Dedico questo lavoro alla vita
a ciò che mi ha dato
agli insegnamenti ricevuti da ciò che non ho più
ai miei grandi amori
alle speranze e alle paure di ciò che sarà
perché in fondo senza questo non sarebbe vita.

UNIVERSITÁ DEGLI STUDI DEL SANNIO
FACOLTÁ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

Tesi di Laurea in
Basi di Dati

"Progettazione e Realizzazione di un Sistema per la ricostruzione ed individuazione dei Requisiti Software attraverso tecniche di Software Repository Mining"

Relatore:
Chiar.mo Prof. Gerardo CANFORA
Prof. Corrado Aaron VISAGGIO
Ing. Luigi CERULO

Candidato:
Leonardo CORBO

Anno Accademico 2006/2007

Indice

Capitolo 1 – Introduzione

1.1 La progettazione del software

L'informatica è una disciplina che, a partire dagli anni settanta, ha rivoluzionato il mondo ed ha cambiato radicalmente il modo di concepirlo, di affrontare i problemi quotidiani e non, e perfino di abbattere i vincoli dovuti alle distanze. La comodità legata al risparmio di risorse personali fisiche e mentali e l'utilità derivante dall'elevata precisione di elaborazione delle unità di calcolo ha sempre suscitato molto interesse nello sviluppo della componentistica degli elaboratori da una parte e delle applicazioni che la sfruttano dall'altra.

Nell'ultimo decennio si è assistito ad una profonda compenetrazione del supporto informatico praticamente in tutti gli ambiti della società civile, siano essi aziendali o domestici, artigianali o industriali, dilettantistici o professionali. Ciò ha creato un vero e proprio mercato basato sulla vendita di servizi sotto forma di applicativi software studiati per risolvere determinati problemi, molto spesso commissionati direttamente dai clienti.

In primo luogo è chiaro che un mercato del genere prospetti potenziali guadagni e quindi sia contraddistinto dalla concorrenza tra le aziende operanti nello stesso settore specifico; molto spesso commettere un errore nello sviluppo di una soluzione software può comportare la perdita di potenziali profitti per il cliente e di conseguenza uno screditamento della reputazione aziendale. Nasce da qui l'esigenza della definizione di un metodo per uno sviluppo applicativo efficace.

In secondo luogo l'aumento della produttività e lo sfruttamento sempre più efficiente delle risorse a disposizione ha aumentato la richiesta di applicativi sempre più elaborati e con numerose funzionalità, rendendo quindi spesso difficoltosa la loro progettazione.
L'insieme di questi due bisogni ha portato alla necessità di definire una metodologia formale e rigorosa per lo sviluppo software, dalla semplice idea iniziale fino alla consegna al cliente dell'applicazione. È su questo campo che si è sviluppata la disciplina dell'*ingegneria del software*, che è volta alla schematizzazione dell'intero processo di sviluppo ed alla produzione di una dettagliata documentazione in ogni fase del ciclo di vita del prodotto software.

1.2 I requisiti software

Secondo i canoni dell'ingegneria del software, prima di poter pensare alla progettazione di un'applicazione bisogna attraversare una fase molto delicata di reperimento delle informazioni necessarie alla corretta e completa comprensione del problema che la soluzione software deve affrontare. Tale raccolta può rivelarsi molto ardua da condurre in quanto bisogna afferrare tutti gli aspetti del particolare dominio in questione, con il quale molto spesso l'ingegnere del software non ha familiarità; bisogna quindi quasi sempre avvalersi del supporto di esperti del settore, con i quali possono comunque esservi dei problemi di comunicazione per via dei diversi linguaggi tecnici utilizzati e dei punti di vista non sempre convergenti.
L'insieme delle necessità emerse dall'analisi del dominio applicativo viene opportunamente formalizzato dando vita ai cosiddetti *requisiti software*, dettagliate descrizioni formali dal punto di vista del progettista software delle funzionalità che

l'applicazione dovrebbe fornire. Il tutto viene solitamente riportato in un documento ufficiale chiamato *SRS* (*Software Requirements Specification*).

La fase di individuazione di requisiti precisi e completi riguarda potenzialmente tutto il ciclo di vita dell'applicazione ed è così importante all'interno dell'ingegneria del software che ha portato alla creazione di una disciplina propria: l'*ingegneria dei requisiti*.

Tipicamente, il processo dell'ingegneria dei requisiti attraversa le fasi seguenti (Figura 1.1):

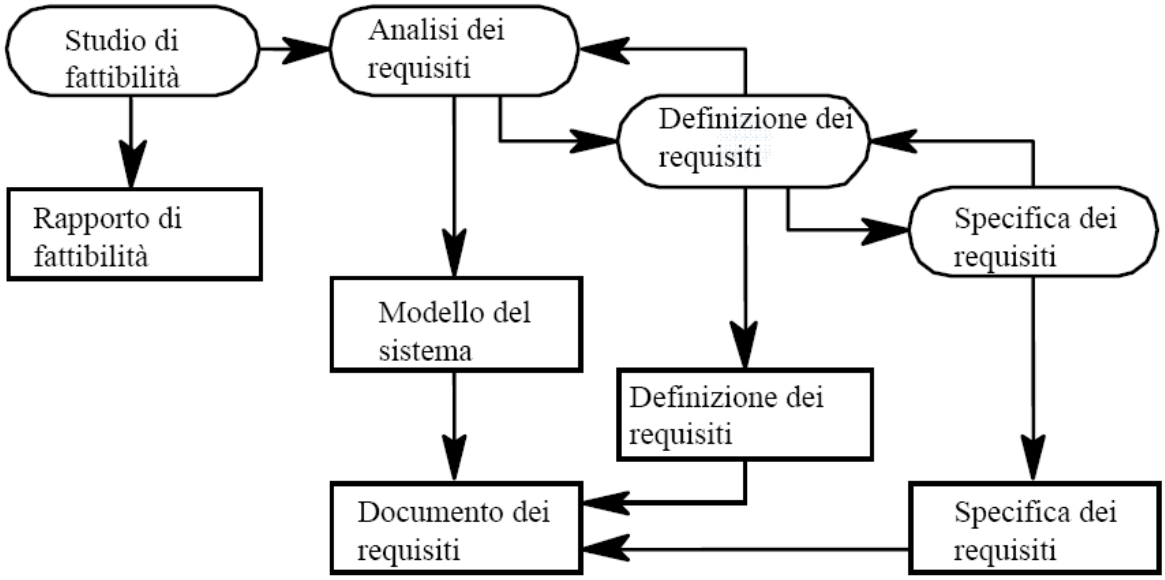


Figura 1.1 Processo dell'ingegneria dei requisiti

in cui le operazioni fondamentali sono quattro:

1. *Studio di fattibilità*: consiste nel valutare se le necessità attuali del cliente possono essere soddisfatte con la tecnologia ed il budget disponibile;
2. *Analisi dei requisiti*: consiste nella comprensione, con l'aiuto di esperti nel settore, dei requisiti del sistema;

3. *Definizione dei requisiti*: consiste nella definizione dei requisiti in una forma comprensibile dal cliente;
4. *Specifica dei requisiti*: consiste nella definizione in dettaglio dei requisiti dal punto di vista dell'ingegnere del software.

1.3 Problematiche nell'ingegneria dei requisiti

Tanti sono i problemi legati allo sviluppo ed alla risoluzione dei requisiti software, ma due li sintetizzano in maniera significativa.

Da una parte, la crescente richiesta di applicazioni sempre più sofisticate e mirate all'implementazione di uno spettro sempre più ampio di funzionalità ha determinato una crescita del numero di requisiti da trattare e soddisfare, con un relativo aumento di complessità nelle definizioni degli stessi e nella stesura del SRS. Soprattutto a causa della crescente affermazione di metodologie di progettazione software open-source, che non seguono un rigido schema operativo tipico di una struttura aziendale, risulta sempre più difficile verificare se i requisiti software sono stati effettivamente affrontati dalla versione attuale dell'applicazione, e in caso affermativo se sono stati affrontati in maniera adeguata.

Dall'altra parte, l'evoluzione dei requisiti non consente neppure di definire e specificare in maniera definitiva il loro numero e la loro forma. Dalla Figura 1.1 si nota subito che, superata la fase dello studio di fattibilità, le altre tre fasi possono richiamarsi l'una con l'altra diverse volte. Ciò accade molto spesso proprio per via della cattiva comunicazione tra l'ingegnere del software ed il cliente come detto nel paragrafo precedente, il che comporta una revisione nell'espressione e/o nel contenuto dei requisiti, ma può essere attribuibile anche ad un cambiamento della conoscenza. Non è raro infatti che tanto il

committente quanto l'ingegnere del software possano rendersi conto di non aver preso in considerazione degli aspetti del dominio, o di non averli considerati adeguatamente, e debbano rivedere il documento dei requisiti per modificare, aggiungere od eliminare alcuni di essi sulla base delle nuove conoscenze.

In genere, i requisiti influenzati da questi aspetti ricadono nelle seguenti tipologie:

- *requisiti duplicati*: requisiti trattanti in realtà lo stesso argomento, o i cui argomenti sono coperti da un insieme di altri requisiti;
- *requisiti instabili*: requisiti che cambiano a causa dell'ambiente del sistema;
- *requisiti emergenti*: requisiti che nascono quando aumenta la comprensione del problema;
- *requisiti indotti*: requisiti che nascono dall'introduzione del sistema software;
- *requisiti di compatibilità*: requisiti che dipendono da altri sistemi o processi organizzativi.
- *requisiti impliciti*: requisiti che sono specifici del dominio e per questo non completamente o tempestivamente esplicitati e quindi formalizzati.

Tra questi, rivestono particolare importanza i requisiti emergenti, perché sono quelli che permettono l'evoluzione del sistema con l'aggiunta di nuove funzionalità o proprietà molto spesso direttamente riscontrabili anche dal cliente.

Il lavoro oggetto della presente tesi affronta le due importanti problematiche appena esposte. Fermo restando che il processo di manutenzione dei requisiti deve essere condotto dagli ingegneri del software, si è cercato di creare uno strumento che possa affiancarli ed essere loro di supporto fornendo innanzitutto una stima su quanto i requisiti attuali sono stati affrontati, e offrendo linee guida per individuare più facilmente eventuali requisiti duplicati ed emergenti.

In particolare, nella trattazione sarà dapprima illustrato lo stato dell'arte nel campo della tracciabilità dei requisiti ed in quello della ricerca e recupero delle informazioni (Capitolo 2); successivamente si vedrà il metodo utilizzato per affrontare le problematiche viste (Capitolo 3), seguito da una dettagliata analisi del tool che lo implementa (Capitolo 4). Infine si darà una valutazione della qualità di quest'ultimo, provandolo su un caso reale (Capitolo 5).

Capitolo 2 – Stato dell'arte

2.1 Introduzione

Come visto nel capitolo precedente, l'ambito in cui si colloca l'oggetto della presente tesi è piuttosto vasto e copre diversi aspetti inerenti l'ingegneria del software. Quello sicuramente più importante, solo accennato nel Capitolo 1, è legato al problema della *tracciabilità dei requisiti*, consistente nella ricerca di una tecnica usata per fornire una relazione tra i requisiti, il progetto e l'implementazione finale del sistema. Queste relazioni permettono ai progettisti di mostrare il motivo ed il modo in cui il progetto soddisfa i requisiti dei vari committenti e aiutano nell'individuazione precoce di quelli che invece non sono ancora soddisfatti.

Inoltre, un ambito che nel corso degli anni ha fornito spunti per l'individuazione di tecniche a supporto della tracciabilità dei requisiti è l'information retrieval: è proprio in quest'ultimo campo che si è mossa la ricerca e lo sviluppo del metodo per la risoluzione delle problematiche presentate, ed è anche questo il campo in cui sono stati condotti numerosi interessanti studi e sviluppate metodologie basate su diversi principi.

Nel seguito si approfondiranno le conoscenze attuali in questi settori.

2.2 Tracciabilità dei requisiti

Lo sviluppo e l'uso di tecniche per il tracciamento dei requisiti hanno origine nei primi anni settanta per influenzare le caratteristiche di completezza e consistenza dei requisiti di un sistema.

Come accennato in precedenza, la tracciabilità dei requisiti si pone nell'ottica di assicurare che il prodotto software soddisfi le aspettative del cliente che lo ha commissionato. Ciò, in particolare, conduce ad un lavoro metodico congiunto tra fornitore e committente, nel quale è possibile dimostrare progressivamente che gli sviluppatori hanno compreso i requisiti, come ogni requisito è stato soddisfatto e viceversa come ogni componente del sistema soddisfa un requisito, e che il prodotto non ha bisogno di caratteristiche non necessarie.

La tracciabilità dovrebbe inoltre mostrare come si è arrivati ai requisiti correnti: il razionale di progettazione deve identificare non solo le decisioni, ma indicare anche le assunzioni e le motivazioni a supporto o a confutazione dietro tali decisioni, esplicitando il contesto nel quale esse sono state prese. In questo modo, viene facilitata la comunicazione tra coloro che sono coinvolti nel progetto e di conseguenza l'intera gestione dei requisiti.

Attraverso le informazioni raccolte, i progettisti e i manutentori possono determinare facilmente gli effetti dei cambiamenti prima di riprogettare il sistema, in modo da avere un'idea dei costi e programmare le attività. Tutto ciò senza dipendere dalla conoscenza di ingegneri e programmatori in tutte le aree coinvolte da tali cambiamenti.

Inoltre, le informazioni di tracciabilità possono essere usate da coloro che sviluppano ed eseguono i test per determinare quali di questi effettuare e per modificarli correttamente nel caso siano individuati degli errori.

2.2.1 Problema della tracciabilità

Ad oggi, il problema della tracciabilità rimane ancora una questione aperta. Nonostante l'aumento sia nella domanda che nell'offerta di tool che includono funzionalità di tracciabilità, il loro utilizzo pratico non è tanto diffuso quanto l'importanza della tracciabilità richiederebbe. Inoltre, i problemi di tracciabilità sussistono anche lì dove viene utilizzata. Studi empirici con professionisti nel settore hanno rilevato che il problema della tracciabilità non è percepito come uniforme, a causa della diversità delle definizioni e ad un numero di conflitti fondamentali.

Le definizioni di tracciabilità dei requisiti, utilizzate dagli sviluppatori e presenti in letteratura, si possono classificare come:

- *Purpose-driven* (definite in termini di cosa dovrebbe fare)

  "...l'abilità di aderire alla posizione di business, alla portata del progetto e ai requisiti chiave che sono stati sottoscritti"

- *Solution-driven* (definite in termini di come dovrebbe farlo)

  "...l'abilità di avere delle tracce da un'entità a un'altra basandosi su relazioni semantiche date"

- *Information-driven* (enfatizza le informazioni di tracciabilità)

  "...l'abilità di collegare insieme funzioni, dati e requisiti alle formulazioni testuali che vi si riferiscono"

- *Direction-driven* (enfatizza la direzione della tracciabilità)

  "...l'abilità di collegare uno specifico item all'inizio di una fase del ciclo di vita software ad un altro alla fine della stessa fase"

Come si può notare, nessuna definizione copre tutti gli aspetti; piuttosto ognuna differisce nell'enfasi e delimita un particolare ambito. Questo ha delle ripercussioni nello sviluppo e

nell'utilizzo di tool che supportano la tracciabilità, in quanto diventa arduo implementarla coerentemente e consistentemente dato che ogni individuo ha la sua concezione di cosa rappresenta il problema stesso della tracciabilità.

A queste difficoltà si deve aggiungere anche che ogni sviluppatore ha la sua idea su quale sia la causa principale del problema della tracciabilità e che, ogni volta in cui la tracciabilità è richiesta, entrano in gioco diversi utenti, progetti, task e requisiti.

2.2.2 Tracciabilità pre/post specifica dei requisiti

Si possono distinguere due tipi di tracciabilità dei requisiti:

- *Tracciabilità pre-RS* (specifica dei requisiti), la quale si riferisce a quegli aspetti della vita di un requisito precedentemente alla sua inclusione nella specifica (requirements production).
- *Tracciabilità post-RS*, che si riferisce a quegli aspetti della vita di un requisito successivamente alla sua inclusione nella specifica (requirements deployment).

La figura seguente mette in evidenza alcuni aspetti relativi alle definizioni appena fornite (Figura 2.1):

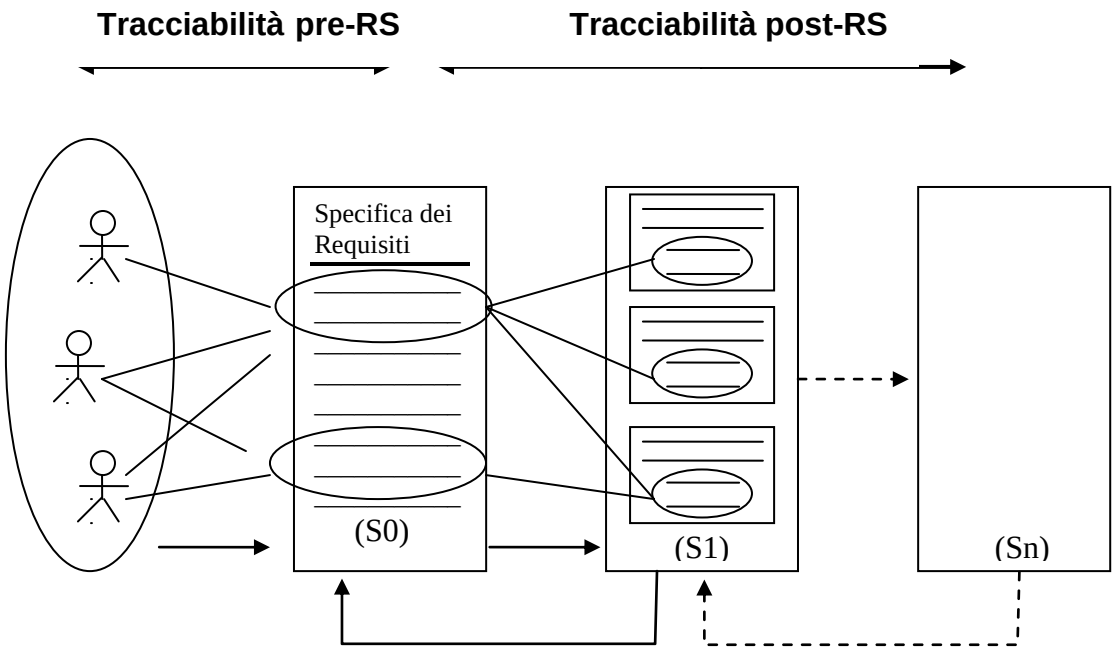


Figura 2.1 Tracciabilità pre-RS e post-RS

È da notare sia come la conoscenza dei requisiti è distribuita e fusa insieme in successive rappresentazioni sia la complicazione aggiunta delle iterazioni e della propagazione dei cambiamenti.

La tracciabilità di tipo *forward* e quella di tipo *backward* sono entrambe essenziali. In questo schema, però, è stata enfatizzata la separazione tra tracciabilità pre-RS e post-RS, dato che i problemi relativi alla tracciabilità che sono stati rilevati sono soprattutto incentrati sulla mancanza di distinzione in questo punto.

Le principali differenze tra questi due tipi di tracciabilità coinvolgono le informazioni con cui hanno a che fare e i problemi che possono affrontare. La tracciabilità post-RS dipende dall'abilità di tracciare i requisiti da e verso la loro specifica, attraverso una successione di artefatti nei quali essi sono distribuiti. I cambiamenti alla specifica dei requisiti devono essere propagati attraverso questa catena. La tracciabilità pre-RS, invece, dipende dall'abilità di tracciare i requisiti da e verso la loro formulazione originaria, attraverso il

processo di produzione e raffinamento dei requisiti, nelle quali le formulazioni provenienti da diverse sorgenti sono prima o poi integrate in singoli requisiti all'interno della specifica. I cambiamenti nel processo devono subire una rilavorazione all'interno della specifica.

Il supporto esistente fornisce principalmente la tracciabilità post-RS. I problemi da risolvere in questo ambito riguardano un artefatto o dei metodi informali di sviluppo. Di contro, i problemi da affrontare con la tracciabilità pre-RS non sono né ben compresi né pienamente supportati: il supporto alla tracciabilità post-RS in questo caso non è adatto, dato che in genere tratta la specifica dei requisiti come una black-box, in cui è messo poco in evidenza che in realtà i requisiti sono il prodotto finale di un processo complesso e in continua evoluzione. Attenersi rigidamente alle categorie per registrare le informazioni rende inoltre difficile rappresentare questo processo a causa della natura dinamica delle sorgenti e dell'ambiente dai quali i requisiti vengono elicitati.

2.2.3 Il problema delle sorgenti umane dei requisiti

L'incapacità di rispondere a domande relative alle sorgenti umane dei requisiti è determinante nei problemi relativi alla tracciabilità dei requisiti.

Tranne alcune eccezioni, gli sforzi per migliorare il potenziale della tracciabilità hanno per lo più riguardato scoprire e registrare la maggior quantità di informazioni possibile circa il processo di ingegneria dei requisiti, collegandola successivamente in svariati modi per la determinazione delle tracce. Questo può condurre ad una massa non strutturata e inutilizzabile di dati, senza una discriminazione a priori circa il tipo e i propositi delle informazioni sui requisiti di cui gli sviluppatori hanno bisogno.

Seguendo studi empirici è possibile dimostrare che l'informazione più importante da registrare per la risoluzione a lungo termine di problemi riguardanti la tracciabilità, è quella identificata dalle sorgenti umane delle informazioni. Si è capito che quelli che si ritengono essere problemi relativi alla tracciabilità dei requisiti tendono a emergere quando gli sviluppatori non sono capaci di rispondere a domande circa il personale coinvolto nella produzione e nel raffinamento dei requisiti. Questo dipende dal fatto che le persone sono spesso considerate l'ultima soluzione quando i requisiti hanno bisogno di essere riesaminati o necessitano di una rilavorazione.

Ciò riflette il fatto che le persone sono spesso l'autorità finale dei requisiti e, in quanto tali, sono frequentemente capaci di prevenire i potenziali problemi di tracciabilità. Nondimeno, l'abilità di collocare individui e gruppi appropriati è nella pratica estremamente difficile. L'incapacità di collocare le sorgenti umane dei requisiti reali, le informazioni ed il lavoro legati ai requisiti (e quindi di accedervi) è uno dei punti cruciali del problema multisfaccettato della tracciabilità. Da qui, è stato proposto di rendere espliciti, e quindi tracciabili, i dettagli delle impostazioni sociali che danno risalto agli artefatti prodotti nell'ingegneria dei requisiti.

2.2.4 Modelli di riferimento per la tracciabilità

Molti standard che richiedono la tracciabilità dei requisiti non forniscono un modello completo di come l'informazione debba essere catturata e utilizzata come parte di uno schema di tracciabilità, in modo da assicurare che questa sia mantenuta durante tutte le fasi del processo di sviluppo, a partire dalla contrattazione con il cliente fino al testing e oltre.

Nel corso degli ultimi anni sono stati proposti diversi modelli, per lo più basati su considerazioni teoriche o su analisi effettuate dalla letteratura esistente. In questa sede ci si concentrerà sui risultati del lavoro di Ramesh. A differenza degli altri modelli, Ramesh ha seguito un approccio empirico per sintetizzare alcuni modelli di riferimento, validati successivamente con dei casi di studio e inclusi in diversi tool per la tracciabilità. Questi modelli comprendono i più importanti tipi di collegamenti di tracciabilità tra i diversi task del processo di sviluppo software.

I modelli di riferimento in generale sono modelli prototipali di alcuni domini applicativi. Lo scopo dei modelli di riferimento è ridurre in maniera significativa la creazione di modelli e sistemi per un dominio applicativo: l'utente seleziona parti rilevanti del modello di riferimento, le adatta al problema in questione e configura la soluzione generale a partire da queste parti adattate. È stato stimato che, dal momento che l'analisi di un dominio può richiedere uno sforzo enorme quando parte dal nulla, l'uso di modelli di riferimento permette di risparmiare fino all'80% dei costi di sviluppo per sistemi in domini standardizzati.

Naturalmente non tutti i domini applicativi sono sufficientemente standardizzati da permettere la definizione di un modello orientato al prodotto finale; d'altra parte è possibile vedere i modelli come modi per riutilizzare l'esperienza legata ai processi di sviluppo, e non come un insieme di regole stabilite. È in quest'ottica che si pone l'uso di modelli per la tracciabilità dei requisiti.

Nei suoi studi Ramesh ha diviso gli utenti in due gruppi distinti rispetto alle loro pratiche di tracciabilità dei requisiti:

- *low-end traceability user*: sono utenti che hanno pochi anni di esperienza nel campo della tracciabilità e la vedono semplicemente come un obbligo imposto dai sostenitori del progetto o dagli standard in uso.

- *high-end traceability user*: sono utenti che hanno diversi anni di esperienza nella tracciabilità dei requisiti e la vedono come una componente fondamentale del processo ingegneristico per sistemi di qualità; per loro rappresenta una maggiore opportunità per raggiungere la soddisfazione del cliente e per la creazione di conoscenza durante tutto il ciclo di vita del sistema.

I modelli di riferimento considerati assumono un'implementazione basata su repository delle tracce (manuali o digitali), i quali devono comprendere almeno tre livelli:

1. il *meta-modello* che definisce il linguaggio in cui i modelli della tracciabilità possono essere definiti;
2. un insieme di *modelli di tracciabilità di riferimento* che possono essere personalizzati all'interno dell'ambito definito dal meta-modello;
3. un *database* (possibilmente distribuito) delle tracce attuali, registrate sotto i modelli scelti.

Gli aspetti fondamentali della tracciabilità possono essere catturati con un meta-modello molto semplice, come quello mostrato in Figura 2.2, il quale quindi fornisce le primitive base per catalogare e descrivere i modelli della tracciabilità con maggior dettaglio:

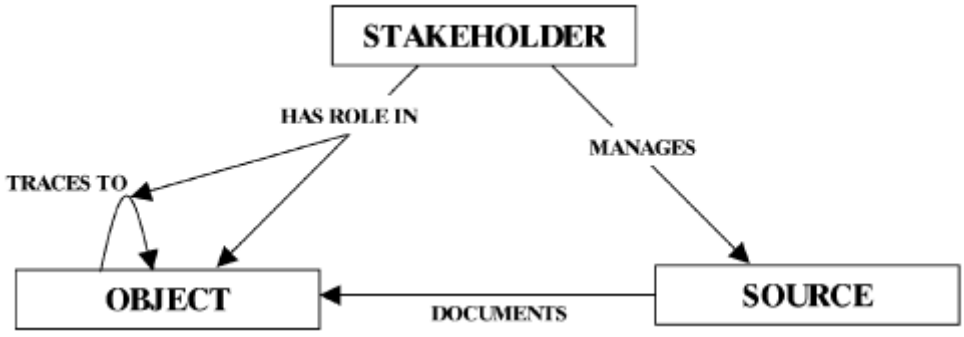


Figura 2.2 Metamodello

Ogni entità e collegamento nel meta-modello può essere specializzato e istanziato per creare modelli di tracciabilità specifici per un progetto o per un'organizzazione.

In questo schema le entità utilizzate sono le seguenti:

- *oggetti*: rappresentano gli input e gli output del processo di sviluppo del sistema (requisiti, assunzioni, progetti, componenti del sistema, decisioni, ecc.). Essi sono creati dai task organizzativi (ad esempio attività di analisi e progettazione del sistema), i quali possono essere considerati gli attributi degli oggetti. La tracciabilità tra vari oggetti è rappresentata dai collegamenti *traces-to*;
- *stakeholder*: rappresentano gli agenti implicati nelle attività di sviluppo del sistema e di mantenimento del ciclo di vita (manager di progetto, analisti di sistema, progettisti, ecc.). Questi assumono diversi ruoli nella definizione e nell'uso degli oggetti concettuali e dei collegamenti;
- *sorgenti*: rappresentano la documentazione degli oggetti, e possono essere materiali (documenti) o intangibili (riferimenti a persone o politiche e procedure non documentate). Esempi di sorgenti sono: documenti di specifica dei requisiti, meeting minute, documenti di progetto, memorandum, chiamate telefoniche, come anche riferimenti ai vari stakeholder che li gestiscono.

2.2.5 Tracciabilità low-end

Il modello per la tracciabilità ricavato dalle pratiche degli utenti low-end è mostrato in Figura 2.3:

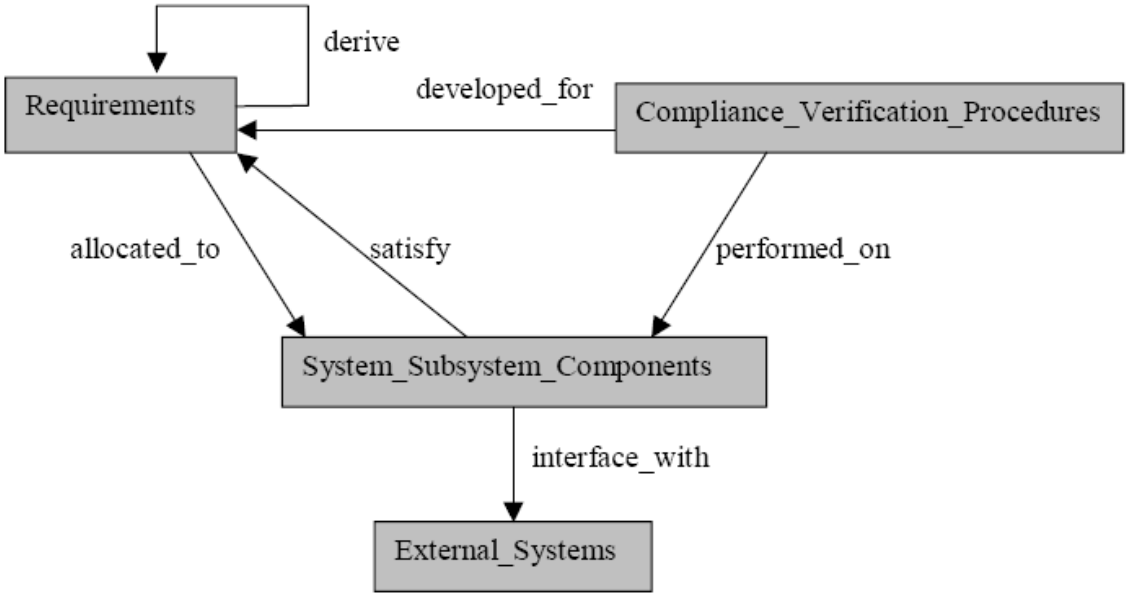


Figura 2.3 Modello di tracciabilità per gli utenti low-end

Esso prende in considerazione i seguenti aspetti:

- la tracciabilità è intesa come il provvedere un collegamento tra i *requisiti* iniziali e i *componenti del sistema* che in un dato momento li soddisfano;
- i requisiti di più basso livello sono derivati da quelli originali di alto livello attraverso un processo di decomposizione ricorsiva. I requisiti così ottenuti sono allocati ai componenti del sistema (ad esempio con una tabella di allocazione);
- le *procedure di verifica della compliance* (CVP), quali test e simulazioni, sono sviluppate a partire dalla versione più recente dei requisiti validati del sistema (mantenuti nel database dei requisiti); queste sono mantenute in una matrice requisiti-test, in modo che, se occorresse modificare un requisito, utilizzando il collegamento di tracciabilità sarebbe possibile identificare le CVP che devono essere cambiate o rielaborate;
- le CVP sono eseguite sui componenti del sistema in modo da verificare che questi soddisfino i requisiti;

- un componente del sistema può dipendere da altri e può anche interfacciarsi con *sistemi esterni*. Questa informazione è usata nella valutazione di come un requisito è soddisfatto da un componente del sistema.

Gli utenti low-end di solito peccano nella cattura del razionale. Ad esempio, nella gestione dei requisiti le informazioni relative alle questioni sui requisiti, su come queste sono risolte e il razionale delle decisioni sono raramente catturate. Ciò accade in modo simile anche nelle fasi di progettazione e implementazione.

2.2.6 Tracciabilità high-end

Rispetto agli utenti low-end, quelli high-end utilizzano schemi di tracciabilità più completi ed inoltre usano le informazioni in modi più ricchi. Per questo motivo il modello high-end può essere suddiviso in quattro sottomodelli:

- modello di *gestione dei requisiti*;
- modello del *razionale*;
- modello di *design/allocation*;
- modello di *verifica della compliance*.

2.2.6.1 Modello di gestione dei requisiti

Con il modello di gestione dei requisiti mostrato in Figura 2.4, i requisiti possono essere rintracciati attraverso il ciclo di vita per permettere agli stakeholder di capire e valutare se il sistema supporta i fattori critici di successo.

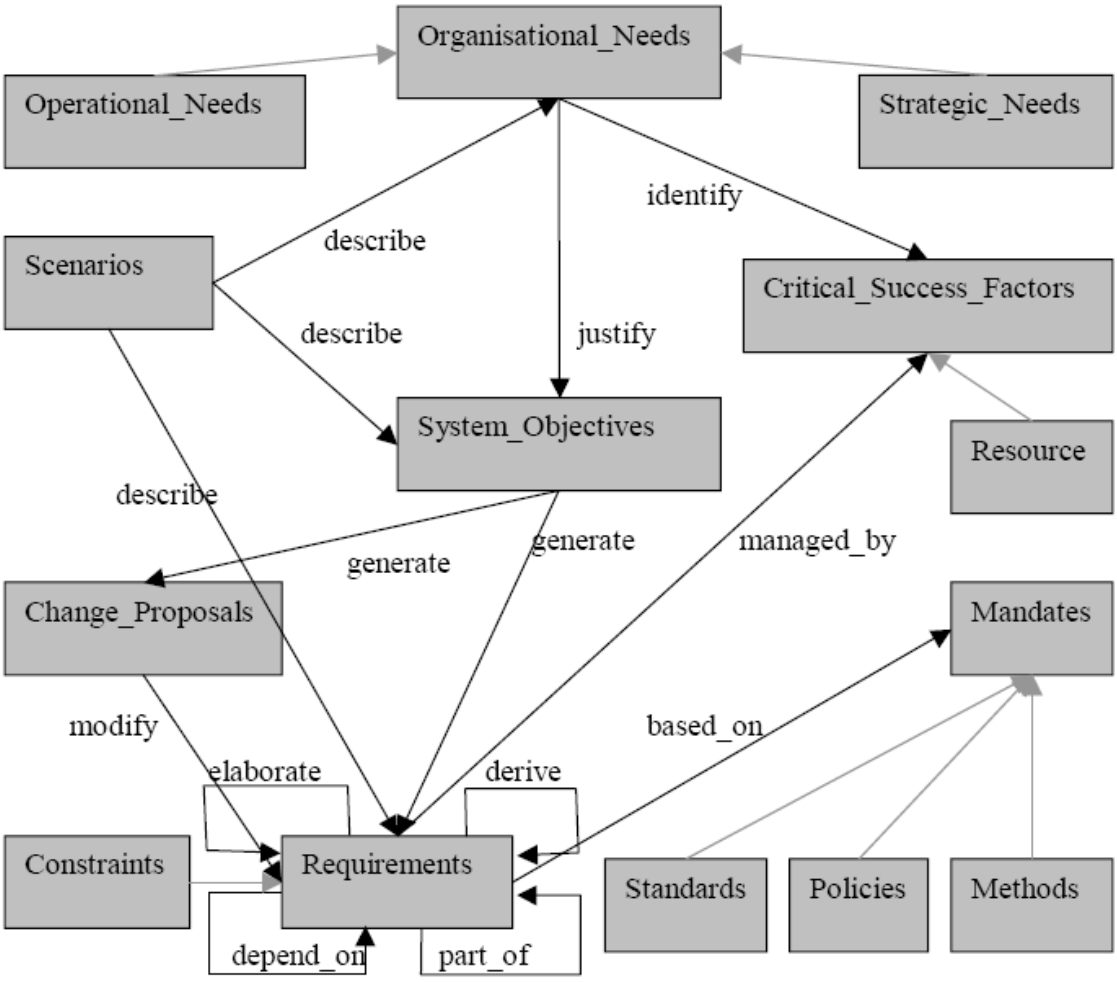


Figura 2.4 Modello di gestione dei requisiti

Il modello prende in considerazione i seguenti aspetti:

- i sistemi sono realizzati per soddisfare le *necessità organizzative* a lungo termine (strategiche) o immediate (operazionali), le quali sono dettagliate negli *scenari* che descrivono lo scopo o gli intenti d'uso del sistema;
- gli *obiettivi* associati al sistema e specificati dai diversi stakeholder sono giustificati dalle necessità organizzative. Essi costituiscono la base per la generazione dei *requisiti* del sistema;

- a causa dell'alto numero di requisiti associati ai sistemi complessi, gli stakeholder che hanno identificato le necessità organizzative devono anche identificare i *fattori critici di successo* (risorse come costi, tempo, peso, tensione, ecc.) sulla base dei quali potranno essere classificati per priorità e gestiti (monitorati, rintracciati e sottoposti a previsioni di bilancio) i requisiti;
- i *requisiti* possono essere basati su standard, politiche e metodi; inoltre, i vincoli possono essere trattati come dei requisiti (forse i più difficili da realizzare);
- occorre mantenere traccia dei *requisiti* che sono derivati da altri di più alto livello, che sono stati elaborati per introdurre chiarimenti sui meno comprensibili, che dipendono da altri o che formano parti di requisiti più complessi, in modo da gestire facilmente l'impatto dei *cambiamenti*.

2.2.6.2 Modello del razionale

Il modello del razionale, mostrato in Figura 2.5, mantiene le informazioni su come sono prese le decisioni per risolvere i conflitti durante tutto il ciclo di vita e per assicurare che i requisiti del cliente siano compresi e soddisfatti:

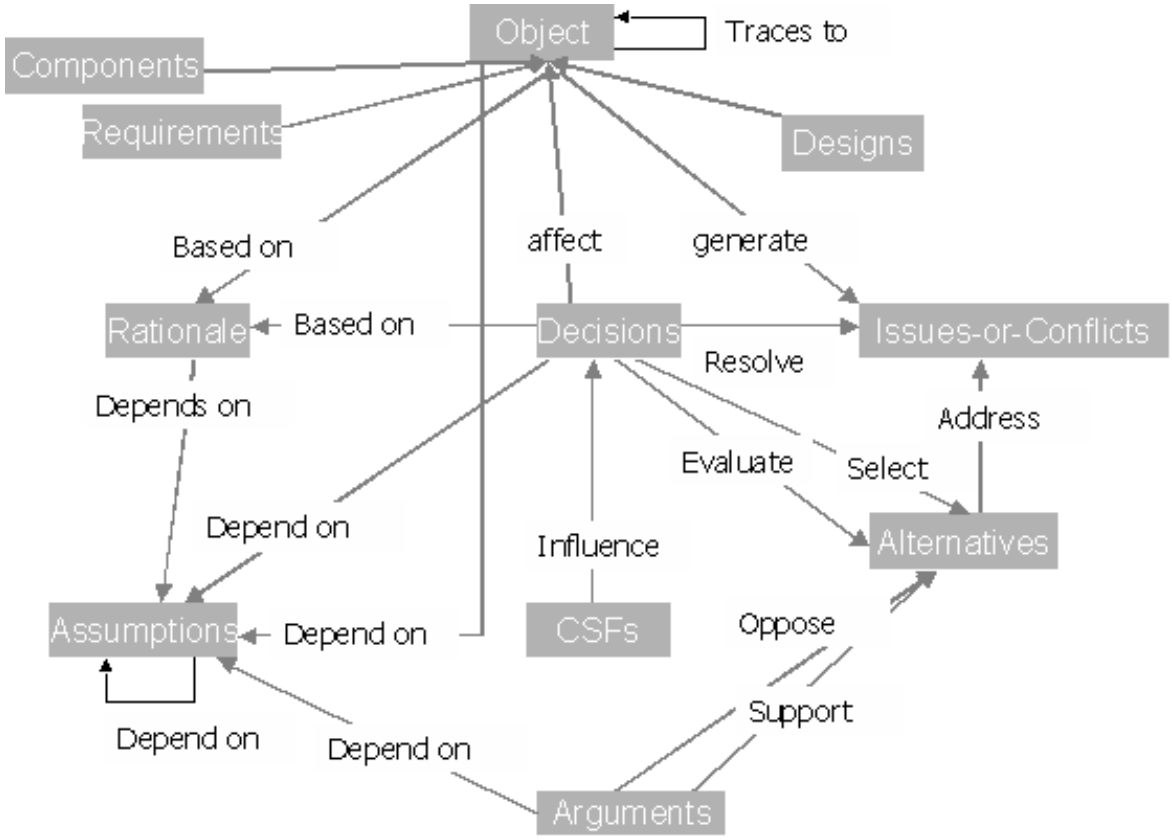


Figura 2.5 Modello del razionale

Esso prende in considerazione i seguenti aspetti:

- devono essere mantenute le informazioni sulle *decisioni* prese in risoluzione ai *conflitti* sugli *oggetti* che sorgono tra gli stakeholder e su come tali decisioni influenzano i requisiti;
- devono essere conservate le diverse *alternative* di risoluzione proposte e gli *argomenti* a favore o contrari (cosa che spesso non avviene soprattutto per le alternative scartate) per evitare di compiere nuovamente il lavoro in presenza di evoluzione dei requisiti;
- devono essere registrate le *assunzioni* su cui si basa il *razionale* delle decisioni;
- la decisione di selezionare una o più alternative è presa sulla base dei *fattori critici di successo* (*CSF*).

Spesso la cattura del razionale al livello di dettaglio sopra descritto è impraticabile a causa dell'overhead da introdurre e per la mancanza di un'adeguata strumentazione a supporto. Ad ogni modo, si possono registrare insieme alle assunzioni delle semplici descrizioni di razionale sul quale si basano i requisiti e i progetti. Un simile sforzo paga soprattutto quando si ha a che fare con progetti complessi e caratterizzati da requisiti in evoluzione e da un alto turnover di personale.

2.2.6.3 Modello di design/allocation

Il modello di design/allocation mostra le relazioni tra requisiti e componenti di progetto (Figura 2.6):

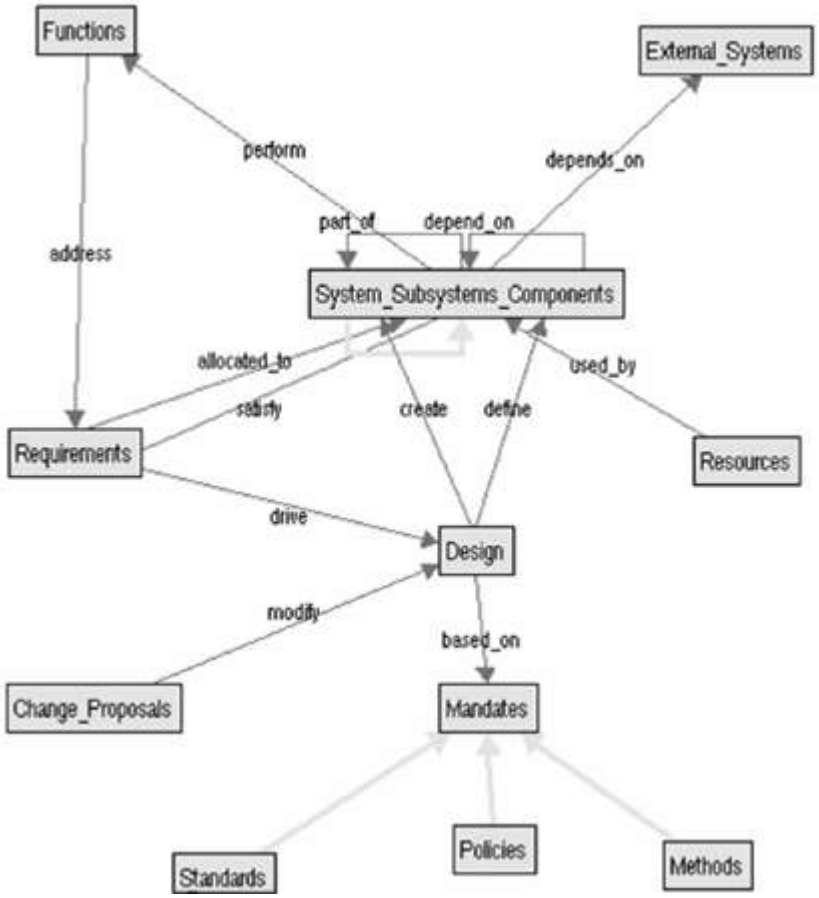


Figura 2.6 Modello di design/allocation

Tenendo presente che con il termine progettazione ci si riferisce in questo caso ad ogni attività che crea artefatti (compresa l'implementazione), tale sottomodello prende in considerazione i seguenti aspetti:

- i *requisiti* guidano la *progettazione*, che spesso è basata su *mandati* come standard, politiche o metodi che pilotano l'attività di sviluppo;
- i *sistemi*, i *sottosistemi* e i *componenti* sono i blocchi base per il sistema; essi sono definiti o creati dal processo di progettazione;
- i *requisiti* sono allocati ai *componenti* (hardware, software, risorse umane) che si suppone li soddisfino;

- i *componenti* possono dipendere da altri componenti (dato che le funzionalità o le prestazioni di uno possono dipendere da un altro) oppure essere parte di altri componenti;
- le *risorse* (denaro, peso, personale, potenza, ecc.) sono usate dai componenti. Le informazioni su allocazione, distribuzione e utilizzo sono importanti soprattutto quando le risorse sono dei CSF;
- occorre identificare come le *funzioni* sono realizzate dai componenti (anche se parzialmente) e come sono collegate ai requisiti identificati nei documenti dei requisiti. Tutti i requisiti devono essere soddisfatti man mano che la progettazione va avanti, anche se spesso è difficile creare collegamenti espliciti tra requisiti non funzionali e componenti del sistema;
- i *componenti* dipendono da *sistemi esterni* dato che spesso si devono interfacciare con questi.

2.2.6.4 Modello di verifica della compliance

Il sottomodello di verifica della compliance è mostrato in Figura 2.7:

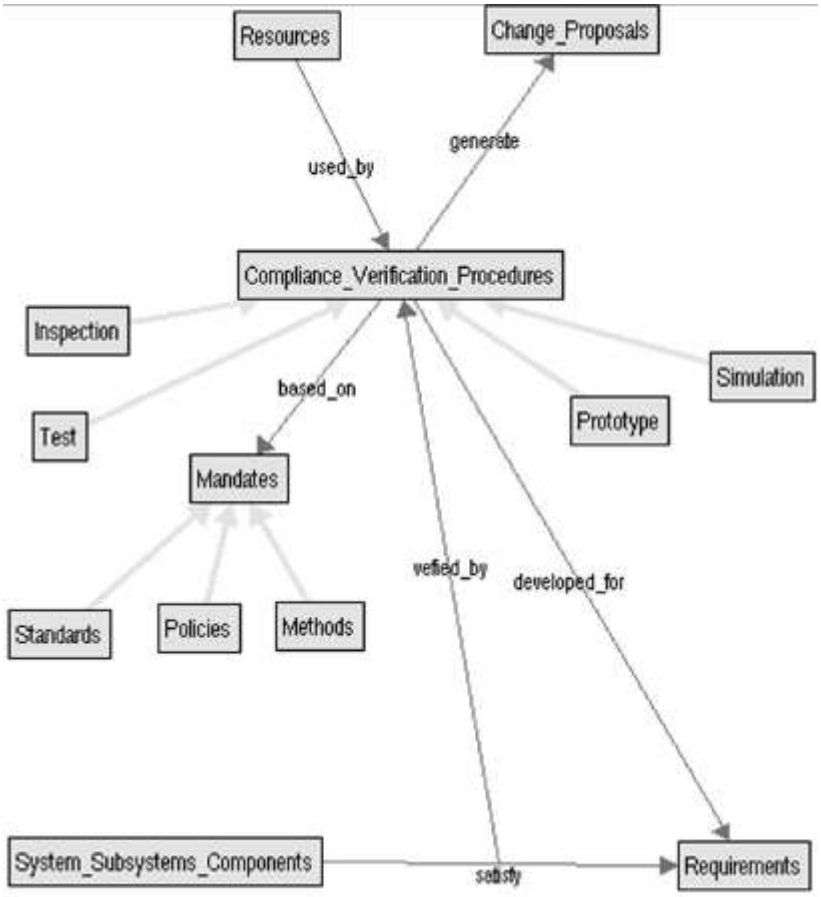


Figura 2.7 Modello di verifica della compliance

Esso è usato per certificare la completezza e la correttezza del sistema e per identificare i cambiamenti che potrebbero essere necessari per raggiungere gli obiettivi:

- per assicurare che ogni *requisito* è soddisfatto adeguatamente, o se un requisito può essere ancora considerato tale, vengono sviluppate diverse *CVP*, tenendo conto delle *risorse* disponibili;
- le *CVP* sono solitamente modellate dai *mandati*, i quali determinano quali procedure sono richieste e come devono essere realizzate;
- i risultati delle *CVP* producono risultati che possono sia indicare in che modo i *componenti* soddisfano i *requisiti* o aiutano a generare delle *proposte di cambiamento* per requisiti, progettazione o implementazione.

2.2.7 Tracciabilità dei requisiti ed information retrieval

Anche se l'importanza della tracciabilità dei requisiti a supporto delle attività di ingegneria del software critiche, come validazione dei requisiti e gestione del cambiamento, è ormai ampiamente riconosciuta, le organizzazioni spesso non possono far fronte ai costi aggiuntivi che una gestione manuale comporta. D'altra parte a volte ci si trova a dover ricostruire i collegamenti di tracciabilità in sistemi in cui tale pratica non è stata seguita.

Per risolvere questo tipo di problemi, molti ricercatori hanno valutato la possibilità di utilizzare metodi di recupero dinamico per la generazione automatica dei collegamenti di tracciabilità. Questi approcci si basano sui metodi di information retrieval per collegare tra di loro i vari artefatti prodotti durante i cicli di sviluppo e manutenzione sulla base dell'occorrenza dei termini. Molti di questi artefatti riguardano la documentazione del sistema software, la quale è quasi sempre espressa in modo informale, in linguaggio naturale e in testo libero. Esempi di tali artefatti includono la specifica dei requisiti, i documenti di progetto, le pagine di manuale, i log di errore e i relativi report di manutenzione. Una profonda trattazione dello stato dell'arte dell'information retrieval sarà tenuta a partire dal prossimo paragrafo.

I collegamenti così prodotti tra la documentazione con il codice sorgente aiutano gli analisti sia nella comprensione top-down (dalle specifiche al codice) che in quella bottom-up (dai frammenti di codice alle specifiche) del sistema software.

Una premessa di questo tipo di ricerche è che i programmatori usino nomi significativi per gli elementi del programma (funzioni, variabili, tipi, classi e metodi). Ciò è verosimile, dato che solitamente il programmatore mette in relazione i concetti di alto livello con quelli implementativi incapsulando negli identificatori la propria conoscenza del dominio

applicativo. In questo modo, è possibile utilizzare le tecniche di information retrieval per mettere in relazione aree di codice con i documenti.

I risultati ottenuti sono incoraggianti perché mostrano che il passaggio dai metodi tradizionali a quelli dinamici è fattibile, anche se questi ultimi soffrono ancora di problemi di precisione. In questo ambito è infatti importante per l'analista identificare tutti gli artefatti rilevanti, dato che anche una piccola percentuale di collegamenti non recuperati può portare a rendere inefficaci le analisi di impatto dei cambiamenti. Per questo motivo, le strategie di recupero devono favorire il *richiamo* piuttosto che la *precisione* (rif. Paragrafo 2.3.1.1), dove la prima misura il numero di documenti corretti recuperati sull'intero insieme di documenti corretti, e la seconda indica il numero di documenti corretti recuperati sul totale di quelli recuperati. L'uso di queste tecniche può ridurre drasticamente il lavoro che un'analista deve compiere per ripristinare le tracce: un lavoro manuale comporta la ricerca in centinaia di artefatti per trovare quelli interessati, mentre in questo caso la ricerca deve essere effettuata solo su una frazione di tali documenti.

2.3 Information retrieval

Il problema della memorizzazione e del recupero delle informazioni è diventato rilevante a partire dagli anni quaranta. Tale problema consiste nell'avere a disposizione una mole enorme di informazioni alla quale si ha bisogno di avere accesso in maniera accurata e veloce, e la difficoltà di ottenere ciò porta ad ignorare delle informazioni rilevanti e quindi a dover ripetere tali operazioni più volte ed in maniere differenti. Con l'avvento degli elaboratori elettronici si è pensato di sfruttare le loro capacità per ottenere sistemi di

recupero rapidi ed intelligenti, ma tranne che per scopi di catalogazione e gestione generale il loro uso non ha risolto efficacemente tale problema.

In linea di principio la memorizzazione ed il recupero delle informazioni è semplice. Supponendo di avere un archivio di documenti ed una persona (utente) con una domanda (richiesta o query), questi può ottenere un insieme di documenti che soddisfano il suo bisogno informativo leggendo tutti i documenti nell'archivio mantenendo quelli rilevanti e scartando tutti gli altri. Ciò costituirebbe il recupero perfetto, ma è ovviamente impraticabile in termini di tempo.

Quando furono disponibili elaboratori ad alte prestazioni per scopi generici si pensò che questi potessero 'leggere' un'intera collezione di documenti per estrarvi quelli rilevanti, ma presto divenne chiaro che usare il testo in linguaggio naturale causa problemi di input e memorizzazione e non risolve il problema della caratterizzazione intelligibile del contenuto dei documenti. Può anche darsi che i futuri miglioramenti dell'hardware rendano le operazioni pratiche sul linguaggio naturale più fattibili, ma la caratterizzazione automatica in termini del processo umano di 'lettura' resta un problema molto pernicioso. Nello specifico, 'leggere' significa provare ad estrarre le informazioni, sia sintattiche che semantiche, dal testo ed usarle per decidere quali documenti sono rilevanti per una particolare richiesta. La difficoltà non sta solo nel sapere come estrarre le informazioni, ma anche nel come usarle per decidere la rilevanza dei documenti. Il lento progresso della linguistica sul fronte semantico e gli scarsi risultati della traduzione automatica hanno mostrato che il problema è ancora insoluto; tuttavia è sembrato utile riportare nella seguente trattazione anche concetti e risultati derivati dal campo della linguistica, per sottolineare come sia poliedrico il mondo dell'informazione e del suo utilizzo.

Il punto cardine dell'information retrieval è il concetto di 'rilevanza'. Lo scopo di una strategia di recupero automatica è quello di recuperare tutti i documenti *rilevanti*

recuperando allo stesso tempo il minor numero di quelli *non-rilevanti*. Per far ciò, la caratterizzazione di un documento elaborato dovrebbe essere tale da renderlo recuperabile nel caso questo sia rilevante per una query. Gli indicizzatori umani caratterizzano i documenti assegnando loro dei termini indice scelti cercando di prevedere quelli che l'utente impiegherebbe per descrivere l'argomento desiderato, nell'ipotesi che quest'ultimo sia l'argomento del documento che si sta caratterizzando. Tutto questo equivale a costruire implicitamente delle query per gli argomenti dei documenti. Quando si indicizza in maniera automatica si assume che analizzando il testo di un documento o di una query tramite la stessa analisi automatica, il risultato sarà una rappresentazione del contenuto.

Per una persona è possibile stabilire la rilevanza di un documento per una query. Per rendere possibile questo anche per un elaboratore bisogna costruire un modello in cui si possano quantificare le decisioni sulla rilevanza. La maggior parte della ricerca nel campo dell'information retrieval tratta i diversi aspetti di tale modello.

Un semplice sistema di information retrieval

Di seguito è illustrato lo schema di un sistema di information retrieval (Figura 2.8):

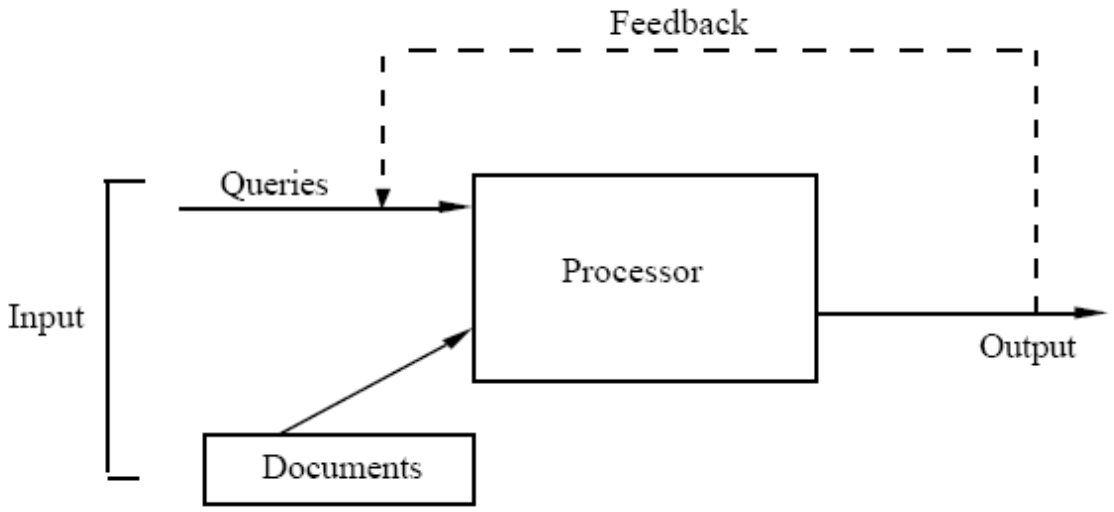


Figura 2.8 Un tipico sistema IR

Si possono individuare tre componenti: ingresso, processore ed uscita.

Per quanto riguarda l'ingresso, il problema è quello di ottenere una rappresentazione dei documenti e delle query adatta al trattamento da parte dell'elaboratore. La maggior parte dei sistemi di information retrieval memorizza solo una rappresentazione del documento o della query, quindi il loro testo viene perso dopo l'elaborazione per la generazione di tale rappresentazione. Un rappresentante del documento potrebbe ad esempio essere una lista di parole estratte considerate significative. Invece di far elaborare alle macchine dati in linguaggio naturale, un approccio alternativo è quello di formulare tutte le query ed i documenti in un linguaggio artificiale appositamente studiato. Ovviamente l'utente deve essere in grado di poter esprimere i suoi bisogni informativi in tale linguaggio.

Successivamente vi è il processore, la parte del sistema che tratta il processo di recupero. Esso può dover strutturare le informazioni in maniera appropriata, ad esempio classificandole. Esso applicherà anche la funzione di recupero, cioè la strategia di ricerca in risposta ad una query. Nel diagramma i documenti sono stati considerati un'entità a parte e posti in un contenitore per evidenziare il fatto che essi non sono solo degli ingressi,

ma possono essere usati durante il processo di recupero in modo tale che la loro struttura sia vista più correttamente come parte di tale processo.

Infine vi sono le uscite, rappresentate di solito da un insieme di citazioni o di numeri assegnati in base all'ordinamento dei documenti. Questo è tutto ciò che è presente in un sistema ufficiale, mentre in un sistema sperimentale bisogna ancora applicare un metodo di valutazione.

L'information retrieval in prospettive diverse

Più in dettaglio, lo schema di funzionamento di un tipico sistema di information retrieval è il seguente (Figura 2.9):

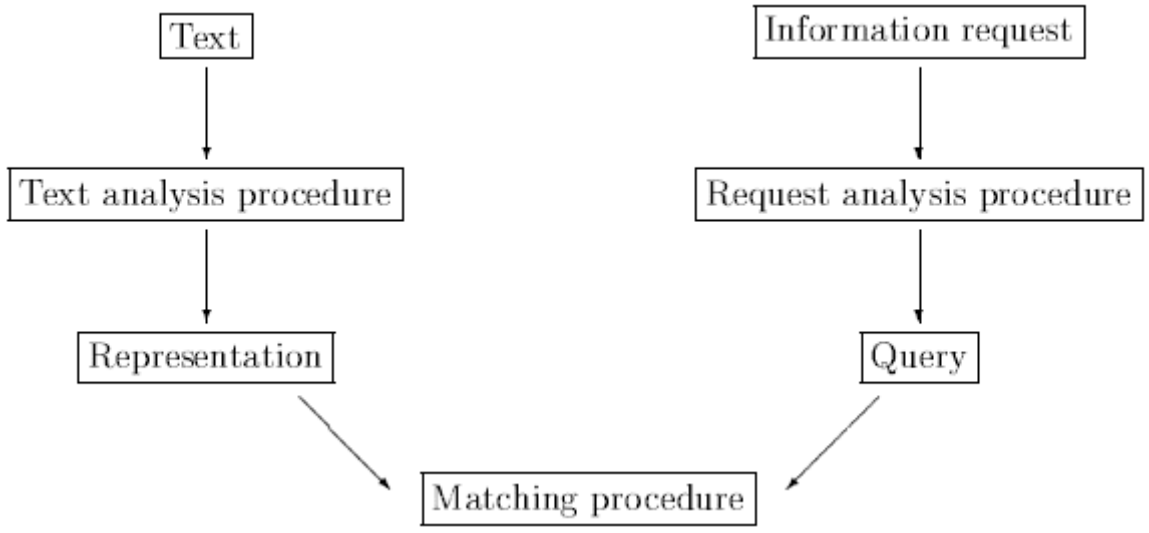


Figura 2.9 Schema di funzionamento di un sistema di information retrieval

Vi è un insieme di testi; le richieste informative sono inserite in un sistema che tratta questo insieme di testi; i testi sono analizzati tramite una qualche forma di procedura di analisi per ottenerne una rappresentazione non testuale; le richieste informative sono a loro volta analizzate tramite una procedura uguale o simile per ottenere una query. Le due

rappresentazioni ottenute vengono poi raffrontate. I testi con i confronti più soddisfacenti sono presentati come potenziali sorgenti di informazione per soddisfare la richiesta.

Solo una piccola parte di questo processo è basata sull'esplicita conoscenza del linguaggio, infatti di norma sia le procedure di analisi che la procedura di confronto sono eseguite usando metodi statistici. Il ruolo della lingua e della conoscenza del linguaggio è di solito quello di migliorare l'analisi delle richieste e dei testi, assumendo ad ogni modo che le rappresentazioni siano in qualche modo alinguistiche e dipendenti da una manipolazione puramente formale. Lo studio delle operazioni di analisi è normalmente portato a ridurre la quantità di informazioni per rendere gestibile la rappresentazione, e ad eliminare l'indeterminazione del linguaggio naturale per facilitare il confronto.

Questo modello abbastanza intuitivo e per molti versi attraente nasconde la complessità dell'uso del linguaggio naturale tramite la procedura di confronto, e tale complessità può quindi essere risolta usando metodi formali. Ciò però non apporta solo benefici. I meccanismi che rendono complicato il confronto – l'indeterminazione del linguaggio naturale – sono ciò che fanno del linguaggio naturale uno strumento di comunicazione; la consapevolezza di ciò è tipicamente riassunta dal processo di ricerca. La differenza principale tra l'utilizzo di un sistema di information retrieval e la consulenza di un analista informativo umano è che quest'ultimo di norma non richiede che la richiesta sia trasformata in una qualche rappresentazione fissa e non ambigua, né richiede che i documenti da analizzare siano in tale rappresentazione. Egli non solo è in grado di gestire ma utilizza la flessibilità delle informazioni nel linguaggio naturale: quest'ultimo non è un ostacolo ma una caratteristica. Un testo apparentemente non correlato (secondo il giudizio di metodi formali) può in realtà contenere degli importanti punti in comune con una determinata richiesta.

Secondo l'approccio formale invece, a conferma di quanto detto nel paragrafo precedente il fine ultimo dei sistemi considerati è di poter applicare proficuamente una strategia di recupero a partire da opportune rappresentazioni dei documenti e delle richieste dell'utente.

Sebbene l'information retrieval si possa dividere in varie componenti, ne sono state identificate tre con le quali è possibile costruire una parte cospicua del sistema: l'analisi del contenuto, le strutture informative, e la valutazione. Brevemente, la prima riguarda la descrizione dei contenuti dei documenti in una forma adatta all'elaborazione; la seconda verte sulla scoperta delle relazioni tra i documenti per migliorare l'efficienza e l'efficacia delle strategie di recupero; la terza la misura dell'efficacia del recupero.

Dal punto di vista della rappresentazione dei documenti, è utile riportare l'approccio seguito da Luhn. Egli usò le frequenze di occorrenza delle parole nel testo del documento per determinare quali di esse fossero sufficientemente significative per rappresentare o caratterizzare il documento in forma digitale: quindi veniva creata per ogni documento una lista di una qualche sorta di 'parole chiave'. In più tali frequenze potevano essere usate anche per indicare un grado di significatività. Ciò ha fornito un semplice schema per pesare le parole chiave in ogni lista e rendere disponibile un rappresentante del documento nella forma di una 'descrizione a parole chiave pesate'.

Il punto cruciale risulta allora essere quello delle parole chiave. Nella letteratura sull'information retrieval ci si riferisce spesso agli oggetti descrittivi estratti dal testo chiamandoli *parole chiave* o *termini*. Tali oggetti sono spesso il frutto di alcuni processi, come ad esempio il raggruppamento di varianti morfologiche diverse della stessa parola.

Il termine *struttura informativa* si riferisce ad un'organizzazione logica delle informazioni, come i rappresentanti dei documenti, mirata all'information retrieval. Lo sviluppo di tali strutture è avvenuto di recente, e la lentezza di ciò è stata dovuta principalmente al fatto

che per molto tempo non si è compreso che nessun elaboratore è in grado di fornire un recupero di informazioni su un grande insieme di documenti in un tempo ragionevole senza imporgli qualche struttura logica. Allo stesso modo lo sviluppo e l'uso di nuove tecniche è stato rallentato dalla mancanza di prove sperimentali di supporto. I primi esperimenti sui sistemi di document retrieval adottavano solitamente un'organizzazione a file seriale che, sebbene fosse efficiente nel caso di un'elaborazione batch di un cospicuo numero di query, si è mostrata inadeguata nel caso di query vincolate a tempi di risposta brevi. L'organizzazione comunemente riconosciuta è invece quella a file invertito. Entrambe le organizzazioni saranno trattate più avanti.

I file sono organizzati in cluster tramite un metodo di classificazione automatica. Fino a pochi anni fa sono stati condotti diversi esperimenti sull'argomento ma su piccola scala, per cui non è possibile valutarne fino in fondo la bontà dato che quest'ultimo si esplica solo su scala abbastanza elevata.

Si è constatato che la valutazione dei sistemi di recupero è estremamente difficoltosa. In passato ci sono stati accesi confronti sulla validità delle valutazioni tramite giudizi sulla rilevanza da parte di persone fisiche. Attualmente l'efficacia di un'operazione di recupero è ancora principalmente misurata in termini di *precisione* (la frazione dei documenti recuperati che sono rilevanti) e *richiamo* (la frazione dei documenti rilevanti che sono stati recuperati), o per mezzo di misure basate su essi. Dato che questi termini propri del concetto di valutazione sono a volte citati nel corso del testo, si ritiene utile riportare di seguito la trattazione sulla valutazione.

2.3.1 Valutazione dell'information retrieval

Gli algoritmi di information retrieval aderiscono ad un modello ben formalizzato sui loro usi e sui loro benefici, e ciò risulta molto utile alla valutazione dell'utilità del sistema. La ricerca sull'information retrieval ha sviluppato un insieme ben definito di strumenti di valutazione, che sono basati sui concetti di *precisione* e *richiamo*. Nella figura seguente è riportato lo schema di funzionamento della Figura 2.10, con l'aggiunta in basso di una *lista dei risultati* del processo, cioè di una lista di documenti potenzialmente utili che sono presentati al lettore in qualche maniera.

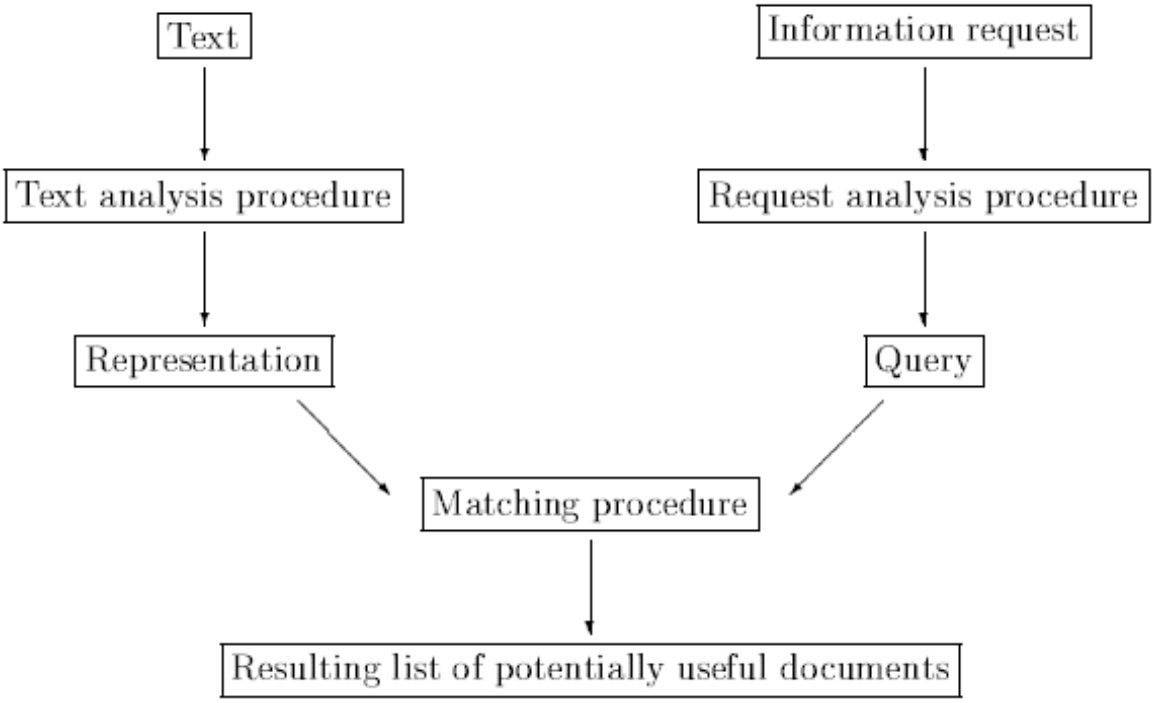


Figura 2.10 Visione completa dello schema funzionale

La precisione ed il richiamo sono misurati esaminando quanti documenti rilevanti sono presenti nell'insieme recuperato.

2.3.1.1 Richiamo e precisione

Richiamo

Se si ha una buona stima del numero di documenti rilevanti contenuti in una base di documenti per una data richiesta di informazioni, si può trarre un dato importante in base alla quantità di documenti rilevanti trovati e recuperati dall'algoritmo utilizzato. Il rapporto tra il numero di documenti rilevanti recuperati ed il numero di documenti rilevanti non-recuperati è chiamato *richiamo*.

Precisione

L'insieme recuperato spesso contiene sia documenti rilevanti che non-rilevanti. Il rapporto tra il numero di documenti rilevanti ed il numero di documenti non-rilevanti in un insieme recuperato è chiamato *precisione*.

Combinazione dei due concetti

Banalmente, se un algoritmo recupera sempre tutti i documenti in una base di documenti, esso ha un richiamo del cento per cento. Comunque, presumibilmente esso avrà una bassa precisione. In questo senso, la precisione ed il richiamo variano in maniera inversamente proporzionale. In molte valutazioni, la precisione è misurata su un numero fisso di documenti recuperati. In altri casi, il richiamo e la precisione sono confrontate l'una con l'altra, calcolando la precisione relativa ad un certo livello di richiamo. Nelle valutazioni del TREC è utilizzata una misura media su "11 punti", con una precisione misurata su ogni dieci per cento di richiamo: richiamo al dieci per cento, richiamo al venti per cento, e così via fino ad un richiamo al cento per cento, in cui si assume che tutti i documenti rilevanti

siano stati recuperati. La precisione media a tutti questi punti di richiamo è poi utilizzata come misura totale.

Efficacia ed efficienza

La ricerca e lo sviluppo nell'information retrieval è tesa a migliorare l'efficacia e l'efficienza del recupero. L'efficienza è misurata in termini di risorse di elaborazione usate, come potenza e tempo di CPU, capacità di memoria, ecc. È difficile misurare l'efficienza in maniera indipendente dalla particolare macchina utilizzata, ma in ogni modo essa sarebbe misurata in congiunzione con l'efficacia per avere un'idea dei benefici ottenuti. Prima è stato detto che l'efficacia si misura in termini di precisione e richiamo: tali entità sono molto importanti proprio perché si fa molto spesso riferimento all'efficacia nella valutazione.

2.3.1.2 Grado di bontà delle misure di valutazione

Prese come misure di valutazione, la precisione ed il richiamo hanno diverse qualità appetibili. Sono intuitivamente valide e possono essere empiricamente determinate in modo da essere affidabili. Comunque, esse risentono di alcuni svantaggi non trascurabili. Per i calcoli richiesti il valutatore dovrebbe sapere quanti sono i documenti rilevanti (oracolo), quanti sono i documenti nella base di documenti, quanti documenti sono stati recuperati, come pesare la rilevanza di un documento per il calcolo della precisione, come determinare cosa vuole esprimere una query, e come dare un giudizio sulla rilevanza. Tutti questi punti possono essere affrontati, ma a rischio di creare una valutazione troppo mirata ed in sé stessa irrilevante; in parole povere si può affermare che gli spazi delle

informazioni raramente hanno una struttura abbastanza semplice da poter essere mappata su questo tipo di spazio di valutazione prototipale. Di seguito si riportano degli aspetti del concetto di valutazione che possono avere risoluzioni differenti in base al particolare problema e contesto considerato.

Campionamento

In generale, non si ha un quadro chiaro di quanti documenti siano rilevanti in una data base di documenti. A meno che non si abbia a disposizione una piccola base di dati sperimentale di cui si possegga il controllo completo, bisogna ricorrere a procedure di campionamento.

Universo

A volte non è possibile determinare ”l’intera base di documenti”: un esempio può essere fornito dal recupero attraverso Internet, caratterizzato da una base di dati molto vasta ed eterogenea per via dei numerosi standard utilizzati.

Iterazione

Una query è ben definita dal punto di vista sperimentale, ma la sua controparte nel mondo reale non lo è. Gli utenti spesso non riescono ad individuare dei termini di ricerca adatti ad esprimere le loro necessità informative in modo da recuperare i documenti rilevanti al primo tentativo di ricerca, ma effettuano un certo numero di iterazioni finché i pochi oggetti visibili in cima all’insieme recuperato sembrano soddisfacenti. Ciò genera il problema di decidere quando effettuare la valutazione del sistema (ad ogni query o alla fine del processo iterativo).

Recupero

Nella maggior parte dei sistemi di punteggio probabilistici l'insieme recuperato non viene delimitato. Un insieme di diverse migliaia di documenti probabilmente non sarà molto utile, quindi si avrà bisogno di decidere quanti documenti assumere come recuperati.

Precisione e richiamo

Come detto precedentemente, è una pratica comune mediare gli andamenti richiamo-precisione, ad esempio in medie su 11 punti, ma ciò ha la conseguenza indesiderata di mascherare le differenze tra algoritmi che danno buone prestazioni solo in ricerche ad alta precisione, solo nei casi in cui ci sono pochissimi documenti da trovare, o solo quando la base di documenti è satura di materiale sull'argomento.

Rilevanza e necessità di informazione

Per alcuni *task* il concetto di rilevanza è molto differente in base alle assunzioni fatte su precisione e richiamo. Un utente può lavorare su un caso giudiziario o su un'applicazione brevettata, e deve quindi avere una lista esaustiva di tutti i documenti presenti nella base di dati. Per questo task, il richiamo è un fattore importante. Ma un altro utente può aver bisogno semplicemente di una risposta ad una domanda, nel qual caso il primo documento estratto può soddisfare l'intero bisogno, e non interessa il resto della collezione. Spesso non è possibile nemmeno definire una data necessità di informazioni. Se si esamina una collezione di foto di famiglia, il bisogno informativo è soddisfatto quando tutte le foto sono state visionate o sorge qualche altro task più interessante. Un sistema per organizzare e recuperare le foto di famiglia non sarà facile da valutare in termini di richiamo e precisione.

Ci sono caratteristiche dei documenti che rendono il giudizio sulla rilevanza molto poco chiaro. Prima di tutto, i documenti possono essere di qualità differente. Ad esempio su Internet si possono trovare liberamente bollettini legali e medici, senza alcuna garanzia di qualità: alcune informazioni possono essere errate, o anche appositamente fuorvianti. In tutti questi casi, la qualità non può essere facilmente inserita nella distinzione binaria della rilevanza. In secondo luogo, può esservi la possibilità che alcuni documenti sostituiscano documenti più vecchi. Una prima versione di un manuale non ha alcuna rilevanza in una query inerente al servizio di supporto, se ne esiste una versione più recente: la sua refutazione può diminuire la rilevanza di un documento da esso riferito. Non si riesce a tenere conto facilmente degli aspetti temporali delle collezioni di documenti per effettuare distinzioni di rilevanza.

Quindi, in conclusione, mentre la precisione ed il richiamo sono concetti sperimentali molto utili per testare algoritmi e confrontarli l'un l'altro, è meno chiara la loro utilità per misurare il successo che un sistema può avere nella pratica in caso di basi di dati dinamiche.

2.3.2 Analisi automatica del testo

Le informazioni sono fornite tipicamente sottoforma di documenti, e perché tali documenti possano essere trattati si ha bisogno di memorizzarli nell'elaboratore. Solitamente però non è una buona idea memorizzare il testo completo in linguaggio naturale, bensì un suo rappresentante che può essere stato prodotto manualmente o automaticamente, partendo dal testo completo oppure dal suo prologo o ancora dal testo.

Di seguito si tratterà degli approcci statistici all'analisi automatica del testo, mentre sarà dato un accenno a quelli linguistici, perché questi ultimi si sono rivelati costosi da implementare e non è chiaro come usarli per migliorare l'information retrieval (in parte dovuto al fatto che ci sono stati pochissimi progressi nella teoria della semantica formale). In un'area di ricerca che principalmente tratta testo ci sarebbero molte situazioni in cui i risultati delle ricerche linguistiche potrebbero essere proficuamente applicati, e numerose domande che potrebbero essere tranquillamente poste ai linguisti per ulteriori studi, ma queste ipotesi non sono utilizzate praticamente.

Si partirà con l'analisi del testo ideata da Luhn, prendendo in considerazione anche diversi aspetti dal punto di vista linguistico. Successivamente si descriverà una maniera concreta per generare rappresentanti dei documenti, ed infine si cercherà di migliorare tali rappresentanti attraverso l'attribuzione di pesi o la classificazione delle parole chiave.

2.3.2.1 Le idee di Luhn

Una citazione da un articolo di Luhn afferma: 'La frequenza di occorrenza delle parole in un testo fornisce una misura utile della significatività delle parole. Inoltre la posizione relativa in una frase delle parole aventi certi valori di significatività fornisce un'utile misura per la significatività delle frasi. Il fattore di significatività di una frase sarà determinato da una combinazione di queste due misure.' Secondo la sua teoria, quindi, si possono usare i dati sulla frequenza per estrarre parole e frasi in modo da rappresentare un documento.

Sia f la frequenza di occorrenza di vari tipi di parole in una certa posizione del testo e r il loro punteggio secondo un certo ordine (cioè l'ordine delle loro frequenze di occorrenza);

allora in un grafico avente come assi *f* e *r* si ritrova una curva simile alla curva iperbolica riportata in figura 2.11.

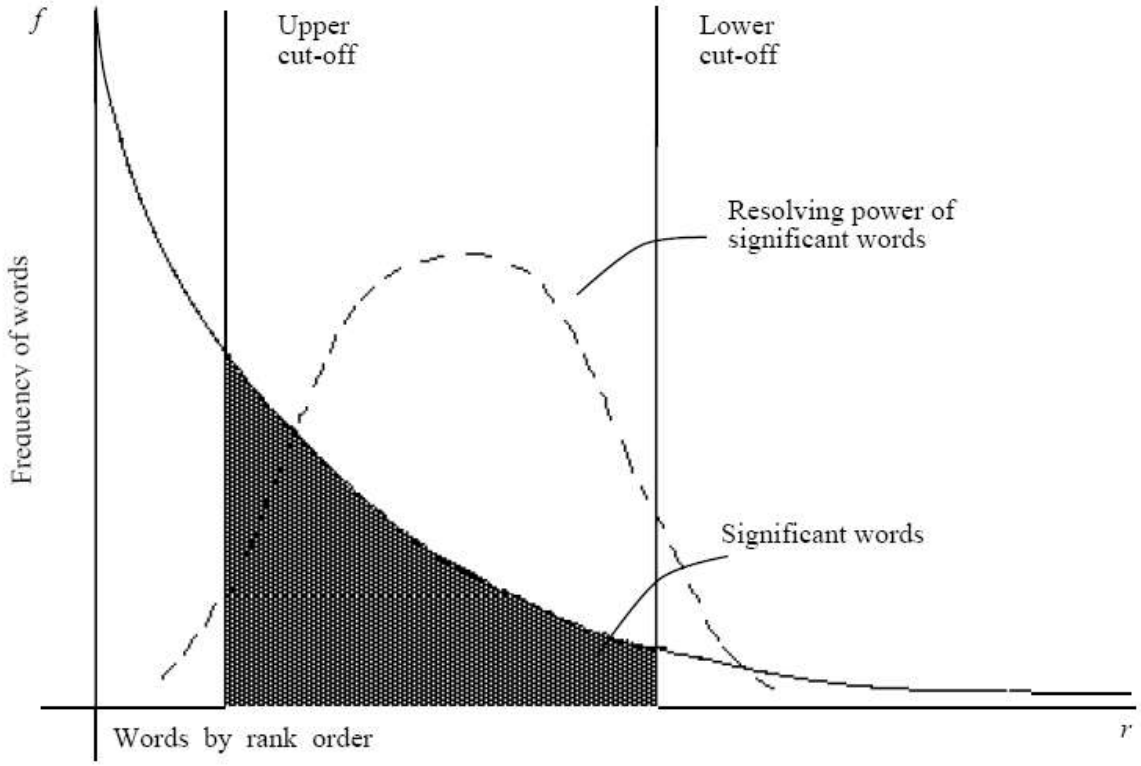


Figura 2.11 Un grafico della curva iperbolica che lega la frequenza di occorrenza f e l'ordine di punteggio r

Questa curva dimostra la Legge di Zipf, la quale afferma che il prodotto tra la frequenza di occorrenza delle parole (molto spesso indicata con *tf*) e l'ordine di punteggio è approssimativamente costante. Luhn utilizzò questo concetto come ipotesi nulla per specificare due soglie (cut-off), una superiore ed una inferiore. Le parole al di sopra della soglia superiore sono considerate comuni e quelle al di sotto sono considerate rare, quindi non contribuiscono significativamente al contenuto del testo. Quindi Luhn definì una tecnica di conteggio per la rilevazione delle parole significative. In armonia con la sua

definizione, si nota che l'autorevolezza delle parole significative raggiunge un picco massimo in corrispondenza della posizione (relativa all'ordine di punteggio) a metà tra le due soglie, e da lì decresce fino quasi ad annullarsi nei punti di soglia. Le soglie non possono essere stabilite deterministicamente, ma tramite tentativi e valutando gli errori, per cui sarà insita sempre una certa arbitrarietà. Queste idee sono alla base di buona parte dei lavori sviluppati nel campo dell'information retrieval.

Non è necessario che un'analisi del genere sia ristretta unicamente alle parole. Si potrebbe applicarla tranquillamente alle radici delle parole o alle frasi, così come infatti spesso si fa.

2.3.2.2 Rappresentanti dei documenti

Come è stato detto precedentemente, sarebbe auspicabile sviluppare un sistema di elaborazione del testo in grado di generare dal testo in ingresso, tramite il minor intervento umano possibile, un rappresentante del documento idoneo al trattamento da parte di un sistema di recupero automatico, ma ciò è fattibile solo in parte. Un rappresentante di un documento consiste semplicemente in una lista di nomi di classi di appartenenza, in cui ogni classe è costituita da un insieme di parole che compaiono nell'intero testo in ingresso. In pratica un testo viene indicizzato con un nome se una delle sue parole significative compare come membro della classe relativa a tale nome.

Un sistema del genere compirà tre operazioni macroscopiche:

1. rimozione delle parole troppo comuni;
2. eliminazione dei suffissi;
3. rilevazione di radici equivalenti.

La rimozione delle parole troppo comuni o delle *stopword* equivale ad implementare la soglia superiore di Luhn. Ciò si realizza confrontando il testo in ingresso con una *stoplist* di parole che saranno rimosse. I vantaggi di tale processo sono l'eliminazione delle potenziali interferenze causate dalle parole non significative alle operazioni di recupero, e la riduzione della dimensione totale del documento di un'aliquota dal trenta al cinquanta per cento.

L'eliminazione dei suffissi è più complicata, e normalmente si realizza utilizzando una lista completa di questi e rimuovendo quelli più lunghi possibile. Il problema è che la rimozione indipendente dal contesto può portare ad un alto tasso d'errore, come ad esempio la rimozione del suffisso ALE appropriata per COMMERCIALE ma non per UGUALE. Per evitare ciò sono state sviluppate regole di eliminazione unicamente in contesti corretti.. 'Corretto' potrebbe significare che:

- ✓ la lunghezza della radice rimanente abbia più di un certo numero di caratteri;
- ✓ la parte finale della radice soddisfi una certa condizione;
- ✓ ecc.

Al termine dell'operazione molte parole tra loro equivalenti si ritrovano portate in un'unica forma morfologica, mentre altre avranno forme diverse pur essendo equivalenti perché continuano ad essere differenti le parti terminali delle radici. Questi ultimi tipi di parole devono essere trattate in maniera speciale, cercando di riportarle alla radice comune. L'assunzione rimane comunque quella che radici uguali si riferiscano a concetti uguali, mentre si capisce bene che ciò non è sempre vero, per via sia di parole diverse che hanno radici uguali (come NEUTRONE e NEUTRALIZZARE) sia di parole uguali con significati diversi in contesti diversi (come PESCA). È chiaro quindi che il processo di eliminazione dei suffissi introdurrà sempre un certo tasso di errore, ma gli esperimenti hanno dimostrato che questo tende ad essere dell'ordine del cinque per cento.

È da notare che il limite di questo tipo di algoritmo non è la potenza di calcolo ma il tempo di elaborazione.

Il risultato dell'algoritmo di fusione è un insieme di classi aventi per nome la radice estratta. Un rappresentante di un documento sarà pertanto una lista di radici delle parole in esso contenute, e tali radici sono spesso chiamate *termini indice* o *parole chiave*.

L'*analisi morfologica* per l'identificazione delle varianti morfologiche di un lessico è normalmente implementata come stemming o eliminazione dei suffissi semplici. Porter descrisse un algoritmo di stemming per l'inglese sensibile al contesto, efficiente e largamente adottato basato su una lista di suffissi. Alternativamente si può incoraggiare l'utente a non inserire parole complete ma direttamente le relative forme troncate.

L'utilità dello stemming per l'inglese è discutibile, ma i vantaggi intuitivi sono interessanti ed il costo computazionale abbastanza basso, ed infatti molti sistemi sono orientati in questa direzione. L'inglese, naturalmente, ha una morfologia eccezionalmente sobria con poche varianti e non tende a generare mescolanze grafiche frequentemente come le altre lingue: entrambe queste caratteristiche sembrerebbero diminuire l'utilità di un'elaborata analisi morfologica per questa lingua.

Non è possibile sfruttare appieno la generalità dei risultati nel campo dato che la lingua mondiale per la ricerca e gli affari attuale è l'inglese. Lo stemming semplice è sufficiente per essa, ma non per la maggior parte delle altre lingue del mondo. Esperimenti condotti sull'analisi morfologica, basati sulla normalizzazione di materiale in lingue diverse dall'inglese, hanno fornito risultati migliori nelle altre lingue: cosa è utile in un testo dipende dalla lingua.

Il processo visto dovrà ovviamente essere applicato anche alle query.

2.3.2.3 Conoscenza del linguaggio

Se si prova a determinare i termini significativi in un documento per rappresentarne il contenuto, si scopre che quelli comuni in un documento, ma anche in tutti gli altri, sono meno utili degli altri termini. La questione vertirà allora sulla *specificità* di un termine per un documento, o su quanto esso sia non comune per gli altri documenti.

La *frequenza della collezione*, o *frequenza del documento inversa* (*idf* per brevità), è una misura della specificità dei termini originariamente definita da Karen Sparck Jones. L'idf è una funzione di K/d_i dove K è una qualche costante, tipicamente dipendente da N (il numero totale di documenti in una collezione), e d_i è il numero di documenti in cui compare il termine i–esimo (la frequenza nei documenti). Questa misura dà valore maggiore ai termini che compaiono solo in pochi documenti. Usato singolarmente, tale indice è però utile quanto la singola frequenza dei termini.

Ci sono diverse modifiche apportabili alla misura dell'*idf*. Invece dei documenti, si possono usare come unità i paragrafi, per modellare il fatto che i primi possono essere non omogenei; oppure si può pesare la misura in maniere differenti in base alle proprietà del documento (ad esempio in base alla sua frequenza all'interno di un documento).

Un potenziale problema con l'*idf* come misura è la definizione dell'universo su cui calcolare la frequenza del documento. Tale calcolo presuppone una vista d'insieme di tutti i documenti nella collezione, e stabilire quale sia l'uso generale di un termine può essere difficoltoso, se non impossibile. In alcuni casi una collezione è così ben definita che un'*idf* limitata ai documenti in essa contenuti è abbastanza adeguata, mentre in altri, in cui i termini possono riferirsi ad argomenti diversi, può non esserlo.

2.3.2.4 Indicizzazione

Un linguaggio indice è il linguaggio usato per descrivere i documenti e le richieste. I suoi elementi sono i termini indice, derivati dal testo del documento o forniti in modo indipendente. I linguaggi indice possono essere *pre-coordinati* o *post-coordinati*: nel primo caso i termini vengono coordinati al momento dell'indicizzazione, mentre nel secondo caso al momento della ricerca. Più precisamente, nell'indicizzazione pre-coordinata può essere utilizzata una combinazione logica dei termini indice come etichetta per identificare una classe di documenti, mentre nell'indicizzazione post-coordinata tale classe sarà identificata al momento della ricerca combinando le classi dei documenti etichettati con i singoli termini indice.

Un'altra distinzione riguarda il vocabolario di un linguaggio indice, che può essere *controllato* o *non controllato*: il primo si riferisce ad una lista di termini indice approvati che possono essere usati da un indicizzatore, mentre il secondo non è soggetto a restrizioni di sorta. I controlli sul linguaggio possono anche includere relazioni gerarchiche tra i termini indice, o imporre l'uso di certi termini unicamente come aggettivi: non vi è alcun limite al tipo di controlli sintattici che si possono operare su un linguaggio.

Il linguaggio indice che risulta dall'algoritmo di fusione visto prima può essere definito non controllato, post-coordinato e derivato. Il vocabolario dei termini indice in ogni fase dell'evoluzione della collezione dei documenti è proprio l'insieme di tutti i nomi di classe derivati dalla fusione.

Ci sono giudizi discordanti sul tipo di linguaggio indice migliore per il recupero dei documenti. Le discussioni principali vertono addirittura sulla certezza che l'indicizzazione automatica sia migliore di quella manuale. Entrambe possono essere rese complesse a piacere, ma sembra che comunque non valga la pena aggiungere controlli più elaborati

dell'attribuzione dei pesi ai termini indice. In altre parole, ciò vuol dire che i vocabolari non controllati basati sul linguaggio naturale hanno un'efficacia di recupero confrontabile con quella dei vocabolari con controlli complessi, e che quindi si può tranquillamente ricorrere a tecniche di controllo dei termini indice basate sui relativi pesi attribuiti (le tecniche più semplici da implementare).

2.3.2.5 Attribuzione di pesi ai termini indice

Per tradizione i due fattori più importanti che influenzano l'efficacia di un linguaggio indice sono l'*esaustività* dell'indicizzazione e la *specificità* del linguaggio indice. Anche se non esistono definizioni universalmente accettate di questi due termini, si riportano quelle dichiarate da Keen e Digger: per ogni documento, *l'esaustività dell'indicizzazione* è definita come il numero di argomenti differenti indicizzati, e la *specificità del linguaggio indice* è la capacità del linguaggio di descrivere precisamente gli argomenti. Inoltre essi definiscono la *specificità dell'indicizzazione* come il livello di precisione con cui un documento viene indicizzato. Gli indicizzatori umani riescono ad attribuire in maniera approssimata un punteggio alla loro indicizzazione per aumentare l'esaustività o la specificità, ma ciò diventa molto difficile per l'indicizzazione automatica.

In pratica, un alto livello di esaustività conduce ad un alto livello di richiamo e ad un basso livello di precisione, mentre un basso livello porta alla condizione opposta. Al contrario, un alto livello di specificità porta ad un alto livello di precisione e ad un basso livello di richiamo, e così via. Sembrerebbe quindi che ci sia un livello ottimo di esaustività e specificità per una data popolazione di utenti.

Diversi studiosi hanno tentato di legare questi due fattori alle statistiche sulle collezioni di documenti. Ad esempio, si può assumere che l'esaustività sia legata al numero di termini indice assegnati ad un dato documento, e la specificità al numero di documenti ai quali è assegnato un dato termine in una data collezione. L'importanza di questa relazione piuttosto vaga è che i due fattori sono legati alla *distribuzione dei termini indice* nella collezione. Variazioni nel numero di termini indice per documento porta a variazioni corrispondenti nel numero di documenti per termine e viceversa.

In precedenza si è visto che Luhn postulò un potere di discriminazione per i termini indice variabile come una funzione dell'ordinamento in base alla loro frequenza di occorrenza. Si può usare lo stesso conteggio delle frequenze per fornire uno schema di attribuzione di pesi ai termini individuali in un documento, in maniera direttamente proporzionale. A prima vista questo schema può sembrare inconsistente con l'ipotesi di Luhn, la quale afferma che il potere di discriminazione crolla alle frequenze alte (vedere Figura 2.1). Ma lo schema diventerebbe consistente se la soglia superiore fosse spostata verso sinistra fino al punto di picco, e questo è ciò che in effetti accade negli esperimenti che usano questo particolare tipo di attribuzione dei pesi.

Sono stati effettuati vari tentativi di attribuzione dei pesi basandosi sul modo in cui i termini indice sono distribuiti nell'intera collezione. Il vocabolario dei termini indice di una collezione ha spesso una distribuzione di Zipf come visto prima, e si è mostrato sperimentalmente che se vi sono N documenti ed un termine indice compare in n di essi allora l'attribuzione di un peso pari a $\log(N/n) + 1$ porta ad un recupero più efficace rispetto al caso in cui il termine fosse usato senza peso. Se si assume che la specificità sia inversamente proporzionale al numero di documenti in cui un termine indice compare allora si nota che l'attribuzione dei pesi dà maggiore importanza ai termini più specifici.

Per riassumere si può quindi affermare che l'attribuzione dei pesi per frequenza nei documenti pone enfasi sulla descrizione del contenuto, mentre l'attribuzione dei pesi per specificità enfatizza la capacità dei termini di discriminare un documento dall'altro.

Esistono poi vari modi di combinare le frequenze dei termini e le frequenze dei documenti inverse, e studi empirici mostrano che la combinazione ottima può variare da collezione a collezione. Generalmente, *tf* è moltiplicata per *idf* per ottenere un peso dei termini combinato. Un'alternativa potrebbe essere ad esempio scartare i termini con *idf* al di sotto di una soglia fissata, il che sembra essere una soluzione sensibilmente migliore per ricerche che richiedono alta precisione. Come detto poco fa, anche in questi casi entrambe le misure sono di solito stemperate prendendone i logaritmi piuttosto che la misura diretta.

Dato che il peso dei termini è definito dal componente *tf*, la formula combinata è influenzata pesantemente dalla lunghezza del documento: infatti un documento lungo è solito avere più occorrenze per un termine rilevante rispetto ad uno breve; questo non dovrebbe riflettere una rilevanza maggiore ma semplicemente una lunghezza maggiore.

La maggior parte degli algoritmi in uso introducono la lunghezza del documento come un fattore di normalizzazione di qualche tipo, nel caso in cui i documenti nella base di documenti variano sensibilmente in lunghezza. È pratica comune ridurre ogni peso del termine nel documento utilizzando un fattore derivato dalla lunghezza del documento espressa in parole o caratteri; la forza della riduzione può essere controllata da un parametro che viene fissato dopo un'opportuna sperimentazione con la collezione in esame. Ciò dà una normalizzazione abbastanza stretta: si promuovono i documenti brevi in maniera spropositata, e di solito gli effetti devono essere in qualche modo attenuati, dato che i documenti lunghi spesso sono più interessanti di ciò che questa normalizzazione ricava.

2.3.2.6 Termini composti e termini tecnici

Si è visto che contare le singole parole di un testo è la tecnica più semplice ed utilizzata, ma assumere che tali parole esprimano da sole l'argomento del testo è una delle più ovvie semplificazioni del modello. Indicizzare testi sull'argomento "porto franco" utilizzando i singoli termini "porto" e "franco" è intuitivamente meno utile rispetto al concentrarsi sulla combinazione "porto franco". Tuttavia si è notato che negli esperimenti progettati per testare l'utilità dei termini composti qualunque altra strategia oltre l'indicizzazione per parola singola è risultata ingombrante e costosa in termini di memoria e risorse di elaborazione, mentre ha aggiunto relativamente poco in prestazioni. In ogni caso, la potenza di discriminazione dei termini a parola singola è molto più forte di quella di ogni altra sorgente di informazioni. La ricerca di termini composti può essere condotta tramite tecniche statistiche o linguistiche.

Esiste un'altra maniera di espandere la ricerca a parole oltre i termini singoli, consistente nel costruire semplicemente una tabella di parole adiacenti nel testo (*n-grammi*). Sono stati compiuti diversi sforzi in tal senso, come ad esempio quello di Magnus Merkel che implementò un tool per il recupero di sequenze di parole ricorrenti nel testo.

Oppure, sfruttando appieno la teoria linguistica, possono essere individuati altri tipi di combinazioni arbitrarie e ricorrenti nel testo, che vanno sotto il nome di *collocazioni*. A tal proposito Frank Smadja implementò un insieme di tool per recuperare collocazioni di vari tipi usando informazioni sia statistiche che lessicali; egli identificò tre tipi principali di collocazioni:

- *relazioni predicative*, che sussistono tra i verbi ed i loro oggetti nelle costruzioni ricorrenti;
- *frasi* con nomi idiomatici;

- *template di frase*, dove solo una certa parte varia da istanza ad istanza.

Ad ogni modo, tali insiemi di informazioni possono essere visti come un insieme di conoscenze a cui applicare delle statistiche semplici ed usarle poi per estrarre *termini tecnici*. I termini tecnici sono una categoria specifica di parole che funziona in maniera molto simile ai nomi propri. Questi non possono essere facilmente modificati (i loro elementi non possono essere mescolati o sostituiti da potenziali sinonimi) e riferiti tramite dei pronomi. Quindi, i termini tecnici tendono a rimanere invariati in tutto il testo, e tra testi diversi.

Il semplice ed affascinante algoritmo sviluppato da Justeson e Katz per individuare i termini tecnici composti considera tutte le sequenze composte da più parole che hanno un nome in testa e memorizza quelle che compaiono più di una volta. Questo metodo dà un quadro sorprendentemente caratteristico di un argomento del testo, dato che quest'ultimo è di natura tecnica o perlomeno non banale. È il caso di notare il punto di forza della tecnica: il fatto che una frase con nome complessa sia usata più di una volta nella stessa identica forma è la prova sufficiente per la sua qualità speciale di termine tecnico. È la ripetizione, e non la frequenza, che viene presa in considerazione per i termini tecnici.

Si suggerisce spesso l'uso di metodi linguistici allo scopo di estrarre termini composti. La prima citazione ai metodi linguistici – nel caso particolare, le *trasformazioni* – per la normalizzazione delle variazioni sintattiche nel testo risale alla fine degli anni cinquanta, e recita "L'utilizzo dell'indicizzazione basata sulla linguistica è motivato dal bisogno di individuare effettivamente i termini composti". La ricerca in tale tipo di indicizzazione praticamente utilizza i termini composti individuati statisticamente come base di partenza e tenta di identificare termini migliori. Un esempio è il lavoro di Strzalkowski, che tentò di trovare combinazioni di parole nel contenuto linguistico attraverso l'analisi statistica di coppie di parole e la relazione di dipendenza tra esse. Egli effettuò la sperimentazione

usando strutture di modifica della testa dai testi completamente parsati per estrarre i termini indice: ciò permette di normalizzare le frasi come “information retrieval” e “retrieval of information” nella stessa rappresentazione indiciale.

Ricollegandosi a quanto detto precedentemente, è stato tuttavia più volte mostrato che i termini composti non migliorano l’efficienza del recupero per i testi in lingua inglese se non marginalmente, e che lo sforzo necessario per implementare ed applicare metodi linguistici in generale non è controbilanciato da un beneficio raffrontabile. In base a queste considerazioni, Robertson e Sparck Jones scoraggiano gli implementatori dal prendere in considerazione altri elementi oltre i termini composti conosciuti in anticipo: “Scoprire, per ispezione, le stringhe composte contenute in un file è un’impresa molto costosa. In generale queste elaborazioni sono difficili da gestire e non raccomandate ai principianti”.

2.3.2.7 Discriminazione e/o rappresentazione

Ci sono due modi contrapposti di vedere il problema della caratterizzazione dei documenti per il recupero. Il primo è di caratterizzare un documento attraverso una rappresentazione dei suoi contenuti, indipendentemente dal modo in cui gli altri documenti possono essere descritti, e prende il nome di *rappresentazione senza discriminazione*. Il secondo è di caratterizzare un documento discriminandolo da tutti (o potenzialmente tutti) gli altri documenti nella collezione, e prende il nome di *discriminazione senza rappresentazione*. Ovviamente, nella pratica non si assume nessuna di queste posizioni estreme, sebbene sia utile identificarle quando si pensa al problema della caratterizzazione; si tenta invece di raggiungere il migliore compromesso tra rappresentazione e discriminazione, bilanciando l’esaustività e la specificità dell’indicizzazione. La maggior parte dei metodi automatici di

indicizzazione sono un misto tra rappresentazione e discriminazione: ad esempio la rimozione delle parole ad alta frequenza tramite una lista di stopword è un tentativo di incremento del livello di discriminazione tra i documenti.

Prediligere la rappresentazione porta a metodi orientati al documento, cioè a concentrarsi totalmente sulla descrizione di ciò di cui tratta il documento. Questo approccio tenderà implicitamente a tirare in ballo risultati dal campo dell'intelligenza artificiale, in particolare quelli riguardanti la costruzione di modelli informativi per i contenuti di un testo.

Prediligere la discriminazione porta a metodi orientati alla query. Questo punto di vista presuppone che si possa prevedere la popolazione delle query presentate al sistema di information retrieval e, in base a questa, provare a caratterizzare i documenti in maniera ottimale.

2.3.2.8 Classificazione automatica delle parole chiave

Molti sistemi di recupero automatici si affidano ai thesauri per modificare i rappresentanti di query e documenti al fine di aumentare la probabilità di recuperare documenti rilevanti. Esperimenti condotti con molti tipi differenti di thesauri hanno concluso che l'uso di quelli semplici è giustificato in termini di miglioramento dell'efficacia del recupero.

Praticamente molti thesauri sono costruiti manualmente in due maniere:

- si collegano le parole considerate relative allo stesso argomento;
- si collegano le parole considerate relative ad argomenti tra loro legati.

Il primo tipo di thesaurus collega parole che sono intercambiabili ponendole in classi di equivalenza. Poi si può scegliere una parola per rappresentare ogni classe ed usare una lista

di queste parole per costituire un vocabolario controllato. Da qui si può istruire un indicizzatore per selezionare le parole idonee per indicizzare un documento, oppure si può istruire l'utente per selezionare quelle appropriate per esprimere la query. Si potrebbe usare lo stesso thesaurus in maniera automatica per identificare le parole di una query adatte al recupero.

Il secondo tipo usa collegamenti semantici tra le parole per legarle, ad esempio gerarchicamente.

Sono stati proposti metodi per la costruzione automatica dei thesauri, tuttavia questi si basano tipicamente sulla sintassi e sulla statistica mentre quelli manuali si basano sulla semantica (ad esempio sono in grado di riconoscere sinonimi più generali o relazioni più specifiche). L'uso della sintassi si è mostrato poco proficuo, perciò si utilizzano metodi statistici, basati principalmente su pattern di co-occorrenza delle parole nei documenti (termini o parole chiave).

La relazione fondamentale dietro la costruzione automatica delle classi di parole chiave è la seguente: Se le parole chiave *a* e *b* sono sostituibili l'una con l'altra nel senso che si può accettare un documento contenente una di queste in risposta ad una richiesta contenente l'altra, allora esse hanno lo stesso significato o si riferiscono ad un argomento comune. Un modo di capire se due parole chiave sono legate è di osservare i documenti in cui esse compaiono: se tendono a comparire entrambe negli stessi documenti, vi è la possibilità che si riferiscano allo stesso oggetto e quindi possano essere scambiate l'una con l'altra.

Partendo da questo principio, non è difficile notare che si può costruire automaticamente una classificazione delle parole chiave, in cui le classi sono usate in maniera analoga a quelle del thesaurus costruito manualmente citato prima. Nello specifico possono essere identificati due approcci principali all'uso delle classificazioni delle parole chiave:

- sostituire ogni parola chiave in un rappresentante di un documento (e di una query) con il nome della classe in cui questa compare;
- sostituire ogni parola chiave con tutte le parole chiave che compaiono nella classe alla quale questa appartiene.

Se si pensa ad una semplice strategia di recupero funzionante tramite il confronto dei descrittori, siano essi parole chiave o nomi di classi, allora 'espandere' i rappresentanti in una delle maniere precedenti aumenterà il numero di corrispondenze tra i documenti e le query, e quindi tenderà a migliorare il richiamo. Il secondo metodo migliorerà anche la precisione. Diversi esperimenti hanno confermato che in generale si ottiene una migliore prestazione nel recupero con l'ausilio della classificazione automatica delle parole chiave rispetto al recupero senza classificazione.

2.3.3 Strutture dei file

Qui si approfondirà la maniera in cui sono usate le strutture dei file nel document retrieval. La maggior parte degli studi sulle strutture di file è concentrata sulle applicazioni nella gestione dati, come apparirà chiaro nella terminologia usata per descrivere i concetti di base.

2.3.3.1 Organizzazione logica o fisica e indipendenza dei dati

Quando si parla di strutture di file bisogna fare una distinzione importante tra l'organizzazione *logica* e *fisica* dei dati. In generale una struttura di file specifica la struttura logica dei dati, cioè le relazioni che esistono tra le entità che costituiscono i dati

indipendentemente dal modo in cui esse sono realizzate all'interno di un elaboratore. L'organizzazione fisica punta ad ottimizzare l'uso della memoria (sia essa volatile o di massa) per immagazzinarvi una particolare struttura logica. Normalmente non vi è una corrispondenza tra unità di memoria fisica e unità di memoria logica: spesso più unità di strutture logiche (record) sono contenute in una unità di struttura fisica.

La teoria delle basi di dati è stata da sempre incentrata sul concetto di *indipendenza dei dati*, la proprietà di poter scrivere applicazioni indipendentemente dalla struttura logica dei dati con cui esse interagiscono. Eventuali cambiamenti di struttura fisica non devono influenzare l'applicazione, e ciò si ottiene interponendo un modello dei dati tra la base di dati e l'utente: quest'ultimo vede il modello piuttosto che la base di dati, e le applicazioni comunicano esclusivamente con il modello.

Purtroppo però non è ancora stato definito in maniera assoluta un buon modello di dati, e non esistono ancora implementazioni ufficialmente riconosciute di quelli che sperimentalmente sono i più avanzati sistemi teorici.

Ci sono state diverse formalizzazioni del trattamento dei dati ad un livello astratto. Quella più conosciuta per ora è senz'altro il *modello relazionale*, sviluppato originariamente da Codd, in cui i dati sono descritti tramite *n*-uple di valori di certi attributi. In maniera più formale, se i dati sono descritti da *relazioni*, si può rappresentare una relazione su un insieme di domini $D_1,\ldots, D_n$ tramite un insieme di *n*-uple ordinate ognuna della forma $(d_1,\ldots, d_n)$ dove $d_i \in D_i$. Dato che è piuttosto difficile trattare con le relazioni generiche, sono stati introdotti vari livelli di normalizzazione (di fatto tre) per limitare il numero di relazioni permesse.

Un altro approccio è quello *gerarchico*, usato in molti DBMS, ed in esso i dati sono rappresentati sotto forma di gerarchie. Questo approccio è sicuramente più restrittivo di quello relazionale, ma in molte applicazioni una struttura del genere dà una buona

approssimazione della struttura naturale dei dati, e la perdita nella precisione della rappresentazione viene controbilanciata da una maggiore efficienza e semplicità di rappresentazione.

Un terzo approccio è quello *a rete*, in cui gli item di dati sono collegati in una rete: esiste un certo collegamento tra due item se esso soddisfa una certa condizione sui loro attributi (ad esempio condividono un attributo). È più generale dell'approccio gerarchico perché un nodo può avere diversi nodi genitore, ed ha la stessa potenza descrittiva dell'approccio relazionale.

Per concludere, sembrerebbe quindi che ogni applicazione nel campo dell'archiviazione automatica e dell'information retrieval possa essere implementata tramite l'uso di una base di dati appropriata, ma ciò non è possibile, almeno nel prossimo futuro, per diverse ragioni. Una di queste è dovuta al fatto che i sistemi di basi di dati sono general-purpose, mentre quelli di archiviazione e recupero automatico sono special-purpose: fornire generalità ha un costo, che in questo caso è ancora troppo grande. Inoltre la stessa implementazione special-purpose non è realizzata molto facilmente.

2.3.3.2 Terminologia di base

La terminologia delle strutture di file si è evoluta in modo confusionario, senza prestare attenzione alla consistenza, all'ambiguità, o alla possibilità di effettuare il tipo di distinzioni realmente importanti. Fu solo molto più tardi che comparve il bisogno di un linguaggio ben definito e non ambiguo, che sarà utilizzato per la presentazione delle strutture dati.

Dato un insieme di 'attributi' *A* ed un insieme di 'valori' *V*, un *record R* è un sottoinsieme del prodotto cartesiano *A* x *V* in cui ogni attributo ha uno ed un solo valore; quindi *R* è un insieme di coppie ordinate della forma (attributo, valore). Ad esempio, il record per un documento elaborato da un algoritmo automatico per l'analisi del contenuto sarebbe

$$R = \{(K_1, x_1), (K_2, x_2), \ldots, (K_m, x_m)\}$$

I K_i sono le parole chiave che fungono da attributi ed il valore x_i può essere visto come un peso numerico. Di solito i documenti sono caratterizzati semplicemente dalla presenza o assenza di parole chiave, nel qual caso si scrive

$$R = \{K_{t1}, K_{t2}, \ldots, K_{ti}\}$$

dove K_{ti} è presente se $x_{ti} = 1$, assente altrimenti.

I record sono collezionati in unità logiche chiamate file: questi ultimi permettono di riferirsi ad un insieme di record tramite il nome del file. I record in un file sono spesso organizzati secondo le relazioni tra tali record, e proprio questa organizzazione è detta struttura di file (o struttura dati).

Nella descrizione è difficile mantenere separate le caratteristiche logiche da quelle fisiche. Queste ultime sono di solito forzate dal mezzo di memoria (nastro, disco, ecc.). Alcune caratteristiche possono essere definite in maniera astratta ma sono poco comprese quando poi sono illustrate concretamente: una di queste è il *campo*. Nelle implementazioni di un record, i valori attributo sono di solito posizionali, cioè l'identità di un attributo è data dalla posizione del suo valore all'interno del record. Quindi i dati all'interno di un record sono

registrati in maniera sequenziale ed hanno un inizio ed una fine ben definiti. Il record è diviso in *campi* e l'*n*-simo campo contiene l'*n*-simo valore attributo (Figura 2.12).

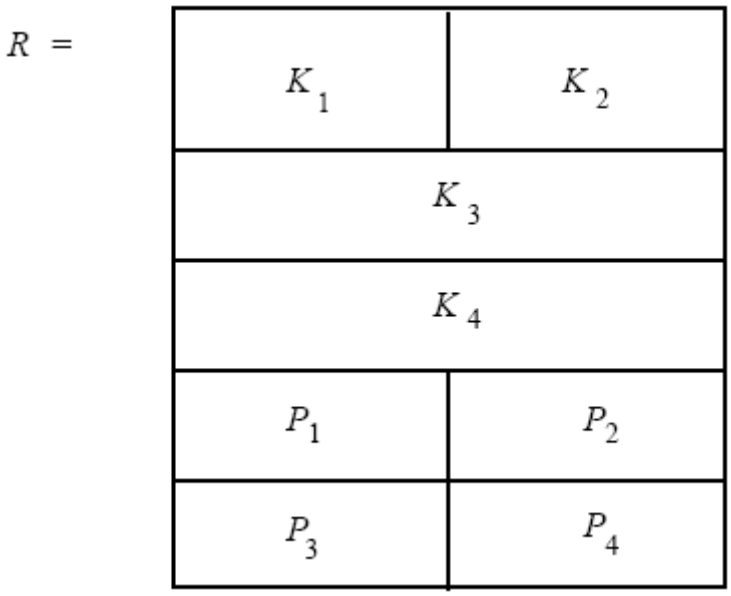


Figura 2.12 Un esempio di record con campi associati

I campi non hanno necessariamente una lunghezza costante; allora, per trovare il valore dell'attributo K_4 bisogna trovare prima l'indirizzo del record *R* (l'indirizzo di partenza del record in memoria fisica) e poi leggere i dati nel quarto campo.

I campi particolari etichettati con P_i in Figura 2.12 contengono indirizzi di altri record, e sono chiamati *puntatori*. Ognuno di essi è associato ad una particolare coppia attributo-valore; ad esempio si possono usare per collegare tutti i record aventi il valore x_1 (relativo all'attributo K_1) pari ad *a*, o in maniera simile aventi x_2 pari a *b*, ecc. (vedere Figura 2.13)

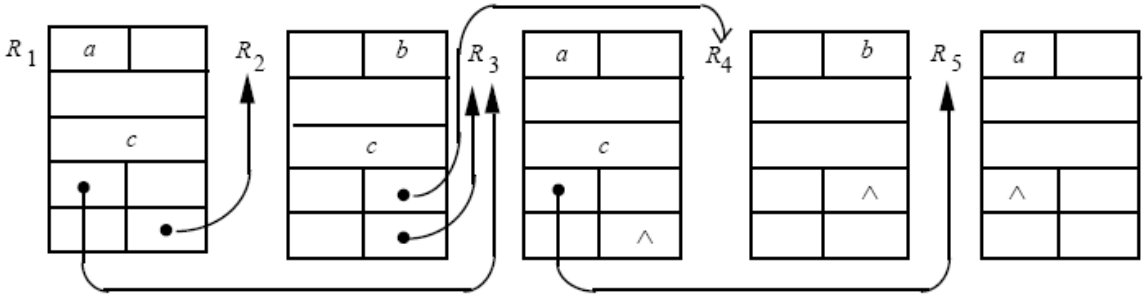


Figura 2.13 Uso di puntatori per collegare record

Per indicare che un record è l'ultimo elemento puntato in una lista di record, si utilizza il *puntatore nullo* $\wedge$. Il puntatore associato all'attributo *K* nel record *R* è un *K-puntatore*, ed un attributo (parola chiave) utilizzato in questo modo per organizzare un file è detto *chiave*.

A questo punto bisogna ancora definire una lista *L* di record rispetto alla parola chiave *K*, brevemente una *K-lista*, come un insieme di record contenenti *K* tali che:

1. i *K*-puntatori siano distinti (prevenzione di cicli);
2. ogni *K*-puntatore non nullo in *L* dia l'indirizzo di un record contenuto in *L* (proprietà insiemistica di chiusura della *K*-lista);
3. ci sia un unico record in *L* non puntato da alcun record contenente *K* (definizione della testa della lista);
4. ci sia un unico record in *L* contenente il *K*-puntatore nullo (definizione della coda della lista).

Nel caso dell'esempio precedente si ottiene quindi:

K_1-lista : R_1, R_2, R_5

K_2-lista : R_2, R_4

K_4-lista : R_1, R_2, R_3

Infine, si ha bisogno della definizione di directory di un file. Sia F un file i cui record contengano m parole chiave differenti K_1, K_2,…, K_m. Sia n_i il numero di record contenenti la parola chiave K_i, e h_i il numero di K_i-liste in F. Inoltre, denotiamo con a_{ij} l'indirizzo di partenza della j-esima K_i-lista. Allora la directory è l'insieme delle sequenze

$(K_i, n_i, h_i, a_{i1}, a_{i2},\ldots, a_{ih_i})$ $i = 1, 2,\ldots, m$

Si possono ora fornire le definizioni dei file sequenziali, dei file invertiti, dei file sequenziali indicizzati e dei file multi-lista.

2.3.3.3 File sequenziali

Un file sequenziale è la struttura più semplice di tutte: non contiene né directory né puntatori di collegamento. I record sono generalmente organizzati nell'ordine lessicografico in base al valore di qualche chiave; in altre parole, si sceglie un particolare attributo il cui valore determina l'ordine dei record. Nel caso in cui il valore dell'attributo sia costante per un gran numero di record (e quindi non riesca a discriminarli efficacemente) si sceglie una seconda chiave per ristabilire un ordine.
Ovviamente, l'implementazione di questa struttura di file richiede l'uso di una funzione di ordinamento.
I vantaggi principali sono:

- facilità di implementazione;

- rapido accesso al record successivo in ordine lessicografico.

Gli svantaggi invece sono:

- difficoltà di aggiornamento (l'inserimento di un nuovo record può richiedere lo spostamento di una grande quantità dei contenuti del file);
- accesso casuale estremamente lento.

A volte si considera un file come sequenziale a dispetto del fatto che non è ordinato secondo alcuna chiave. Si potrebbe pensare che la data di acquisizione sia il valore chiave, così le entry più nuove sarebbero aggiunte alla fine del file e quindi non ci sarebbero difficoltà nell'aggiornamento.

2.3.3.4 File invertiti

Un file invertito è una struttura di file in cui ogni *K*-lista contiene solo un record. Questo implica che la directory sia tale che $n_i = h_i$ per tutti gli i, e cioè che il numero di record contenenti K_i sia uguale al numero di K_i-liste. Allora la directory ha un indirizzo per ogni record contenente K_i. Dal punto di vista del document retrieval ciò significa che data una parola chiave si possono immediatamente individuare gli indirizzi di tutti i documenti contenenti quella parola chiave. Nell'esempio precedente si assume che una entry non vuota nel campo corrispondente ad un attributo indichi la presenza di una parola chiave, in caso contrario la sua assenza. Quindi la directory punta al file come mostrato in Figura 2.14.

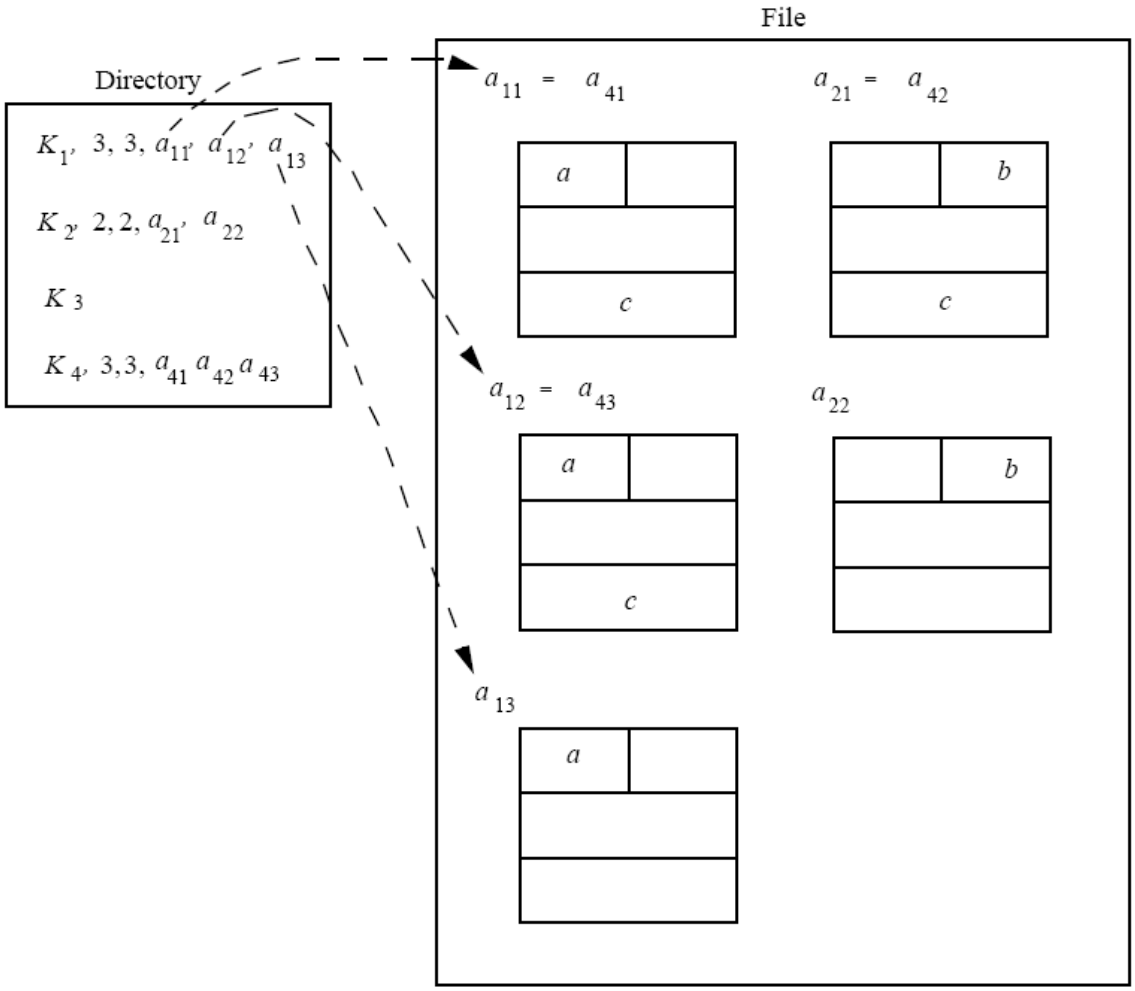


Figura 2.14 File invertito

La definizione di file invertito non richiede che gli indirizzi nella directory siano in un certo ordine; tuttavia, per facilitare operazioni come la congiunzione (AND) e la disgiunzione (OR) su una qualunque delle due liste invertite, gli indirizzi sono normalmente mantenuti nell'ordine dato dai numeri di record. Ciò significa che le operazioni di AND e OR possono essere eseguite in un solo passo in entrambe le liste. Il rovescio della medaglia è ovviamente una maggiore lentezza di aggiornamento.

2.3.3.5 File sequenziali indicizzati

Un file sequenziale indicizzato è un file invertito in cui per ogni parola chiave K_i si ha che $n_i = h_i = 1$ e $a_{11} < a_{21} < a_{m1}$. Ciò è valido solo se ogni record ha un'unica parola chiave o coppia attributo-valore. Quindi in pratica questo insieme di record può essere ordinato in modo sequenziale tramite una chiave. Ogni valore della chiave compare nella directory con l'indirizzo associato del suo record. Un'interpretazione ovvia di chiave di questo tipo è il numero progressivo del record: nell'esempio trattato infatti nessuno degli attributi funge allo scopo se non il numero del record. Un esempio di file di questo tipo è riportato in Figura 2.15. In essa compare R_i al posto di K_i proprio per evidenziare la natura della chiave.

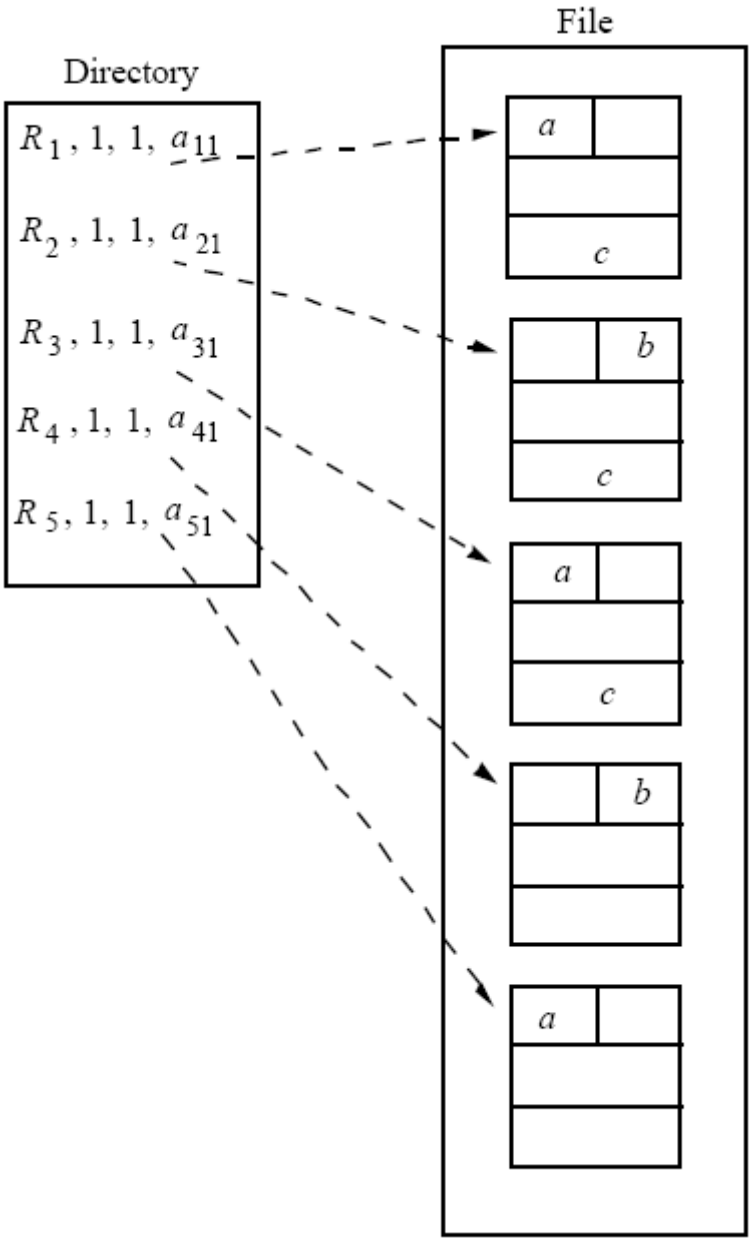


Figura 2.15 File sequenziale indicizzato

In letteratura un file sequenziale indicizzato è visto solitamente come un file sequenziale con una gerarchia di indici. Ciò non vuole essere una contraddizione della definizione precedente, ma semplicemente una descrizione della maniera in cui è implementata la directory. Non è strano che spesso gli indici (qui ‘indice ‘ sta per ‘directory’) siano legati

alle caratteristiche del dispositivo di memoria; ad esempio (Figura 2.16) ci possono essere tre livelli di indicizzazione: traccia, cilindro e master.

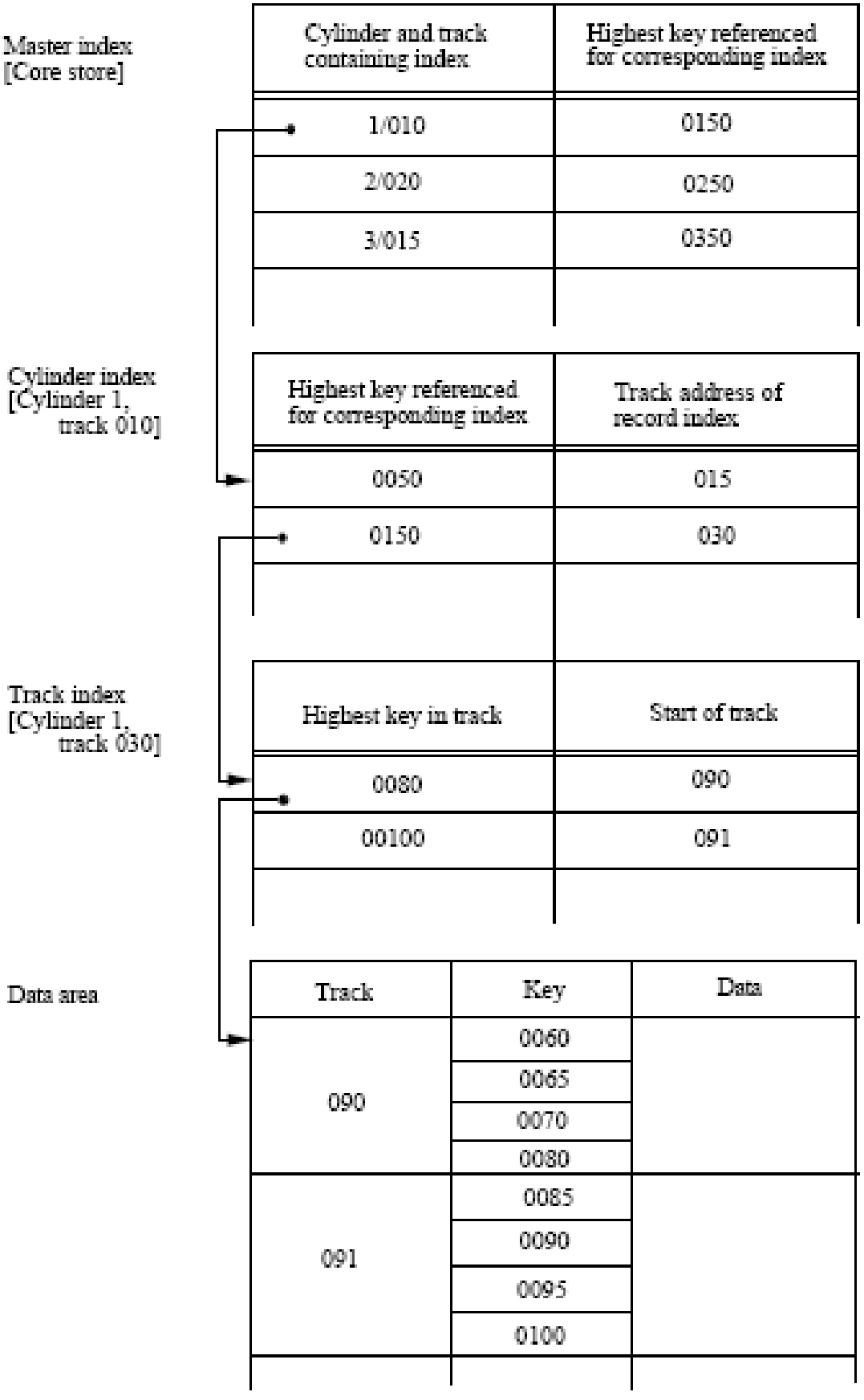


Figura 2.16 Esempio di implementazione di un file sequenziale indicizzato

Ogni entry nell'indice di traccia contiene informazioni sufficienti per individuare l'inizio della traccia e la chiave dell'ultimo record nella traccia, che è poi anche il valore più alto in quella traccia. Esiste uno di tali indici di traccia per ogni cilindro. A sua volta ogni entry nell'indice di cilindro fornisce l'ultimo record di ogni cilindro e l'indirizzo dell'indice di traccia per quel particolare cilindro. Se lo stesso indice di cilindro è memorizzato su tracce, allora l'indice master fornisce la chiave più grande referenziata da ogni traccia dell'indice di cilindro e l'indirizzo di partenza di tali tracce.

Non si è prevista la possibilità di un overflow durante il processo di aggiornamento, ma questo problema è arginato predisponendo un'area di overflow gestita nella directory: ciò naturalmente incrementa il numero di entry di conteggio in ogni entry dell'indice.

2.3.3.6 Multi-liste

Una multi-lista è solo un file invertito leggermente modificato. Vi è solo una lista per parola chiave, quindi $h_i = 1$. I record contenenti una particolare parola chiave K_i sono collegati insieme a formare la K_i-lista, e l'inizio di questa è fornito dalla directory, come illustrato in Figura 2.17.

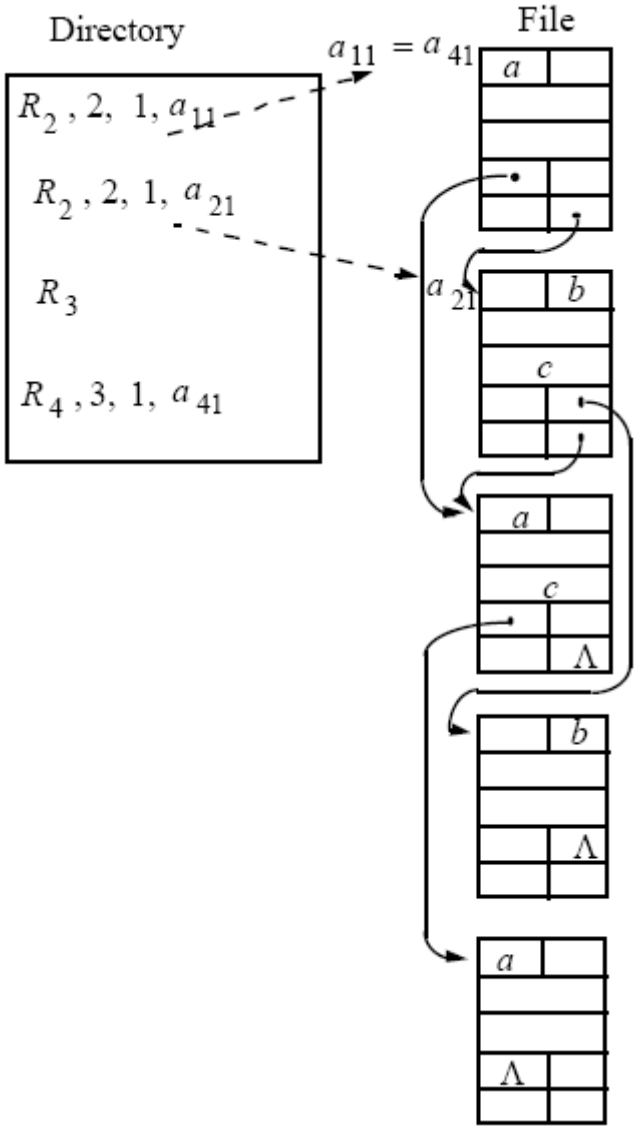


Figura 2.17 Multi-lista

Dato che non c'è alcuna K_3-lista, si potrebbe omettere il campo riservato al suo puntatore, e si potrebbe fare ciò per ogni puntatore nullo, in maniera da non generare ambiguità su quale puntatore appartenga a quale parola chiave. Per ottenere ciò, specialmente se i valori dei dati (attributo-valori) sono in formato fisso, si fa in modo che il puntatore non punti all'inizio del record ma alla locazione del prossimo puntatore nella catena.

La multi-lista è progettata per sopperire alle difficoltà dovute all'operazione di aggiornamento in un file invertito. In quest'ultimo caso gli indirizzi nella directory sono

mantenuti secondo l'ordine del numero del record, ma quando bisogna aggiungere un nuovo record al file dovendo mantenere questo ordine l'operazione può diventare molto costosa in termini di tempo. Con la multi-lista invece non vi sono problemi in quanto si aggiornano le *K*-liste appropriate semplicemente collegando in coda il nuovo record. Lo svantaggio è ovviamente un aumento del tempo di ricerca, cosa comunque comune in molte delle strutture di file esistenti.

2.3.3.7 Multi-liste cellulari

Un'ulteriore modifica della multi-lista si basa sull'osservazione che molti dispositivi di memoria sono divisi in *pagine*, recuperabili una alla volta. Una *K*-lista può attraversare diversi confini di pagina, e questo si traduce nel fatto che può essere necessario accedere a diverse pagine per recuperare un record. Per evitare ciò si può utilizzare la multi-lista cellulare, in cui le *K*-liste sono limitate in modo da non attraversare i confini della pagina (cellula).

Allora, nel rispetto della notazione precedente, la directory per una tale struttura è l'insieme delle sequenze

$(K_i, n_i, h_i, a_{i1},\ldots, a_{ih_i})$ $i = 1, 2,\ldots, m$

in cui h_i è preso in modo da assicurare che una K_i-lista non attraversi il confine di pagina. Nella fase di implementazione, come per un file sequenziale indicizzato, si memorizzano informazioni aggiuntive insieme ad ogni indirizzo per poter individuare la pagina giusta per ogni valore della chiave.

2.3.4 Strategie di ricerca

Finora è stato detto molto poco sul processo tramite cui si individuano le informazioni richieste. Nel caso del recupero di documenti le informazioni sono un sottoinsieme dei documenti che paiono rilevanti alla query. Nei capitoli precedenti si è fatto riferimento solo occasionalmente all'efficienza della ricerca ed all'idoneità di una struttura di file alla ricerca. Il tipo di ricerca di interesse non è quella solita in cui il risultato è un giudizio secco (l'oggetto è presente o è assente), ma si è orientati verso strategie di ricerca in cui i documenti recuperati possano essere più o meno rilevanti per la richiesta.

Tutte le strategie di ricerca sono basate sui confronti tra la query e i documenti memorizzati.

A volte le distinzioni fatte tra tipi differenti di strategie di ricerca possono essere comprese osservando il linguaggio della query, cioè il linguaggio con cui si esprime il bisogno informativo. Spesso la natura del linguaggio di query definisce la natura della strategia di ricerca. Ad esempio, un linguaggio che permette di esprimere i comandi di ricerca in termini di combinazioni logiche di parole chiave di solito impone l'uso di una ricerca booleana, che trae risultati da confronti logici (e non numerici) tra la query ed i documenti. Nel seguito però non si esamineranno i linguaggi di query, ma si tratteranno i meccanismi di ricerca per coglierne le differenze.

2.3.4.1 Elaborazione tipica delle query

Nel modello standard la richiesta di informazioni, posta in linguaggio naturale o come un insieme di termini di ricerca, è trattata in maniera molto simile ai documenti nel database: è

analizzata sulla base dell'occorrenza dei termini, ed è trasformata in un vettore di pesi dei termini (in cui di solito il termine *query* è riservato) simile ai vettori di termini calcolati per i documenti.

Ma i documenti e le richieste informative sono tipicamente molto differenti. Originariamente, dal punto di vista del ricercatore il modello originale di Luhn consisteva nel costruire un qualcosa approssimativamente della stessa forma di ciò che era stato fatto per i documenti. Si assumeva infatti che questi fossero brevi, nella forma di riepiloghi piuttosto che di testi interi, cosa che faceva sembrare pratico questo modello; ciò naturalmente non è più vero. Al contrario, come è stato stabilito sia dall'osservazione informale che da diversi esperimenti formali, le richieste informative ai sistemi di information retrieval tendono ad essere molto brevi: la maggioranza è composta di tre o anche meno parole. Ciò dà pochissimo spazio alla maggior parte dei metodi linguistici, optando per metodi che incoraggino i ricercatori a produrre richieste più articolate utilizzando un maggior numero di termini.

Dato che le richieste sono di lunghezza e tipo differenti rispetto ai documenti obiettivo, i rispettivi vettori di termini sono di solito trattati in maniera differente. Ad esempio, la maggior parte dei sistemi usa un calcolo della frequenza binaria per i termini della query: si utilizza l'occorrenza, piuttosto che la frequenza, come base per pesare i termini della query.

2.3.4.2 Confronto tra query e documenti

Data una query ed una rappresentazione dei documenti, entrambe in forma vettoriale, indipendentemente da come siano calcolati gli elementi nel vettore, il problema consiste

nel confronto tra i due vettori di termini per trovare i documenti che soddisfano la richiesta. La maggior parte dei motori di ricerca pone maggiore sforzo nella fase di indicizzazione per evitare algoritmi di confronto complicati: un metodo comune è utilizzare il prodotto scalare convenzionale dei due vettori, aggiungendo semplicemente i prodotti a coppie di ogni elemento dei due vettori considerati, ed ottenendo quindi un *punteggio di similarità* per ogni documento confrontato al vettore. Esistono numerose variazioni a questo schema, create per sfruttare le caratteristiche della particolare collezione e/o popolazione di utenti. La maggior parte dei modelli esistenti mette a disposizione un certo numero di parametri per ottenere tali variazioni.

2.3.4.3 Ricerca booleana

Il recupero booleano è basato sulla semplice teoria algebrica degli insiemi, ed usa i pesi binari per modellare la collezione: i documenti sono rappresentati dalle occorrenze delle parole o dei termini in essi contenuti, e non dalle frequenze (in maniera simile alle query). Si esamina l'insieme dei documenti nel database per vedere quali di questi condividono termini con la query in maniera esatta, senza tenere conto delle frequenze o di altre funzioni di pesatura, e quindi se il prodotto scalare, menzionato prima, dei due vettori è maggiore di zero. Ciò significa che per ogni documento viene presa una decisione binaria: o questo è nell'insieme e soddisfa la richiesta, o è all'esterno e non la soddisfa.

Tale metodo funziona ragionevolmente bene se l'insieme dei termini indice è limitato per progetto a parole assegnate in maniera particolare, o in alternativa per basi di documenti in cui questi sono brevi e concisi (ad esempio una lista di riepiloghi o titoli di documenti). In questo caso si mantiene basso anche il numero di termini spuri in ogni documento. Il

presente approccio ha diverse caratteristiche appetibili: è facile ed efficiente da implementare e teoricamente comprensibile attraverso le ben comprese proprietà della teoria insiemistica.

Dal punto di vista pratico, una strategia di ricerca booleana recupera i documenti che sono 'veri' per la query. Tale formulazione ha senso solo se le query sono espresse tramite termini indice (o parole chiave) combinati per mezzo dei connettivi logici soliti AND, OR e NOT. Ad esempio, se la query è Q = (K_1 AND K_2) OR (K_3 AND (NOT K_4)) allora la ricerca booleana recupererà tutti i documenti indicizzati da K_1 e K_2, o anche quelli indicizzai da K_3 che *non* sono indicizzati da K_4.

Alcuni sistemi operanti con ricerca booleana permettono di allargare o restringere la ricerca dando accesso ad un dizionario strutturato che, per ogni parola chiave, memorizza le parole chiave collegate che possono essere più generali o più precise. Ad esempio, nella struttura ad albero seguente (Figura 2.18) la parola chiave K_1^1 è contenuta nella parola chiave più generale K_1^0, ma può anche essere divisa nelle quattro parole chiave più precise K_1^2, K_2^2, K_3^2 e K_4^2.

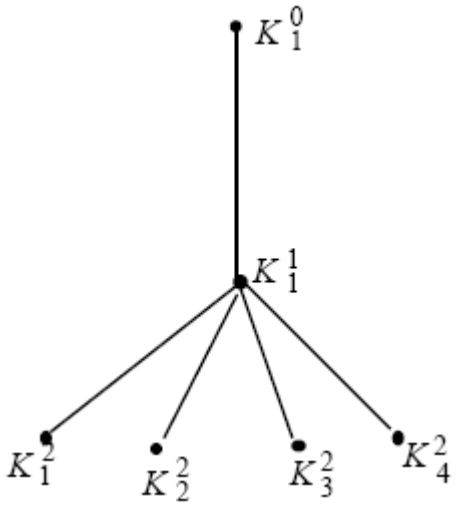


Figura 2.18 Un insieme di parole chiave legate gerarchicamente

Quindi in un sistema interattivo la ricerca può essere facilmente riformulata usando alcuni di questi termini collegati.

Un modo intuitivo di implementare la ricerca booleana è utilizzare il file invertito. Si memorizza una lista per ogni parola chiave nel vocabolario, ed in ogni lista si inseriscono gli indirizzi (o i numeri) dei documenti contenenti tale parola. Per soddisfare una query si eseguono le operazioni insiemistiche, corrispondenti ai relativi connettivi logici, sulle K_i-liste. Ad esempio, se

K_1-lista : D_1, D_2, D_3, D_4

K_2-lista : D_1, D_2

K_3-lista : D_1, D_2, D_3

K_4-lista : D_1

e $Q = (K_1 \text{ AND } K_2) \text{ OR } (K_3 \text{ AND } (\text{NOT } K_4))$

allora per soddisfare la parte (K_1 AND K_2) si intersecano le K_1-lista e K_2-lista, per soddisfare la parte (K_3 AND (NOT K_4)) si sottrae la K_4-lista dalla K_3-lista. Infine la OR è soddisfatta effettuando l'unione dei due insiemi di documenti ottenuti per le singole parti. Il risultato è l'insieme $\{D_1, D_2, D_3\}$, che soddisfa la query ed in cui ogni documento è 'vero' per la query.

Una leggera modifica della ricerca booleana generale consiste nel consentire solo la logica AND, ma tenendo conto del numero di termini che la query ha in comune con un documento. Tale numero è chiamato *livello di co-ordinamento*, e la strategia è spesso chiamata *matching semplice*. Dato che ad ogni livello si può avere più di un documento, si dice che i documenti sono *parzialmente* ordinati tramite i livelli di co-ordinamento.

Per l'esempio precedente con la query $Q = K_1$ AND K_2 AND K_3 si ottiene il seguente punteggio:

Livello di co-ordinamento	
3	D_1, D_2
2	D_3
1	D_4

Infatti il matching semplice può essere visto come l'uso di una funzione di matching primitiva. Per ogni documento D si calcola $| D \cap Q |$, cioè la dimensione della sovrapposizione tra D e Q, in cui ogni elemento è rappresentato come un insieme di parole chiave.

2.3.4.4 Recupero booleano e probabilistico

I sistemi booleani sono ampiamente usati, specialmente da documentaristi esperti; per utenti disinformati e per collezioni di documenti ad ampio raggio, l'approccio booleano è stato quasi totalmente abbandonato in favore degli approcci di *recupero probabilistico*, che assegnano un *rango* ai documenti recuperati come probabilità che i risultati sulla rilevanza ottenuti risolvano la richiesta informativa espressa tramite i termini di ricerca. Ciò in un certo senso fornisce un modello di incertezza e permette di evitare la maggior parte dei problemi di un linguaggio di query specifico, al costo di diminuire la predicibilità, l'efficacia e, in alcuni casi, l'espressività dell'interfaccia. Tuttavia questo è un passo verso l'uso della linguistica, o almeno verso i metodi orientati al linguaggio per elaborare

documenti e query. Ancora una volta, l'intento è di effettuare la parte più grande possibile di analisi in anticipo per semplificare il processo di confronto, la cui flessibilità non è secondaria ma cruciale per definire il funzionamento del linguaggio.

Le differenze tipiche tra i sistemi probabilistici prototipali ed i sistemi booleani sono:

1. l'attribuzione di pesi ai termini tramite l'importanza assunta per una rappresentazione dei documenti, ad esempio tramite alcuni dei metodi presentati nei paragrafi precedenti;
2. un algoritmo di confronto che assegna un rango ai documenti tramite la probabilità della rilevanza;
3. meccanismi di formulazione delle query relativamente liberi.

2.3.4.5 Funzioni di matching

Molte delle strategie di ricerca più sofisticate sono implementate tramite una funzione di matching, cioè una funzione che misura l'associazione tra una query ed un documento.

Ci sono molti esempi di funzioni di matching in letteratura. Uno dei più semplici è quello associato alla strategia di ricerca con matching semplice.

Se M è la funzione di matching, D è l'insieme delle parole chiave che rappresentano il documento, e Q è l'insieme che rappresenta la query, allora:

$$M = \frac{2|D \cap Q|}{|D| + |Q|}$$

è un altro esempio di funzione di matching.

Un'altra funzione comunemente usata, chiamata *correlazione a coseno*, assume che il documento e la query siano rappresentati come vettori numerici in uno spazio *t*-dimensionale, cioè $Q = (q_1, q_2,\ldots, q_t)$ e $D = (d_1, d_2,\ldots, d_t)$, dove q_i e d_i sono pesi numerici associati alla parola chiave *i*. La correlazione a coseno è quindi semplicemente

$$r = \frac{\sum_{i=1}^{t} q_i d_i}{\left(\sum_{i=1}^{t} (q_i)^2 \sum_{i=1}^{t} (d_i)^2 \right)^{\frac{1}{2}}}$$

oppure, nella notazione degli spazi vettoriali con norma euclidea,

$$r = \frac{(Q,D)}{\|Q\|\|D\|} = cos\,\theta$$

dove θ è l'angolo tra i vettori *Q* e *D*.

2.3.4.6 Ricerca seriale

Sebbene sia riconosciuto che le ricerche seriali sono lente, queste ultime sono ancora frequentemente usate come parte di sistemi più grandi. Esse forniscono anche un'utile dimostrazione dell'uso delle funzioni di matching.

Si supponga che vi siano *N* documenti D_i nel sistema, allora la ricerca seriale procede calcolando *N* valori $M(Q, D_i)$ e si determina poi l'insieme dei documenti da recuperare. Ci sono due maniere per fare ciò:

1. si assegna un'opportuna soglia alla funzione di matching, recuperando i documenti al di sopra della soglia e scartando quelli al di sotto. Se T è la soglia, allora l'insieme recuperato B è l'insieme $\{D_i \mid M(Q, D_i) > T\}$;
2. i documenti sono ordinati in ordine crescente del valore della funzione di matching. Si sceglie una posizione d'ordine R come limite superiore, e sono recuperati tutti i documenti al di sotto di essa di modo che $B = \{D_i \mid r(i) < R\}$ dove $r(i)$ è la posizione d'ordine assegnata a D_i. In ogni caso la speranza è che i documenti rilevanti siano contenuti nell'insieme recuperato.

La difficoltà principale connessa a questo tipo di strategia di ricerca è la specifica della soglia o del limite superiore. Questa sarà sempre arbitraria dato che non vi è alcun modo di sapere in anticipo quale valore per ogni query produrrà il recupero migliore.

2.3.4.7 Formulazione della ricerca interattiva

Quando ci si confronta con un sistema di recupero automatico non si è quasi mai in grado di esprimere il proprio bisogno informativo al primo tentativo, ma si preferisce optare per un processo di affinamento guidato dagli errori, in cui si formula la query in base alle informazioni che il sistema può fornire sulla query stessa. I tipi di informazioni che in genere si usano per la riformulazione delle query sono:

1. la frequenza di occorrenza nella base di dati dei termini di ricerca;
2. il numero di documenti che sarebbero recuperati dalla query;
3. termini alternativi e correlati a quelli usati nella ricerca;
4. un piccolo campione delle citazioni che sarebbero recuperate;
5. i termini usati per indicizzare le citazioni.

Tutto ciò può essere fornito da un sistema di recupero interattivo durante una sessione di ricerca. Se si scopre che uno dei termini di ricerca compare molto frequentemente lo si può rendere più specifico scegliendo tra le opzioni ricavate dalla consultazione di un dizionario gerarchico. In modo simile, se la query recuperasse troppi documenti la si potrebbe rendere più specifica.

Il campione di citazioni e la loro indicizzazione dà un'idea di quale tipo di documenti possano essere recuperati, e di conseguenza anche un'idea dell'efficacia dei termini di ricerca nell'esprimere il bisogno informativo: la query può essere modificata in base al recupero di tale campione. Tale processo di modifica può essere descritto come una sorta di *feedback*, concetto affrontato nel prossimo paragrafo.

2.3.4.8 Feedback

La parola feedback è usata di norma per descrivere il meccanismo con il quale un sistema può migliorare le proprie prestazioni in un certo lavoro tenedo conto delle prestazioni passate. In altre parole, un semplice sistema ingresso-uscita riporta le informazioni in uscita in modo che queste possano essere usate per migliorare le prestazioni al prossimo ingresso. La nozione di feedback è ben consolidata nei sistemi biologici e di controllo automatico, e nell'information retrieval è stata usata con risultati degni di nota.

Si consideri ora una strategia di recupero che sia stata implementata tramite una funzione di matching *M*. Inoltre, si supponga che sia la query *Q* che i documenti *D* siano vettori *t*-dimensionali a componenti reali, dove *t* è il numero di termini indice. Per semplicità si prenderà in considerazione solo la ricerca seriale.

Lo scopo di ogni strategia di recupero è recuperare i documenti rilevanti A e scartare quelli non rilevanti $\bar{A}$. Sfortunatamente la rilevanza è definita rispetto all'interpretazione *semantica* della query da parte dell'utente: dal punto di vista del sistema di recupero questo tipo di formulazione potrebbe non essere idoneo. Una formulazione ideale sarebbe quella che recupera solo i documenti rilevanti. Nel caso di una ricerca seriale il sistema recupererà tutti i documenti D per i quali $M(Q, D) > T$ e scarterà qualunque D per il quale $M(Q, D) \leq T$, dove T è una soglia specifica. Quindi se M è la funzione di correlazione a coseno, ad esempio

$$M(Q,D) = \frac{(Q,D)}{\|Q\|\|D\|} = \frac{1}{\|Q\|\|D\|} \cdot (q_1 d_1 + q_2 d_2 + ... + q_t d_t)$$

la procedura decisionale

$$M(Q, D) - T > 0$$

corrisponde ad una funzione discriminante lineare usata per separare linearmente i due insiemi A e $\bar{A}$ in R^t. Si supponga per il momento che A e $\bar{A}$ siano conosciuti in anticipo, allora la formulazione corretta della query Q_0 sarebbe quella per cui

$$M(Q_0, D) > T \qquad \text{per } D \in A$$

e

$$M(Q_0, D) \leq T \qquad \text{per } D \in \bar{A}$$

La proprietà interessante è che partendo con un Q qualunque lo si può aggiustare iterativamente usando le informazioni di feedback in modo che possa convergere a Q_0. Esiste un teorema che afferma che data l'esistenza di Q_0 è possibile trovare una procedura iterativa che permetterà di far convergere Q a Q_0 in un numero finito di passi.

Tale procedura è detta *a correzione d'errore ad incremento fisso*.

Ne consegue che:

$$Q_i = Q_{i-1} + cD \qquad \text{se} \quad M(Q_{i-1}, D) - T \leq 0 \quad \text{e} \quad D \in A$$

$$Q_i = Q_{i-1} - cD \qquad \text{se} \quad M(Q_{i-1}, D) - T > 0 \quad \text{e} \quad D \in \bar{A}$$

c è l'incremento della correzione: il suo valore è arbitrario e quindi è solitamente posto pari ad uno. In pratica può essere necessario controllare diverse volte l'insieme dei documenti prima che sia ottenuto l'insieme corretto dei pesi che separano linearmente A e $\bar{A}$ (sempre che una soluzione esista).

La situazione reale però non è così semplice: gli insiemi A e $\bar{A}$ non sono conosciuti in anticipo, infatti A è l'insieme che si cerca di recuperare. Comunque, data una formulazione di query Q ed i documenti da essa recuperati, si può informare il sistema su quali dei documenti recuperati sono rilevanti o meno, ed il sistema può di conseguenza modificare automaticamente Q in modo da poter analizzare correttamente i documenti presi in considerazione. Si assume che ciò migliorerà il recupero all'esecuzione successiva in virtù del fatto che la sua prestazione è migliorata su un campione.

Ciò non è ancora tutto. Spesso è difficile fissare la soglia *T* in anticipo in maniera che i documenti siano ordinati in uscita per valori di matching decrescenti. Ora è più difficile definire cosa si intende per formulazione ideale di query. Nella propria tesi, Rocchio definì la query ottima Q_0 come quella che massimizza la quantità :

$$\Phi = \frac{1}{|A|}\sum_{D\in A} M(Q,D) - \frac{1}{|\bar{A}|}\sum_{D\in \bar{A}} M(Q,D)$$

Se si assume che *M* sia la funzione coseno (*Q*, *D*) / ||*Q*|| ||*D*|| allora si può mostrare facilmente che Φ è massimizzata da

$$Q_0 = c\left(\frac{1}{|A|}\sum_{D\in A}\frac{D}{\|D\|} - \frac{1}{|\bar{A}|}\sum_{D\in \bar{A}}\frac{D}{\|D\|}\right)$$

dove *c* è una costante di proporzionalità arbitraria.

Se le sommatorie sono effettuate, piuttosto che su *A* e $\bar{A}$, su $A \cap B_i$ e $\bar{A} \cap B_i$ dove B_i è l'insieme dei documenti recuperati all'*i*-esima iterazione, allora si ha una formulazione della query ottimale per B_i, un sottoinsieme della collezione dei documenti. Si aggiunge ora tale vettore alla formulazione della query all'*i*-esimo passo e si ottiene:

$$Q_{i+1} = w_1 Q_i + w_2\left[\frac{1}{|A\cap B_i|}\sum_{D\in A\cap B_i}\frac{D}{\|D\|} - \frac{1}{|\bar{A}\cap B_i|}\sum_{D\in \bar{A}\cap B_i}\frac{D}{\|D\|}\right]$$

dove w_1 e w_2 sono coefficienti di pesatura. Salton usò una versione leggermente modificata, in cui la differenza più importante consiste nella presenza di un'opzione per generare Q_{i+1} da Q_i, o Q (la query originale). L'effetto di tali aggiustamenti può essere riassunto dicendo che la query è automaticamente modificata in maniera tale che siano attribuiti pesi maggiori ai termini indice nei documenti rilevanti recuperati (promozione) e pesi minori ai termini indice nei documenti non rilevanti (retrocessione).

Gli esperimenti hanno mostrato che il feedback di rilevanza può essere molto efficace. Sfortunatamente il raggio d'azione dell'efficacia è piuttosto difficile da stimare, dato che è difficile separare il contributo ad una maggiore efficacia del recupero prodotta quando i singoli documenti aumentano di rango dal contributo prodotto quando sono recuperati *nuovi* documenti. Ovviamente quest'ultimo caso è quello che interessa di più.

Per concludere, in generale sembra che l'implementazione della tecnica del feedback di rilevanza su un sistema ufficiale sia problematica. Non è infatti chiaro come si possa asserire la rilevanza, o non rilevanza, di un documento da informazioni scarse come le citazioni. In un tale sistema è semplice generare dei riepiloghi da visualizzare ma è probabile che si avrà bisogno di sfogliare gli stessi documenti recuperati per determinare la loro rilevanza.

Capitolo 3 – Metodo

3.1 Introduzione

Nel presente capitolo saranno trattati in maggiore dettaglio i metodi e le strategie utilizzate per la realizzazione di un sistema per la verifica dei requisiti.

Come detto nel capitolo introduttivo, ci si trova in un contesto nel quale è presente un progetto software realizzato e funzionante che ha la necessità di essere analizzato ai fini di una maggiore comprensione dello stesso dal punto di vista progettuale, per cogliere eventuali imprecisioni o omissioni nella sua specifica e rendersi conto del suo stato di avanzamento nella realizzazione. In pratica il sistema oggetto del presente lavoro intende affrontare due aspetti cruciali del processo di manutenzione del software:

1. verificare se e quanto i requisiti definiti nella fase di progetto sono stati in realtà affrontati ed implementati;
2. fornire dei suggerimenti per l'individuazione di eventuali requisiti duplicati, impliciti ed emergenti.

Da una prospettiva metodologica, il primo punto viene affrontato cercando di individuare delle relazioni logiche tra i requisiti considerati e dei feedback rilasciati dagli utenti dell'applicazione. Tali feedback denotano eventuali malfunzionamenti o imprecisioni e sono espressi sotto forma di report testuali: sono prodotti tipicamente nella fase di testing dell'applicazione (anche se il processo continua praticamente per tutto il suo ciclo di vita) e costituiscono i cosiddetti *bug* del programma. La presenza di un bug relativo ad un aspetto

dell'applicazione legato ad un suo requisito denota che quest'ultimo è stato affrontato in fase di realizzazione.

Il secondo punto è teso a fornire un supporto agli ingegneri della manutenzione su una fase molto delicata: la *reingegnerizzazione dei requisiti*. Questa verte sulla valutazione dell'effettiva *completezza* e *minimalità* dell'insieme dei requisiti attualmente definiti rispetto agli obiettivi che l'applicazione dovrebbe perseguire. Si ricorda che con il termine *completezza* si intende la proprietà per cui l'insieme esprime tutte le necessità che l'applicazione deve soddisfare (e non solo quelle richieste dal committente) per poter essere considerata completa nell'ambito in cui essa opera; un'analisi approfondita secondo questo punto di vista può portare alla definizione di due nuove categorie di requisiti:

- *requisiti impliciti*: derivano dalla natura del sistema, che impone vincoli o funzionalità non prevedibili in una fase astratta come quella progettuale (ad esempio affidabilità, disponibilità, sicurezza, ecc.);
- *requisiti emergenti*: come dice la parola stessa, emergono da una maggiore comprensione del problema. Spesso l'ingegnere dei requisiti può non afferrare completamente tutti gli aspetti della problematica assegnatagli, o a volte anche lo stesso committente può non esserne del tutto consapevole. Di solito si tratta di nuove funzionalità non previste in fase di progetto.

Il sistema cercherà di essere di supporto analizzando i report che non hanno legami con i requisiti attualmente definiti, e che quindi possono indicarne implicitamente la richiesta di nuovi.

La *minimalità* invece è la proprietà dell'insieme di contenere tutti e soli i requisiti che esprimono le richieste che il progetto deve soddisfare. Se alcuni di questi sono legati molto spesso agli stessi report ci si potrebbe trovare di fronte a *requisiti ridondanti* (i quali

trattano degli stessi argomenti), ed alcuni di essi potrebbero essere eliminati. Il sistema, mostrando a quali requisiti ogni report è legato, può essere d'aiuto in questo senso.

3.2 Vista d'insieme

L'analisi dei requisiti per conseguire gli obiettivi visti nel paragrafo precedente verte concettualmente sulle fasi seguenti:

1. organizzazione dei requisiti software e dei report sui bug tramite tecniche di information retrieval per un rapido ed efficace trattamento;
2. creazione di un'apposita rappresentazione di tali documenti idonea al confronto;
3. verifica delle associazioni esistenti tra i requisiti software ed i report sui bug, e stima del relativo livello;
4. riorganizzazione dei report non associati e loro analisi, come supporto alla scoperta di eventuali requisiti emergenti.

Rimandando la trattazione dei dettagli puramente tecnici ed implementativi al capitolo successivo, di seguito sarà fornita una trattazione metodologica dei punti precedenti.

3.3 Documenti ed information retrieval

All'inizio si hanno a disposizione da una parte un certo numero di documenti contenenti ognuno un diverso requisito funzionale, e dall'altra una certa quantità (sicuramente maggiore della prima) di documenti contenenti ognuno un report ufficiale su un bug.

Affinché tali documenti possano essere confrontati tra di loro, occorre effettuarne delle trasformazioni in modo da riportarli in una forma più pratica.

Uno strumento adatto al trattamento di documenti testuali per l'estrazione di informazioni è un sistema di *information retrieval* (o in breve IRS). Come si è visto ampiamente nel capitolo sullo stato dell'arte, l'information retrieval è una disciplina (connessa ad un vasto insieme di tecniche e strategie) mirata all'organizzazione di un insieme di testi allo scopo di rendere quanto più efficaci ed efficienti possibile le ricerche su particolari argomenti (o perlomeno su particolari parole). Tale organizzazione è raggiunta attraversando una fase importantissima e delicata denominata *indicizzazione*. Questa è composta da tre sottofasi affrontate sequenzialmente:

1. conversione del testo;
2. analisi del testo;
3. creazione dell'indice.

Le operazioni necessarie all'indicizzazione sono riassunte in Figura 3.1:

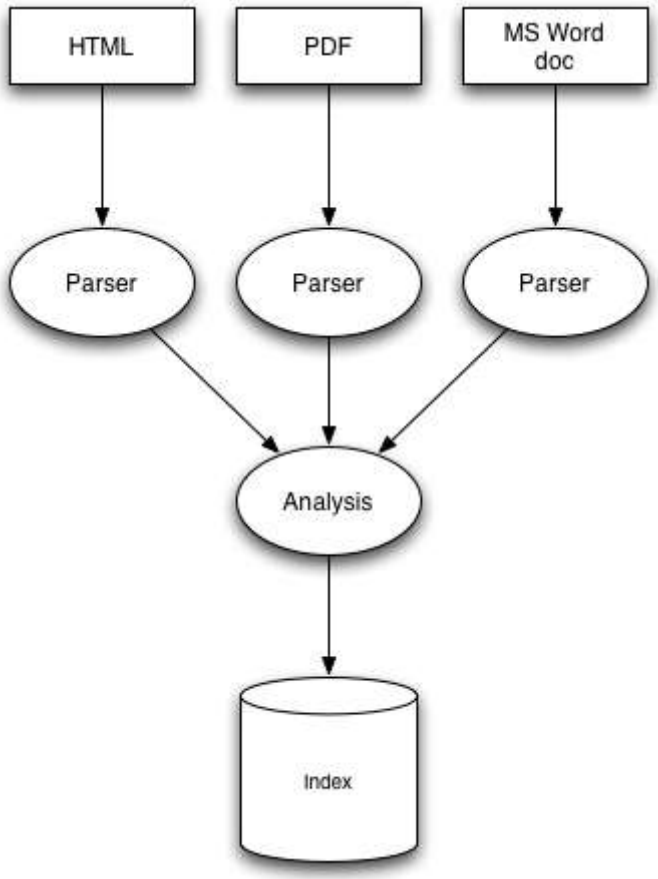


Figura 3.1 Processo di indicizzazione

3.3.1 Conversione del testo

La fase di conversione del testo prevede la trasformazione del documento, che può essere sotto forma di file HTML, PDF, Microsoft Word, ecc., in una forma particolare, solitamente testo piatto senza alcun tipo di formattazione. Purtroppo però non è immediato estrarre il contenuto testuale da formati di documenti codificati e/o crittografati (come PDF e documenti Microsoft Word), ed anche quelli apparentemente semplici e leggibili con un qualunque editor di testo ASCII (come XML o HTML) possono dare diversi problemi, dato che si ha bisogno comunque di scartare i tag interni. Le diverse situazioni possibili hanno in genere soluzioni diverse, che devono essere valutate attentamente.

3.3.2 Analisi del testo

Superato questo scoglio, si ha a disposizione il testo pronto per l'elaborazione. Prima di memorizzarlo in qualche modo, è buona norma effettuarne un'analisi per migliorare le prestazioni del sistema (sia in termini di memoria che di tempo di elaborazione) e potenziarne le capacità di estrazione delle informazioni rilevanti.

L'*analisi* è il processo che permette di ricavare dal testo i termini, cioè gli elementi primitivi utilizzati per l'indicizzazione; l'*analizzatore* è l'esecutore della procedura di analisi. Un analizzatore ricava i termini eseguendo certe operazioni sul testo, come l'estrazione delle singole parole, la rimozione della punteggiatura e degli accenti, la trasformazione in caratteri minuscoli (*normalizzazione*), l'eliminazione delle stopword, lo stemming delle parole e l'uso di una forma comune per i termini sinonimi. Questi ultimi tre aspetti sono i più delicati e cruciali, e di seguito se ne fornisce qualche dettaglio teorico in più.

Eliminazione delle stopword

La presenza di parole troppo comuni e con scarso potere di discriminazione all'interno del testo, come ad esempio gli articoli, appesantisce molto l'elaborazione e la forma finale del testo. Tali parole vengono quindi eliminate prima di passare alle altre fasi, in base ad un elenco fissato di stopword (*stoplist*) fortemente dipendente dal formalismo comune dei documenti e dalla lingua utilizzata. L'applicazione oggetto della tesi si avvale di due stoplist (rispettivamente per la lingua inglese e italiana) di termini comuni di uso generico con l'aggiunta di quelli legati al processo di manutenzione software, come bug, issue, ecc.

Stemming delle parole

Lo *stemming* è un processo per l'eliminazione delle forme morfologiche e flessive più comuni delle parole; in altri termini, esso riduce le varie forme che può assumere una parola alla radice comune. Per la maggior parte dei linguaggi occidentali, e questo vale anche per la lingua inglese, le parole tendono a rimanere costanti all'inizio e a variare nella parte finale. La rimozione, o la sostituzione, di queste parti finali (o suffissi) è proprio ciò che si definisce "stemming".

La scelta di una tecnica dipende dall'ambito di utilizzo nel quale questa operazione si svolge. Solitamente, comunque, un programma di rimozione dei suffissi (*stemmer*) contiene una lista di suffissi analizzati e, per ognuno, il criterio in base al quale esso può essere rimosso o sostituito.

Dal punto di vista della valutazione delle prestazioni si può affermare che l'operazione di stemming permette l'incremento del richiamo, in quanto riduce le parole ad una comune radice morfologica; d'altra parte, però, esso tende a ridurre la precisione qualora non si ponga attenzione al rischio di ridurre ad una radice comune termini che non sono correlati in significato.

Esistono diversi algoritmi di stemming in letteratura. Nel prossimo capitolo sarà illustrato in dettaglio quello utilizzato come soluzione al problema.

Fusione dei termini sinonimi

Infine, un ultimo metodo per aumentare l'associazione tra termini diversi è quello di rintracciare l'eventuale proprietà di sinonimia di una loro coppia. A questo scopo torna utile un dizionario di parole semanticamente legate, detto *thesaurus*, da consultare per generare tutti i sinonimi di una data parola (classe). Tutte le parole appartenenti alla stessa classe sono poi sostituite con un particolare rappresentante di quest'ultima (ad esempio la

parola con occorrenza maggiore nell'insieme dei documenti): ciò fa in modo che una successiva ricerca di testi contenenti una certa parola dia un maggior numero di risultati.
In fase di progetto dell'applicazione si è però scelto di non avvalersi di tale tecnica per via di diversi svantaggi che questa comporta:

- elevato tempo di elaborazione per la classificazione delle parole e la loro successiva sostituzione con il rappresentante della classe;
- maggiore tempo di risposta nelle ricerche per via della possibile riapplicazione del metodo al termine oggetto della query;
- possibili errori di valutazione per particolari parole polisemiche (come "pesca"); per evitarli sarebbe necessaria quindi l'esatta definizione dell'argomento del testo.

3.3.3 Creazione dell'indice

Dopo che il testo è stato analizzato, è pronto per essere aggiunto ad una particolare struttura dati, l'*indice*, adatta ad una memorizzazione ed un recupero efficienti e veloci. La creazione e la gestione di tale struttura sono tipicamente parte integrante del sistema di information retrieval scelto. Le funzionalità utilizzate per realizzare l'indicizzazione dei requisiti software e dei report sui bug sarà affrontata nel prossimo capitolo.

3.4 Rappresentazione vettoriale dei documenti

È stato precedentemente detto che lo scopo principale dell'applicazione è quello di eseguire dei confronti tra i documenti testuali rappresentanti i requisiti funzionali e i report

sui bug. Affinché tali confronti possano essere eseguiti risulta pratico esprimere i documenti in una forma particolare, detta *rappresentazione vettoriale*.

Se l'insieme di tutti i testi presi in considerazione contiene in totale *n* termini diversi, dove un termine è una parola sottoposta al processo di analisi visto prima, allora è possibile pensare al documento *j*-esimo come ad un vettore in uno spazio *n*-dimensionale, avente la rappresentazione seguente:

$$\vec{d}_j = (w_{1,j}, w_{2,j}, ..., w_{n,j})$$

in cui $\vec{d}_j$ è il vettore *j*-esimo e $w_{i,j}$ è il peso assegnato all'*i*-esimo termine nel *j*-esimo documento. Se il termine *i*-esimo non compare nel *j*-esimo documento si avrà $w_{i,j} = 0$. Graficamente quindi il documento può essere rivisto come in Figura 3.2, considerandolo per semplicità composto da soli tre termini e quindi contenuto in uno spazio vettoriale tridimensionale:

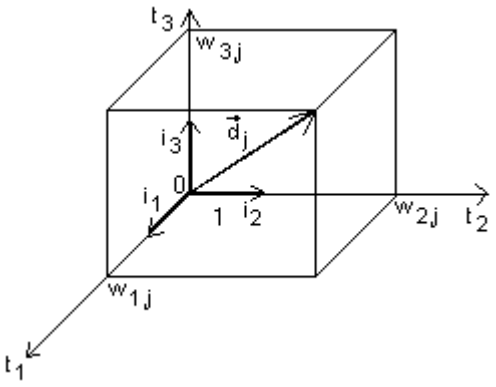


Figura 3.2 Rappresentazione vettoriale di un documento

Esistono svariati metodi per assegnare pesi ai termini, fondati su considerazioni di importanza semantica: di solito si fa in modo di attribuire pesi maggiori a termini meno comuni, aventi una buona potenza di discriminazione, e di rendere quasi irrilevanti gli altri termini. Nella situazione in esame tutte le parole che trattano di un certo requisito sono

contenute nel testo del requisito stesso, per cui non vi sarà bisogno di effettuare particolari operazioni di pesatura (anche perché, come si è detto nel capitolo precedente, di solito i miglioramenti apportati da tali soluzioni non sono commisurati allo sforzo progettuale ed elaborativo); al contrario, si è preferito utilizzare come valore di peso semplicemente il numero di occorrenze del termine considerato nel particolare documento ($f_{i,j}$).

Questo metodo di pesatura però è da considerarsi di tipo assoluto, perché è fortemente dipendente dalla lunghezza del documento: testi più lunghi produrranno quasi sicuramente pesi maggiori per un termine rispetto a quelli degli altri testi più brevi per lo stesso termine, anche se magari i primi trattano relativamente poco l'argomento espresso dal termine considerato. Ciò significa che i testi non saranno direttamente confrontabili tra loro perché utilizzano metriche diverse: si rende così necessaria una fase di *normalizzazione* dei pesi calcolati. Tuttavia nell'algoritmo utilizzato non è stato necessario applicare subito tale aggiustamento in quanto, come si vedrà in seguito, questo è stato integrato nella fase di valutazione delle associazioni tra i requisiti ed i report disponibili.

Riepilogando, dopo la fase di indicizzazione dei documenti (requisiti e report) il sistema crea per ognuno di essi la propria rappresentazione vettoriale utilizzando per i termini pesi derivati dal conteggio delle occorrenze senza normalizzazione.

3.5 Verifica delle associazioni tra requisiti e report

Le fasi illustrate precedentemente sono laboriose, ma servono solo a rappresentare le informazioni in una forma comoda per le elaborazioni. Ora si affronterà la parte principale del capitolo, cioè la fase di confronto dei documenti raccolti per ottenere una stima del livello di trattamento e svolgimento dei requisiti software di un particolare progetto.

Questo è il passo in cui opera effettivamente ed estensivamente il metodo studiato per il presente lavoro di tesi, e che fa il maggior uso di risorse di elaborazione del sistema.

La stima cercata è ottenuta attraversando tre fasi in successione:

1. calcolo del livello di similitudine tra ogni requisito ed ogni report;
2. scelta dei report con maggiore similitudine, per ogni requisito, per la conduzione della stima;
3. stima del livello di trattazione globale di ogni requisito.

3.5.1 Calcolo del livello di similitudine tra requisiti e report

La stima sul livello di trattazione di un requisito è ovviamente legata a quanto ogni report tratta nel proprio testo l'argomento di quel requisito: è dunque necessario preliminarmente verificare il grado di associazione (se esiste) tra ogni requisito ed ogni report.

Come accennato prima, ciò è ottenuto calcolandone il livello di similitudine delle loro rappresentazioni vettoriali; infatti, immaginando i testi nella loro forma vettoriale, si comprende subito che la distanza vettoriale fra essi può essere considerata un buon indicatore: quanto più i vettori rappresentativi sono vicini nello spazio vettoriale creato tanto più utilizzano gli stessi termini (e nelle stesse quantità relative) e quindi tanto più presumibilmente trattano dello stesso argomento. Per chiarire questo aspetto si pensi a due testi *A* e *B* espressi in forma vettoriale, uno con *n* e l'altro con *m* termini diversi (e quindi definiti rispettivamente in uno spazio *n*- e *m*-dimensionale): se per semplicità di rappresentazione grafica si considera una loro proiezione su uno spazio tridimensionale, ad esempio relativo ai termini "mamma", "bella" e "brava", si nota che quanto più l'angolo

tra le loro rappresentazioni diminuisce tanto più essi trattano in uguale misura del fatto che "la mamma è bella e brava" (Figura 3.3):

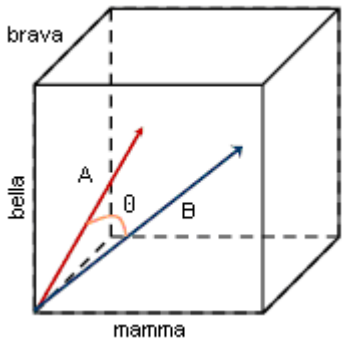


Figura 3.3 Distanza vettoriale tra due vettori

Estendendo tale ragionamento a spazi vettoriali a dimensioni maggiori (al limite pari alle dimensioni dei testi) si comprende facilmente che in questo modo è possibile stabilire quanto due testi trattano dello stesso argomento in base alla distanza angolare tra i due vettori.

Dal punto di vista matematico, la distanza vettoriale è misurata dall'angolo θ compreso tra i due vettori, e tale angolo si può determinare operativamente tramite la formula

$$\theta = cos^{-1} \frac{(A,B)}{\|A\|\|B\|}$$

In realtà, è molto utile usare al posto dell'angolo θ il suo coseno, cioè

$$c = \frac{(A,B)}{\|A\|\|B\|}$$

perché questa è l'espressione del *coefficiente di correlazione a coseno* (rif. Paragrafo 2.6.5). Quindi in definitiva si calcolano le quantità

$$c_{A,B} = \frac{\sum_{i=1}^{n} a_i b_i}{\left(\sum_{i=1}^{n} (a_i)^2 \sum_{i=1}^{n} (b_i)^2 \right)^{\frac{1}{2}}}$$

dove *A* e *B* sono i vettori rappresentanti il requisito e il report (o viceversa), e a_i e b_i sono le occorrenze del termine *i*-esimo all'interno del loro testo.

Infine si noti che la formula è composta dal rapporto tra il prodotto scalare delle rappresentazioni vettoriali ed il prodotto dei moduli dei singoli vettori. Tale rapporto assicura la normalizzazione dell'operazione: il valore di *c* sarà una quantità compresa tra uno e zero, dove uno indica la totale dissimilarità e zero rappresenta la similitudine perfetta. Ecco spiegata la ragione per cui non è stato necessario normalizzare anticipatamente le singole rappresentazioni dei documenti.

3.5.2 Scelta dei report significativi

Si hanno a disposizione tutti i livelli di associazione tra requisiti e report, e per ogni requisito il livello di trattazione stimato sarà dato da delle operazioni effettuate su una certa combinazione di questi. Sorge quindi il problema di capire quanti e soprattutto quali dei report dovranno far parte dell'insieme dei documenti coinvolti nel calcolo.

Proprio come per i motori di ricerca, ciò che si deve ottenere è un buon compromesso tra i livelli di *precisione* (cioè il rapporto tra il numero di report effettivamente significativi tra quelli scelti ed il numero totale dei report scelti) e di *richiamo* (cioè il numero di report effettivamente significativi tra quelli scelti ed il numero di tutti i report significativi), che si sanno essere inversamente proporzionali (rif. Paragrafo 2.3.1): aumentare il numero di documenti presi in considerazione aumenterebbe presumibilmente il richiamo, ma farebbe

diminuire la selettività e quindi la precisione (e viceversa). Dato che la valutazione rigorosa di tali parametri richiederebbe l'ausilio di un oracolo (cioè la conoscenza a priori di quanti e quali report si riferiscono ad un particolare requisito), si è scelto di utilizzare la strategia *top-n*: per ogni requisito si costruisce una graduatoria dei report ordinati per valori di similitudine con questo decrescenti, e si prendono in considerazione i primi *n* documenti di tale graduatoria. *n* è lasciato alla scelta dell'utente, ed è di solito definitivamente fissato da questi dopo aver condotto diverse prove e valutato i relativi risultati. Questa strategia permette, indipendentemente dal numero, di includere sempre nell'insieme dei report scelti quelli che presumibilmente trattano in maniera più estesa il requisito considerato, e quindi cerca di mantenere la precisione ai livelli massimi possibile.

In più, dualmente per ogni report si costruisce una graduatoria dei requisiti ordinati per gli stessi valori di similitudine già calcolati. Ma diversamente da prima qui non si usa la strategia top-*n*, bensì si prendono in considerazione solo le associazioni precedentemente stabilite. A prima vista ciò può far pensare ad un'inutile ridondanza di rappresentazioni, ma in realtà queste sono preziose informazioni perché consentono di individuare due categorie particolari di report:

- report con associazioni multiple;
- report non associati.

Da un'analisi dei report nella prima categoria potrebbe scaturire la presenza di requisiti che si compenetrano: se infatti un certo numero di report su dei bug ha una forte associazione con due particolari requisiti, allora è altamente probabile che nella pratica tali requisiti si riferiscano ad un'unica funzionalità e possano quindi essere fusi, anche solo parzialmente.

I report nella seconda categoria, invece, possono essere il sintomo del bisogno della definizione di requisiti emergenti. Questa possibilità viene analizzata più avanti nel Paragrafo 3.6.

3.5.3 Stima del livello di trattazione dei requisiti

La fase finale della parte principale dell'applicazione è stimare il livello globale di trattazione di ogni requisito, prendendo in considerazione solo i top-*n* report selezionati. Dal punto di vista prettamente teorico, nulla vieta di pensare all'insieme dei report scelti come ad un unico grande documento testuale contenente tutti questi sequenzialmente uno dopo l'altro; esso sarebbe composto dalla somma di tutte le parole dei singoli documenti, ed avrebbe una dimensione pari alla somma delle loro singole dimensioni. Adottando questa strategia si riesce quindi facilmente a fondere in un'unica entità l'intero insieme, limitandosi ad una semplice somma vettoriale di *n* vettori (Figura 3.4):

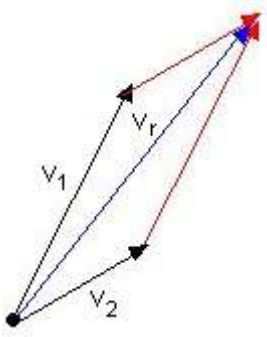


Figura 3.4 Somma di due vettori

Per ricavare la stima desiderata è quindi sufficiente calcolare la similitudine tra la rappresentazione vettoriale del requisito considerato ed il relativo vettore risultante dei report appena illustrato, utilizzando ancora una volta la funzione di correlazione a coseno vista nel Paragrafo 3.5.1.

Il valore di similitudine appena calcolato, unitamente a quelli dei singoli top-*n* report, possono essere presentati all'utente (magari espressi in percentuale) e costituiscono l'analisi quantitativa sullo svolgimento dei requisiti.

3.6 Supporto alla ricerca di requisiti emergenti

Con il completamento della fase precedente si è raggiunto lo scopo principale del sistema; quasi certamente tale fase ha scartato un certo numero di report a causa di uno dei due motivi seguenti:

1. scelta del parametro *n* troppo basso nella definizione del top-*n*;
2. scarsa similitudine con un qualsiasi requisito.

Supponendo che siano state evitate cause del primo tipo, e cioè supponendo che tutti i report rilevanti siano stati presi in considerazione, sorge allora il bisogno di analizzare in maniera più approfondita tali "report scarto".

Come visto nel Capitolo 2, la teoria sull'ingegneria dei requisiti fa una distinzione tra diversi tipi di requisiti software; in particolare esiste una categoria caratterizzata dai *requisiti emergenti*, definiti come "requisiti che emergono quando aumenta la comprensione del problema". Si è visto che il ciclo di vita del software non è composto da rigide fasi con un ordine cronologico definito, ma esistono diversi modelli schematici in cui la fase di progetto resta viva durante il processo di implementazione e testing, ma molto più frequentemente anche nel processo di manutenzione. È in questo frangente, a partire dalle problematiche più chiare che si mostrano agli utenti dell'applicazione e dalle più accurate analisi condotte dagli ingegneri della manutenzione, che si può prefigurare la possibilità di definire nuovi requisiti, legati a nuove funzionalità che l'applicazione dovrebbe supportare per soddisfare i bisogni dell'utente o già supporta per merito del processo di manutenzione: i report scartati, a prima vista inconsistenti e fuori luogo, possono essere preziose fonti di informazione per la loro definizione.

La loro valutazione è una questione molto delicata e molte volte soggettiva, per cui non può essere automatizzata ed è demandata all'ingegnere dei requisiti. Il sistema può però

dare una mano in tal senso, fornendo utili informazioni sulla distribuzione di particolari parole chiave al loro interno per permettere un orientamento più semplice.

Si è quindi deciso di effettuare un'ulteriore indicizzazione unicamente dei report non associati, e da questo insieme estrarre un certo numero di parole ritenute più significative delle altre, cioè quelle ad occorrenza totale maggiore, e suggerirle allo sviluppatore. A quest'ultimo viene poi data la possibilità di richiedere maggiori informazioni su una di tali parole oppure su una parola direttamente suggerita dall'utente. Sfruttando le potenzialità del sistema di indicizzazione e ricerca, sono quindi mostrati all'utente tutti i report che la contengono, con le relative frequenze di occorrenza, ordinati in ordine decrescente di quest'ultima.

Capitolo 4 – Implementazione

4.1 Introduzione

Nel capitolo precedente è stato analizzato il metodo con cui è stato sviluppato il tool "DevelopAider". Qui si vedranno gli aspetti prettamente pratici ed implementativi relativi alla sua realizzazione.

Prima di dare uno sguardo più approfondito alle diverse parti componenti, si ritiene utile dare una breve definizione della struttura globale. Essa è stata realizzata progettando ed implementando sei classi specializzate:

- *DevelopAider*, da cui si istanzia l'oggetto contenente tutti i componenti dell'interfaccia grafica e la gestione dell'intero processo di analisi dei requisiti;
- *IndexUtils*, contenente tutti i metodi specializzati per le operazioni di indicizzazione e recupero dei dati;
- *RepresentationUtils*, responsabile della creazione delle rappresentazioni vettoriali dei documenti considerati;
- *AssociationUtils*, preposta alla verifica sull'esistenza di associazioni tra i requisiti ed i report sui bug ed alla loro analisi quantitativa, nonché alla creazione di opportune strutture logiche per la loro rappresentazione;
- *ValutationUtils*, atta alla valutazione finale sul grado di trattamento dei requisiti ed al supporto per la scoperta di quelli emergenti;
- *Quicksort*, tramite cui si realizza l'ordinamento dei risultati ottenuti per una presentazione all'utente più efficace.

L'oggetto di tipo *DevelopAider* contiene al suo interno tutte le strutture dati necessarie alla memorizzazione delle risorse e dei risultati delle computazioni; esso è quindi il nucleo del tool, e per le operazioni specializzate si avvale dei metodi statici delle altre classi di supporto applicati alle proprie strutture (così come si utilizzano i metodi della classe *Arrays* del package *java.util* per effettuare operazioni particolari sugli array).

Rispettando le prescrizioni e le regole di buona progettazione dettate dall'ingegneria del software, sono stati preparati i diagrammi secondo lo standard *UML* (Unified Modeling Language) e la documentazione testuale ritenuti utili per una definizione formale della struttura dell'applicazione e dell'interazione tra i suoi componenti, e per un elenco delle sue funzionalità ed il loro corretto uso da parte dell'utente. Tale materiale è riportato per comodità di riferimento in Appendice A.

Di seguito si espliciteranno le particolari strutture e librerie utilizzate nella stesura del codice applicativo, seguendo per quanto possibile il filo logico usato nel Capitolo 3. Infine si presenterà un esempio d'uso basato su dati reali, in modo da realizzare una presentazione guidata che possa permettere all'utente di familiarizzare con l'interfaccia del tool.

4.2 Documenti ed information retrieval

Nell'omonimo paragrafo del capitolo precedente si è parlato della necessità di indicizzare i documenti oggetto dell'analisi prima di elaborarli, e delle problematiche connesse. Per ognuna di esse vale la pena esplicitare in che modo sono state affrontate, ricordando che il codice che realizza le funzionalità è contenuto nella classe *IndexUtils*.

4.2.1 Conversione del testo

L'applicazione deve trattare documenti che esprimono i requisiti funzionali di un progetto ed i report sui bug pubblicati su un server Bugzilla[1]. Dato che dall'analisi del testo risulta difficile delimitare esattamente i diversi requisiti nel caso siano contenuti in un unico file fisico, si è scelto di separarli preventivamente e memorizzarli in file diversi. Invece i report sono naturalmente distinti l'uno dall'altro, e l'assegnazione di un file per ognuno di essi è implicita.

I requisiti fanno tipicamente parte del documento di progetto SRS, che per la sua natura discorsiva e di presentazione è usualmente scritto tramite elaboratori di testo avanzati, il più famoso dei quali è Word della Microsoft: l'applicazione assumerà quindi che il testo dei requisiti sia contenuto in appositi file Microsoft Document (con estensione .DOC). Dall'altra parte, i report pubblicati da Bugzilla sono presentati via web in HTML, ma possono essere visualizzati (e salvati) anche nel più comodo formato XML: è proprio questo il tipo di file che si sottoporrà all'applicazione per la loro analisi.

Il problema iniziale è dunque "tradurre" i testi codificati in queste due maniere nel formato in testo piano comprensibile da Java.

Conversione da Microsoft Word

Per la conversione da formato Word ci si è avvalsi di una libreria Java chiamata *Textmining.org*[2], sviluppata da Ryan Ackley, e giunta alla versione 0.4 – che è quella utilizzata dall'applicazione – nel 2004. Tale libreria si presenta sotto forma di package java (*tm-extractors-0.4.jar*) ed è immediatamente utilizzabile con una semplice importazione:

[1] http://www.bugzilla.org
[2] http://www.textmining.org

```
import org.textmining.text.extraction.*;
```

La libreria non mette a disposizione molte funzionalità; essa è stata scritta solo per la conversione diretta da Word 6.0 / 97 / 2000 / XP / 2003 a testo piano. Per ottenere ciò basta istanziare un oggetto di tipo *WordExtractor* ed eseguire il suo metodo *extractText()* su un oggetto di tipo *InputStream* contenente il testo originale; in un'unica riga basta scrivere:

```
result = new WordExtractor().extractText(is);
```

dove *is* è il testo originale e *result* è la variabile di tipo *String* che conterrà il testo convertito.

Conversione da XML

Nel caso dei report è bastato invece l'ausilio delle classi proprie della JDK, in particolare di quelle dei package *javax.xml.parsers* e *org.w3c.dom*:

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
```

Nel file XML tutte le informazioni riguardanti il bug – compreso il testo del report – sono espresse in testo piatto ma racchiuse all'interno di tag identificati secondo lo standard adottato da Bugzilla, per cui ciò che bisogna fare è prelevare solo il testo compreso nei tag giusti. La procedura è la seguente:

1. si crea un'istanza della classe *DocumentBuilderFactory*, responsabile della definizione di parametri standard o propri del sistema necessari per la creazione di un successivo modello di rappresentazione XML:

```
DocumentBuilderFactory docBuildFac =
      DocumentBuilderFactory.newInstance();
```

2. da questa si trae un'istanza dell'oggetto di tipo *DocumentBuilder*, responsabile in maniera diretta della creazione dei modelli di documento XML:

```
DocumentBuilder docBuild = docBuildFac.newDocumentBuilder();
```

3. si crea finalmente l'oggetto documento XML utilizzando il metodo *parse()* per scandire il file di testo XML:

```
xmlDoc = docBuild.parse(xmlText);
```

   in cui *xmlText* è l'oggetto di tipo *InputStream* con il testo XML e *xmlDoc* è il relativo modello di tipo *org.w3c.dom.Document*;

4. si estraggono da questo gli elementi costituenti in un oggetto di tipo *Element*, per poi analizzarli:

```
Element xmlEl = xmlDoc.getDocumentElement();
```

5. ancora, si estraggono tra tutti gli elementi quelli inclusi nei tag interessati tramite il metodo *getElementsByTagName()* memorizzandoli in un oggetto lista di tipo *NodeList*; nel caso di studio ne sono stati estratti diversi, ma come esempio si riporta quello più importante, il testo del report memorizzato nei tag "*thetext*" (uno per ogni commento rilasciato da un utente diverso):

```
NodeList xmlTextNodes = xmlEl.getElementsByTagName("thetext");
```

6. si leggono infine i singoli elementi e si memorizzano in una stringa:

```
String txtContents[i] = xmlTextNodes.item(i).getTextContent();
```

4.2.2 Analisi del testo

A questo punto possono cominciare le prime elaborazioni sui testi e la loro indicizzazione. Di seguito viene spiegata ogni singola fase.

Eliminazione delle stopword

Le stopword sono state determinate in maniera manuale sia tramite ricerche nei dizionari in rete preparati da linguisti sia da considerazioni dettate dal particolare ambito specialistico in cui il tool opera. In pratica è stato predisposto un file di testo ASCII per ogni lingua supportata (per ora solo inglese ed italiano) in cui sono riportate tali parole comprese le relative variazioni sintattiche (genere e numero), una per riga in modo da facilitarne la lettura da parte dell'applicazione. Il tool poi in fase di esecuzione carica le parole sequenzialmente dal file della lingua selezionata e le pone in un array di stringhe: queste saranno utilizzate nella fase successiva.

Stemming delle parole

Uno dei più famosi algoritmi di stemming, sia per quanto riguarda le buone prestazioni in termini di tempo che l'efficacia delle operazioni, è lo *stemmer di Porter* [16], ideato nel 1980 dal Dr. Martin Porter. Si tratta di un algoritmo studiato appositamente per la lingua inglese ed in grado di analizzare e rimuovere circa sessanta suffissi. L'approccio usato si basa sull'applicazione di più passi consecutivi, ed ognuno di essi valuta la rimozione di suffissi.

Senza entrare nei particolari implementativi, il difetto principale riscontrato riguarda il fatto che non viene preso in considerazione il significato dei risultati restituiti. L'obiettivo di questo algoritmo non è quello di restituire una vera radice linguistica, piuttosto quello di

incrementare le prestazioni in velocità. Comunque, l'applicazione di regole molto generiche e l'assenza di un controllo sul significato, può portare ad errori sotto forma di normalizzazioni errate che non dovrebbero essere fatte o omissioni di normalizzazioni.

Nell'applicazione oggetto della tesi si è avuto bisogno di effettuare lo stemming di parole appartenenti a lingue diverse, per lo meno l'inglese e l'italiano, per cui vi è stata la necessità di cercare uno strumento più flessibile. La soluzione è stata trovata nel progetto *Snowball*, sviluppato dallo stesso Porter, che scelse il nome come tributo al linguaggio di programmazione *SNOBOL*, dai cui concetti di base ha preso largamente spunto. Esso è un vero e proprio piccolo linguaggio di programmazione per il trattamento delle stringhe, che permette di generare, fornendo le regole adatte, praticamente non solo lo stemmer necessario, ma addirittura un analizzatore completo.

In rete è disponibile la libreria omonima, la *Snowball*[3] appunto, sviluppata anche essa da Porter. Tale libreria è open-source, come la *Textmining.org*, ed è stata implementata come un modulo aggiuntivo della libreria di indicizzazione *Lucene*, che sarà illustrata più avanti: questa integra già al suo interno alcuni tipi di analizzatori di testo standard, ma permette di utilizzarne anche altri definiti esternamente. *Snowball* è giunta alla versione 2.2.0 ed è stata ottenuta come package java (*lucene-snowball-2.2.0.jar*).

Il suo utilizzo è molto semplice. Prima di tutto è stata importata nel progetto:

```
import org.apache.lucene.analysis.snowball.*;
```

e poi è stato creato un oggetto di tipo generico *Analyzer* istanziato come un *SnowballAnalyzer*:

```
Analyzer analyzer = new SnowballAnalyzer(lang, stopwords);
```

[3] http://snowball.tartarus.org

In pratica tale oggetto modella un analizzatore sintattico completo che verrà usato successivamente direttamente in fase di indicizzazione dei documenti. Esso accetta unicamente due parametri:

- *lang*, una stringa che identifica la lingua secondo le quali regole l'analizzatore deve effettuare lo stemming (ad esempio *"English"* per l'inglese, *"Italian"* per l'italiano, ecc.);
- *stopwords*, un semplice array di stringhe contenente tutte le stopword da eliminare nei testi analizzati (come descritto più sopra).

4.2.3 Creazione dell'indice

Infine si è pronti per la fase di indicizzazione. che nel caso in esame è "Lucene", una libreria open-source

Nel caso in esame il sistema utilizzato è una particolare libreria open-source scritta – tra gli altri linguaggi – in Java e chiamata *Lucene*, implementata originariamente da Doug Cutting nel 2000 ed entrata in seguito a far parte del progetto Jakarta della Apache Software Foundation [17]. Sebbene sia stata concepita per realizzare applicazioni che necessitano di funzionalità di indicizzazione e ricerca full text, *Lucene* è molto nota ed usata per la realizzazione di motori di ricerca sia sul World Wide Web che sulle intranet private. Questo ha portato all'affermazione di una percezione del prodotto come di un motore di ricerca, ma i suoi usi possono essere i più disparati, e quello che ne fa l'applicazione oggetto della presente tesi ne è un esempio.

È interessante sottolineare la struttura degli indici di *Lucene*. Questi sono semplicemente dei file aventi al loro interno delle sottostrutture logiche atte a contenere ognuna un

documento testuale; a loro volta ogni sottostruttura documento è suddivisa in un certo numero di campi composti dal termine o testo da indicizzare e dalla relativa qualificazione (titolo, corpo, codice identificativo, ecc.). Un esempio è riportato in Figura 4.1:

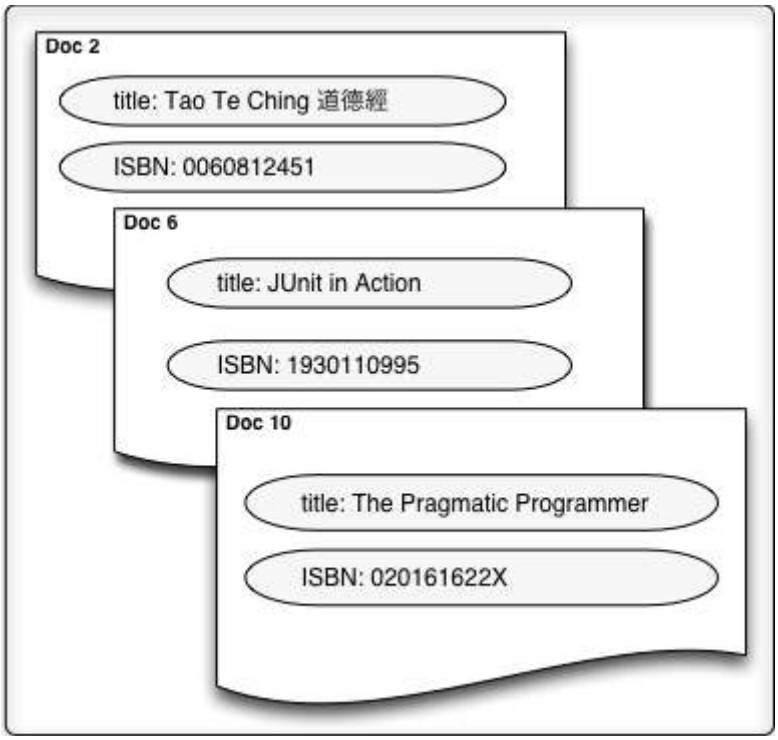


Figura 4.1 Vista logica di un indice di Lucene

Tale struttura logica è poi realizzata tramite una struttura fisica a file invertito (rif. Paragrafo 2.5.4), che permette una ricerca per termine più efficiente.

La particolare versione della libreria utilizzata è stata la 2.2.0[4] ed è stata ottenuta sotto forma di package java (*lucene-core-2.2.0.jar*).

Come al solito, si effettuano prima le importazioni necessarie:

```
import org.apache.lucene.analysis.*;
import org.apache.lucene.document.*;
import org.apache.lucene.index.*;
```

[4] http://lucene.apache.org

e poi, utilizzando l'analizzatore definito precedentemente, bastano pochi semplici passi:

1. si crea l'oggetto di tipo *IndexWriter* associato all'indice e responsabile delle sue modifiche:

   ```
   IndexWriter writer = new IndexWriter(indexDir, analyzer, true);
   ```

   in cui *indexDir* è un oggetto di tipo *File* che specifica la directory in cui sarà posizionato l'indice, *analyzer* è l'analizzatore definito, ed il terzo è un parametro booleano che fissato a *true* indica che l'indice sarà sovrascritto se già esistente;

2. si crea, per ogni documento da indicizzare, un oggetto di tipo *Document* che ne rappresenta il modello dal punto di vista di *Lucene*, personalizzabile tramite un numero arbitrario di campi di vario genere:

   ```
   Document doc = new Document();
   ```

3. si aggiungono all'oggetto precedente i campi necessari; in genere questi sono:

   - il titolo o l'ID del documento;
   - il contenuto testuale;
   - il percorso completo del file del documento.

   Come esempio, si riporta la creazione del campo più importante, cioè quello del contenuto testuale del documento:

   ```
   doc.add(new Field("contents", txtText, Field.Store.YES,
         Field.Index.TOKENIZED, Field.TermVector.YES));
   ```

   In un'unica riga sono state condensate due operazioni: la creazione di un oggetto di tipo *Field* che rappresenta uno dei campi del documento, e l'aggiunta di tale campo al modello di documento tramite il metodo *add()*. L'inizializzazione del *Field* ha bisogno di diversi parametri:

   - il nome del campo (nel caso in esame *"contents"*);
   - la stringa che rappresenta il contenuto del campo (*txtText*);

- un valore che esprime la volontà di memorizzare o meno il campