
85    aggiungiAlDizionarioButton.addActionListener(new ActionListener() {
86      @Override
87      public void actionPerformed(ActionEvent e) {
88        ButtonDizionarioClick(e);
89    }});
90    automaticCorrectionButton.addActionListener(new ActionListener() {
91      @Override
92      public void actionPerformed(ActionEvent e) {
93        ButtonAutomaticCorrection(e);
94    }});
95    removeErrorButton.addActionListener(new ActionListener() {
96      @Override
97      public void actionPerformed(ActionEvent e) {
98        deleteAcronymError(e);
99    }});
100   completeListButton.addActionListener(new ActionListener() {
101     @Override
102     public void actionPerformed(ActionEvent e) {
103       if(completeListButton.getText().equals(textAcronymListButton))
104         changeList(e,gestioneAcronimi.getAcronymList(),
    textAcronymList, textErrorListButton);
105       else
106         changeList(e,gestioneAcronimi.getErrorList(),textErrorList,
    textAcronymListButton);
107   }});
108   correctionButton.addActionListener(new ActionListener() {
109     @Override
110     public void actionPerformed(ActionEvent e) {
111       singleCorrection(e);
112   }});
113 }
```

Codice 4.15: Costruttore della classe Screen

Per la prima parte della transizione che stiamo prendendo in esame, precisamente Visualizza, è essenziale il listener della lista, ovvero ListSelectionListener() che richiama il metodo **ListAcronymSelection**, tramite quest'ultimo è possibile mostrare all'utente la lista degli acronimi errati (2.2 a pagina 13). Per ogni acronimo che l'utente seleziona dalla lista, l'interfaccia ne mostra le informazioni e permette di intraprendere delle azioni, abilitando o disabilitando i corretti pulsanti. Il codice di questa classe è molto lungo e ripetitivo, suscita interesse però la parte dedicata alla lista: il metodo refreshAcronymList (codice 4.16) permette di aggiornare la lista visualizzata, mostrando come etichetta dell'oggetto acronimo l'acronimo stesso con in aggiunta il nome breve, se presente.

```
330  private void refreshAcronymList(){
331    ListAcronymModel.removeAllElements();
332    for(Acronimo a : acronimi) {
333      if(a.getShortName()!=null)      // se l'acronimo ha il nome breve
334        ListAcronymModel.addElement(a.getAcronym() + " - [" + a.
     getShortName() + "]");
335      else                            // solo nome esteso
336        ListAcronymModel.addElement(a.getAcronym());
337    }
338  }
```

Codice 4.16: Metodo refreshAcronymList della classe Screen

È possibile ottenere la posizione nella lista dell'elemento selezionato dall'utente grazie all'istruzione:

```
122  int acronymNumber = acronymList.getSelectedIndex();
```

Codice 4.17: Istruzione getSelectedIndex() del metodo ListAcronymSelection

in questo modo il metodo ListAcronymSelection può attuare azioni personalizzate per ogni oggetto selezionato. In particolare, dopo aver disposto nei campi di visualizzazione le informazioni di nome esteso e problema riscontrato, in base al codice dell'acronimo in questione abilita i pulsanti necessari, così da attivare solo il pulsante "Aggiungi al Dizionario" se l'acronimo non è presente nel dizionario, mentre attivare questo, "Correggi" e "Ignora" se l'acronimo ha nome esteso non corretto. In quest'ultimo caso il metodo popola la lista drop-down con le correzioni proposte e seleziona, grazie al metodo della classe acronimo getOptimalCorrection (4.1.3 a pagina 27), quella ottimale (si tratta della transizione **Seleziona Correzione**).

4.7 Correggi Acronimo

Questa transizione scatta dopo che l'utente seleziona una correzione (o lascia quella selezionata di default) e preme il pulsante "Correggi". L'implementazione spetta alla classe Screen, che richiama metodi della classe Gestione Acronimi e Gestione Documento. Partiamo dalla classe Screen, il metodo in questione è chiamato **singleCorrection**.

```
234   private void singleCorrection(ActionEvent e){
235     int acronymNumber = acronymList.getSelectedIndex();
236     if(acronymNumber >=0) {
237       Acronimo a = acronimi.get(acronymNumber);
238       Component frame = null;
239       try {
240         gestioneAcronimi.correctAcronym(gestioneDocumento.getArray(), a
      ,correctionsComboBox.getSelectedIndex());
241         JOptionPane.showMessageDialog(frame, "Acronimo " + a.getAcronym
      () + " Corretto in\n" + correctionsComboBox.getItemAt(
      correctionsComboBox.getSelectedIndex()) );
242         clearField();                    // ripristino i campi
243         gestioneDocumento.overwrite();   // riscrivo il documento
244         update();                        // aggiorno le liste
245       } catch (AcronimoNonValidoException | IOException
      acronimoNonValidoException) {
246         acronimoNonValidoException.printStackTrace();
247         JOptionPane.showMessageDialog(frame, "Errore durante la
      correzione\n" +
248         acronimoNonValidoException.getMessage() ,"Errore",JOptionPane.
      ERROR_MESSAGE);
249       }
250     }
251   }
```

Codice 4.18: Metodo singleCorrection della classe Screen

Il metodo gestisce le eccezioni e richiama in primo luogo il metodo della calasse Gestione Acronimi correctAcronym, per correggere l'acronimo e in secondo luogo il metodo overwrite della classe Gestione Documento, per riscrivere il documento con l'acronimo corretto. Oltre a ciò mostra con un messaggio pop-up l'esito dell'operazione.

Per quanto riguarda il metodo **correctAcronym**, richiede l'array contenente il documento originale, l'acronimo da correggere e la posizione della correzione scelta dall'utente. Scorre tutto l'array fino a trovare l'acronimo da correggere e una volta trovato lo sostituisce con quello corretto. Per fare ciò crea un Pattern con i campi dell'acronimo in questione utilizzando le espressioni regolari (3.1.1 a pagina 18) e cerca la corrispondenza in ogni riga del array. Quando la trova crea una riga con lo stesso formato, sostituisce il nome esteso con la correzione e la rimpiazza. Nel codice 4.19 nella pagina successiva è mostrata solo la parte relativa alla correzione dell'acronimo con nome breve, quella senza è analoga.

```
164  public void correctAcronym(ArrayList<String> array, Acronimo
     acroToBeCorrected,int c) throws AcronimoNonValidoException {
165   if (acroToBeCorrected.getCode() != 3)
166     throw new AcronimoNonValidoException("L'acronimo non ha
     correzione");
167   String acroToBeCorrectedAcro = acroToBeCorrected.getAcronym();
168   String acroToBeCorrectedShortName = acroToBeCorrected.getShortName
     ();
169   String acroToBeCorrectedExtendedName = acroToBeCorrected.
     getExtendedName();
170   String acroToBeCorrectedCorrection = acroToBeCorrected.
     getCorrection().get(c);
171   Pattern patternAcro = Pattern.compile("(.*)(\\\\acro\\s*\\{)(.*)");
     // creo il pattern con 3 gruppi --> gruppo1:(eventualmente
     qualsiasi carattere) gruppo2:(istruzione \acro qualsiasi eventuale
     spaziatura e parentesi {) gruppo3:(qualsiasi eventuale carattere)
172   if (acroToBeCorrectedShortName != null) {
173     Pattern patternAcroShort = Pattern.compile("(\\s*" +
     acroToBeCorrectedAcro + "\\s*)(}\\s*\\[)(\\s*" +
     acroToBeCorrectedShortName.replaceAll("\\\\", "").trim() + "\\s*)
     (]\\s*\\{)(\\s*" + acroToBeCorrectedExtendedName.replaceAll("\\\\",
      "").trim() + "\\s*)(})(.*)");  // creo il pattern con 7 gruppi -->
      gruppo1:( acronimo ), gruppo2:(parentesi { qualsiasi eventuale
     spaziatura e parentesi [ ), gruppo3:(nome breve), gruppo4:(
     parentesi ] qualsiasi ev. spaziatura e parentesi { ), gruppo5:(
     nome esteso ), gruppo6:(parentesi } ), gruppo7:(qls ev. carattere)
174       for (int i = 0; i < array.size(); i++) {
175         String s = array.get(i);
176         Matcher matcherAcro = patternAcro.matcher(s);
177           if (matcherAcro.matches()){
178             Matcher matcherAcroShort = patternAcroShort.matcher(
     matcherAcro.group(3).replaceAll("\\\\", ""));
179             if (matcherAcroShort.matches()) { // con il nome breve
180             String backslash = ""; // Stringa che mi permette di
     aggiungere il carattere speciale LaTeX "\" senza causare problemi
181             if(acroToBeCorrectedShortName.startsWith("\\"))
     // nel nome breve "\" e' posizionato all'inizio del nome
182               backslash="\\\\";
183             String replacement = matcherAcro.group(1) + "\\\\acro{" +
      acroToBeCorrectedAcro + "} [" + backslash +
     acroToBeCorrectedShortName + "] {" + acroToBeCorrectedCorrection +
     "}" + matcherAcroShort.group(7); //correggo l'acronimo, lascio
     invariato cio' che c'era prima e dopo di esso
184             StringBuilder buffer = new StringBuilder();
185             matcherAcroShort.appendReplacement(buffer, replacement);
186             array.set(i, matcherAcroShort.appendTail(buffer).toString
     ()); // inserisco la nuova riga nell'array
187           }
188         }
189       }
190     } else {...}
191  }
```

Codice 4.19: Metodo correctAcronym della classe Gestione Acronimi

Terminato il processo di correzione il lavoro è delegato al metodo **overwrite** (codice 4.20) della classe Gestione Documento, che semplicemente scrive il contenuto del array appena modificato nel documento, sovrascrivendo ciò che c'era scritto precedentemente.

```
115  public void overwrite() throws IOException {
116    PrintWriter out = new PrintWriter(new FileWriter(file, false));
117    for(String s : array)
118      out.append(s + "\n");
119    out.close();
120  }
```

Codice 4.20: Metodo overwrite della classe Gestione Documento

A questo punto il documento è stato corretto e la classe Screen manda in esecuzione il metodo **update** che non fa altro che rifare le transizioni di Acquisizione Documento (4.2 a pagina 29), Rilevazione Acronimi (4.3 a pagina 31), Rilevazioni Errori (4.5 a pagina 34) fino ad arrivare alla Visualizzazione e Interazione (4.6 a pagina 35). L'utente si ritrova così a lavorare con la lista degli errori senza l'acronimo appena corretto.

4.8 Aggiorna Dizionario

L'utente può fare scattare questa transizione ogni volta che seleziona un acronimo non corretto. E' gestita dalla classe Screen, la quale richiama metodi della classe Gestione Acronimi e Mappa Acronimi. L'aggiornamento può avvenire per una aggiunta o per una rimozione e fa effetto sul file dizionario e sulla HashMap. L'azione per aggiungere l'acronimo nel dizionario è abilitata ogni qual volta l'acronimo è errato, mentre la rimozione può essere fatta solo quando un acronimo è stato aggiunto manualmente nel dizionario. Questa scelta tra le azioni da intraprendere è svolta nel metodo **buttonDizionarioClick** della classe Screen, il quale oltre a gestire le eccezioni, verifica il codice dell'acronimo e delega alla classe Gestione Acronimi il lavoro di effettuare fisicamente delle modifiche sul dizionario.

Per quanto riguarda l'aggiunta di un elemento la classe Screen dirige il metodo **addInDictionary** della classe Gestione Acronimi (4.21 nella pagina successiva), il quale prima verifica che l'acronimo non sia già presente nel dizionario, poi controlla se la chiave dell'acronimo è già esistente, in questo caso preleva la lista degli acronimi, aggiunge l'acronimo in questione e la reinserisce, se invece non è presente crea una nuova lista con solo l'acronimo all'interno e la aggiunge nella mappa, successivamente passa ad aggiungere una riga nel file dizionario e a scrivere l'acronimo, facendo attenzione al formato giusto da usare, per finire rimuove l'acronimo dalla lista degli errori, per inserirlo in quella degli acronimi inseriti manualmente nel dizionario.

```
86  public void addInDictionary(Acronimo ac, Map<String, List<Acronimo>>
     map) throws AcronimoNonValidoException, IOException {
87   if(map.containsKey(ac.getAcronym())){       // chiave presente
88     if(map.get(ac.getAcronym()).contains(ac)) // acronimo presente
89       throw new AcronimoNonValidoException("Acronimo gia' presente
    nel dizionario.");
90     else{ // chiave presente, ma non l'acronimo
91       List<Acronimo> list = map.get(ac.getAcronym());//lista presente
92       ac.setCode(5);  // codice 5 = acronimo aggiunto manualmente
93       list.add(ac);   // aggiungo l'acronimo alla lista corrente
94       map.put(ac.getAcronym(),list); // reinserisco la lista
95       dictionaryList.add(ac);  // inserisco l'acronimo nella lista
    degli acronimi inseriti manualmente nel dizionario
96     }
97   }
98   else {  // chiave non presente
99     List<Acronimo> list = new ArrayList<>(); // creo una lista nuova
100    ac.setCode(5);  // codice 5 = acronimo aggiunto manualmente
101    list.add(ac);   // aggiungo l'acronimo alla lista appena creata
102    map.put(ac.getAcronym(),list);  // inserisco la lista nella mappa
103    dictionaryList.add(ac);  // inserisco l'acronimo nella lista
    degli acronimi inseriti manualmente nel dizionario
104  }
105  PrintWriter out = new PrintWriter(new FileWriter(MainPlugIn.
    getFileDizionario(), true)); //in una nuova riga inserisco l'
    acronimo nel dizionario senza sovrascrivere il contenuto
106  out.append("\n" + ac.getAcronym()  + " " + ac.getExtendedName());
107  out.close();
108  errorList.remove(ac);// rimuovo l'acronimo dalla lista degli errori
109 }
```

Codice 4.21: Metodo addInDictionary della classe Gestione Acronimi

Per la rimozione invece la classe Screen dirige il metodo **removeAcroFromDictionaryList**, sempre della classe Gestione Acronimi, che non fa altro che rimuovere l'acronimo dalla lista degli acronimi aggiunti manualmente nel dizionario e richiamare due metodi, **acronymAndListSet** sempre della classe Gestione Acronimi, visto precedentemente 4.13 a pagina 35, con la funzione di valutare nuovamente l'acronimo settandogli il codice di errore appropriato e inserendolo nella lista degli errori e **removeFromMap** della classe Mappa Acronimi, che fa l'opposto del metodo addInDictionary appena visto, ossia rimuove la chiave dalla mappa se la lista corrispondente contiene solo il l'acronimo che si vuole rimuovere, altrimenti richiede questa lista, toglie l'acronimo in questione e la reinserisce, successivamente passa a sovrascrivere il dizionario, riscrivendolo dall'inizio saltando la riga dove è presente l'acronimo da rimuovere.

4.9 Ignora Errore

L'utente ha la facoltà di far scattare questa transizione ogni volta che seleziona un acronimo errato, in modo da rimuoverlo dalla lista degli errori. Il sistema non considererà più errore il suddetto acronimo per tutta la durata dell'esecuzione, ma tornerà a segnarlo in utilizzi successivi. L'implementazione è svolta dalla classe Screen tramite il il metodo **deleteAcronymError** (codice 4.22) che molto semplicemente delega il compito di rimuovere fisicamente l'acronimo dalla lista degli errori alla classe Gestione Acronimi e successivamente controlla la lista che sta visualizzando l'utente e se è quella degli errori rimuove l'elemento, altrimenti si limita ad aggiornare.

```
254  private void deleteAcronymError(ActionEvent e) {
255    int acronymNumber = acronymList.getSelectedIndex();
256    if(acronymNumber >=0) {
257      Acronimo a = acronimi.get(acronymNumber);
258      gestioneAcronimi.ignoreError(a); // ignoro l'acronimo errato
259      if(labelErrorList.getText().equals(textErrorList))  // se sto
     visualizzando la lista degli errori
260        removeAcronym(a); // rimuovo l'elemento dalla lista errori
261      else
262        refreshAcronymList(); // aggiorno lista completa acronimi
263      clearField(); // ripristino i campi
264      removeErrorButton.setEnabled(false); //disabilito pulsante ignora
265    }
266  }
```

Codice 4.22: Metodo deleteAcronymError della classe Screen

Il metodo **ignoreError** (codice 4.23) della classe Gestione Acronimi rimuove l'acronimo dalla lista degli errori, setta il codice 4 all'acronimo (acronimo ignorato) e aggiunge quest'ultimo alla lista degli acronimi ignorati.

```
210  public void ignoreError(Acronimo a){
211    removeAcroError(a);
212    a.setCode(4);   // codice di errore 4 = errore ignorato
213    ignoreList.add(a);
214  }
215  public void removeAcroError(Acronimo a){
216    errorList.remove(a); // rimuove l'acronimo dalla lista degli errori
217  }
```

Codice 4.23: Metodo ignoreError della classe Gestione Acronimi

4.10 Autocorrezione

L'autocorrezione è attivata tramite un pulsante dedicato dall'utente e crea una nuova versione del documento, correggendo automaticamente tutti gli acronimi che hanno nome esteso non corretto con la correzione ritenuta ottimale dal programma. L'implementazione spetta alla classe Screen con il metodo **ButtonAutomaticCorrection** che gestisce le eccezioni e richiama il metodo **autoCorrection** (codice 4.24 nella pagina successiva) della classe Gestione Acronimo, che similmente al metodo correctAcronym (4.19 a pagina 40), permette di rilevare e correggere gli acronimi presenti nel documento. Il metodo richiede l'array dove è presente il documento originale e inizia a sfogliarlo riga per riga, per ognuna di esse cerca la corrispondenza con due pattern che ha creato, il patternAcroShort che serve ad individuare la dichiarazione di un acronimo con il nome breve e il patternAcroExt utilizzato invece per la versione con solamente il nome esteso. Una volta trovata una dichiarazione di acronimo crea temporaneamente un oggetto acronimo con i giusti dati estrapolati e lo cerca all'interno della lista degli errori. Se è presente verifica il codice di errore e in caso sia uguale a 3, cioè il problema verificato è il nome esteso non corretto, passa a correggerlo con la correzione ottimale. Per correggere l'acronimo crea una nuova riga da rimpiazzare a quella precedente, creando una nuova dichiarazione con i dati corretti e lasciando invariato tutto ciò che c'era prima e dopo di essa. Nel codice 4.24 nella pagina successiva è descritto il comportamento per il caso con il nome breve, quello con solo il nome esteso è analogo. Una volta che si ha l'array corretto, sempre il metodo della classe Screen ButtonAutomaticCorrection aggiorna il documento andando a sovrascriverlo tramite il metodo overwrite della classe Gestione Documento (visto precedentemente nel paragrafo 4.7 a pagina 41).

```
117  public void autoCorrection(ArrayList<String> array) {
118    Pattern patternAcroShort = Pattern.compile("(.*)(\\\\acro\\s*\\{)
     (\\w*)(}\\s*\\[](.+)(]\\s*\\{)(.+)(})(.*)");  // creo il pattern
     con 9 gruppi --> gruppo1:(qualsiasi eventuale carattere), gruppo2:(
     istruzione \acro, eventuale qualsiasi spaziatura e parentesi { ),
     gruppo3:(acronimo), gruppo4:(parentesi } spazio eventuale e
     parentesi [ ), gruppo5:(nome breve), gruppo6:(parentesi ] spazio
     eventuale e parerntesi { ), gruppo7:(nome esteso), gruppo8:(
     parentesi } ), gruppo9:(qualsiasi eventuale carattere)
119    Pattern patternAcroExt  = Pattern.compile("(.*)(\\\\acro\\s*\\{)(\\
     w*)(}\\s*\\{)(.+)(})(.*)"); // creo il pattern con 7 gruppi -->
     gruppo1:(qualsiasi eventuale carattere), gruppo2:(istruzione \acro,
      eventuale qualsiasi spaziatura e parentesi { ), gruppo3:(acronimo)
     , gruppo4:(parentesi } spazio eventuale e parentesi { ), gruppo5:(
     nome esteso), gruppo6:(parentesi } ), gruppo7:(qls ev. carattere)
120    for (int i=0; i < array.size();i++) {
121      String s = array.get(i);
122      Matcher matcherAcroShort = patternAcroShort.matcher(s);
123      Matcher matcherAcroExt = patternAcroExt.matcher(s);
124      if (matcherAcroShort.matches()) { // versione nome breve
125        String acronym = matcherAcroShort.group(3);
126        String shortName = matcherAcroShort.group(5);
127        String extendedName = matcherAcroShort.group(7);
128        Acronimo a = new Acronimo(acronym, shortName, extendedName);
129        if(searchInErrorList(a)!=null){// presente lista errori
130          Acronimo acro = searchInErrorList(a); // restituisce l'
     acronimo presente nella lista degli errori
131          if(acro.getCode()==3) { // code3 = nome esteso non corretto
132            String backslash = ""; // Stringa che mi permette di
     aggiungere il carattere speciale LaTeX "\" senza causare problemi
     con le regex
133            if(shortName.startsWith("\\")) // nel nome breve "\" e'
     posizionato all'inizio del nome
134              backslash="\\\\";
135            String correctName = acro.getOptimalCorrection(); //
     correzione ottimale
136            String replacement = matcherAcroShort.group(1) + "\\\\acro{
     " + acronym + "} [" + backslash + shortName + "] {" + correctName +
      "}" + matcherAcroShort.group(9); //correggo l'acronimo, lascio
     invariato cio' che c'era prima e dopo di esso
137            StringBuilder buffer = new StringBuilder();
138            matcherAcroShort.appendReplacement(buffer, replacement);
139            array.set(i, matcherAcroShort.appendTail(buffer).toString()
     ); // inserisco la nuova riga nell'array
140          }
141        }
142      } else if (matcherAcroExt.matches()) {...}
143    }
144  }
```

Codice 4.24: Metodo autoCorrection della classe Gestione Acronimi

4.11 Main

Il funzionamento collettivo è diretto dalla classe **Main PlugIn** che si occupa di istanziare oggetti delle classi Gestione Acronimi, Gestione Documeto, Mappa Acronimi e Screen, fornire i percorsi del file documento e del file dizionario e di seguire il funzionamento descritto dalla rete di Petri in figura 4.1 a pagina 24, gestendo le eccezioni.

Capitolo 5

Esempio funzionale

Di seguito vediamo un esempio delle funzionalità che offre l'interfaccia grafica.

Prendiamo il caso dell'acronimo Particulate Matter (PM), nella versione italiana Particolato. Nel documento è stato definito erroneamente come Particolare, per questo è finito nella lista degli errori (esempio in figura 5.1).

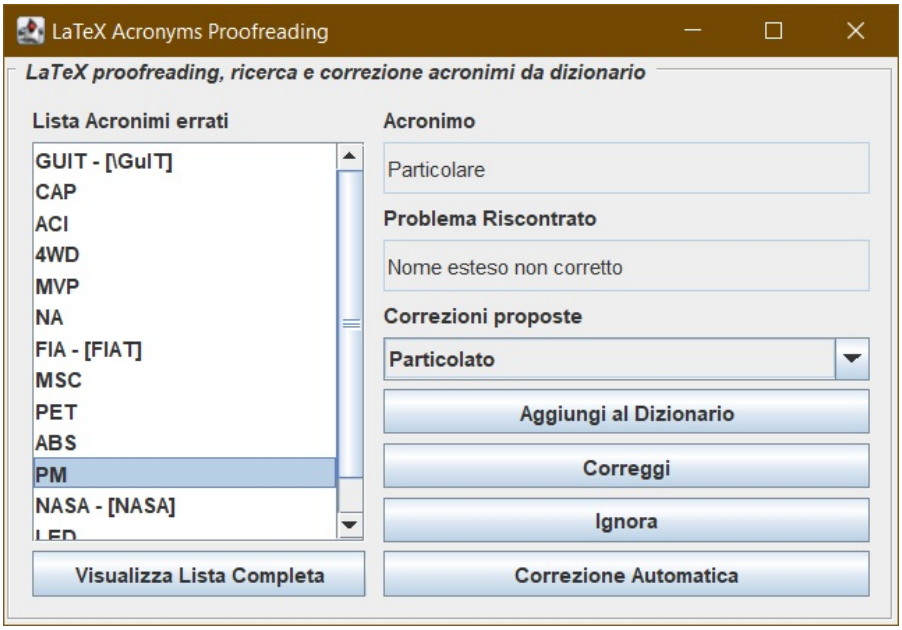

Figura 5.1: Esempio acronimo errato

possiamo vedere nel campo Acronimo il nome esteso errato dell'acronimo (Particolare) e nel campo Problema Riscontrato appunto la conferma che è stato inserito un nome esteso non corretto. Nel campo Correzioni proposte ci è suggerita la correzione Particolato, aprendo la lista drop-down possiamo vedere tutte le correzioni disponibili per questo acronimo (figura 5.2 nella pagina successiva):

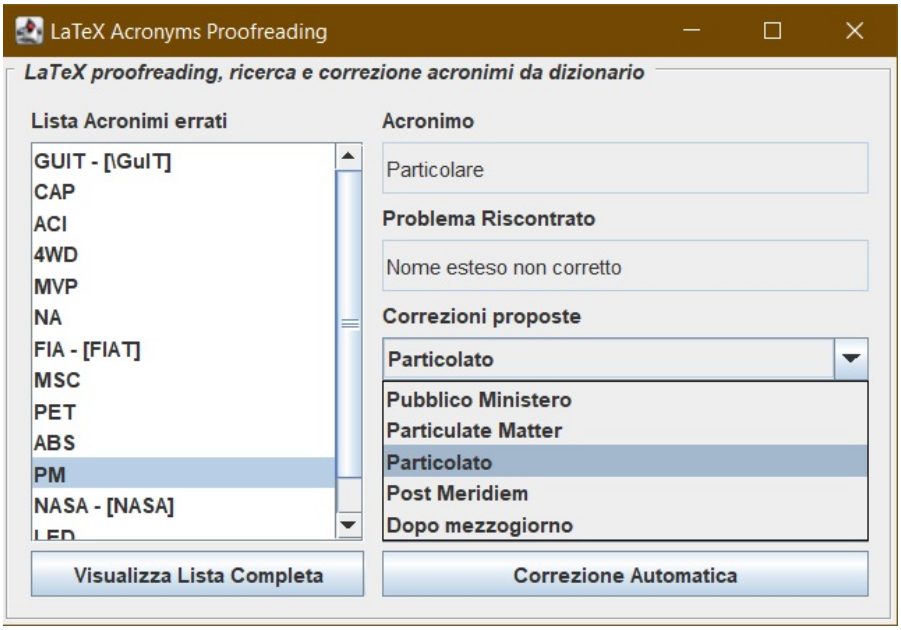

Figura 5.2: Esempio correzioni proposte

Infatti il plugin ci propone come correzione i nomi estesi di Pubblico Ministero, Particulate Matter, Particolato (selezionato di default), Post Meridiem e Dopo Mezzogiorno. Lasciamo selezionato Particolato, il programma ci offre le funzionalità di correggere, ignorare o aggiungere nel dizionario l'acronimo, premiamo sul pulsante Correggi:

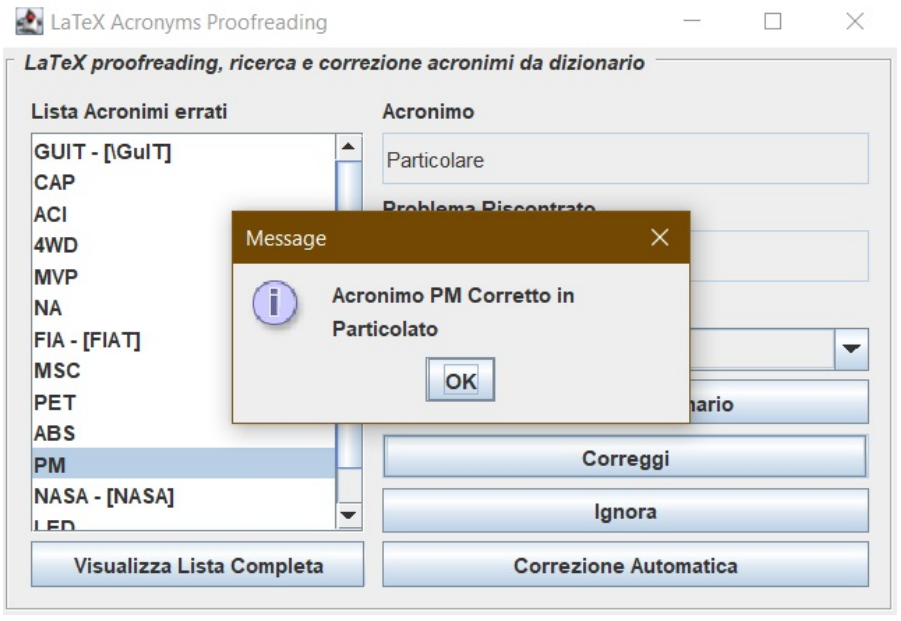

Figura 5.3: Esempio messaggio pop-up

Il messaggio pop-up in figura 5.3 ci comunica che l'operazione di correzione è andata a buon fine.

Capitolo 6

Limiti e possibili sviluppi

Lo scopo del proofreading è verificare la totale assenza di errori grammaticali, sintattici, ortografici e, in caso contrario, correggerli. Solitamente il proofreading di un documento si svolge in modo manuale, infatti è consigliato stampare il testo e leggerlo totalmente per scovare errori. Come si può facilmente intuire questa procedura può essere molto lunga ed è possibile comunque non accorgersi di alcune scorrettezze. Questo programma costituisce un potente strumento a supporto di questa funzionalità, ma si limita a considerare solamente gli acronimi. Fare il proofreading è diverso da eseguire un semplice controllo grammaticale, poiché una frase può essere costituita da parole grammaticalmente corrette, ma essere sintatticamente scorretta. Un possibile sviluppo di questo tool può essere quello di aggiungere altre funzionalità oltre al proofreading degli acronimi. Ad esempio una direzione di espansione è costituita dall'aggiungere in primo luogo il controllo di simboli, spazi e apostrofi, per poi passare in secondo luogo alla capitalizzazione delle parole, aggiungere progressivamente un controllo di tutti gli elementi che possono costituire una frase, per poi in ultimo luogo controllare tutte le parole e il loro senso sintattico in base alle altre.

Acronimi

WYSIWYG What You See Is What You Get 7

WYSIAYG What You See Is All You Get 7

WYSIWYM What You See Is What You Mean 7

YAFIYGI You Asked For It You got it 7

VPN Virtual Private Network 9

LED Light Emitting Diode 14

GUI Grafic User Interface 19

SO Sistema Operativo 19

GUIT Gruppo Utilizzatori Italiani di TEX e LATEX 20

IDE Ambiente di sviluppo integrato 20

PM Particulate Matter 47

Bibliografia

[1] *Documentazione Oracle*. URL: https://docs.oracle.com/javase/8/docs/api/overview-summary.html.

[2] *Guida Java di html.com*. URL: https://www.html.it/guide/guida-java/.

[3] Tobias Oetiker. *An Acronym Environment for LaTeX*. 2020. URL: https://ctan.mirror.garr.it/mirrors/ctan/macros/latex/contrib/acronym/acronym.pdf.

[4] Lorenzo Pantieri. *LaTeXpedia*. 2017. URL: http://www.lorenzopantieri.net/LaTeX_files/LaTeXpedia.pdf.

[5] Lorenzo Pantieri e Tommaso Gordini. *L'arte di scrivere con LaTeX*. 2008. URL: http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf.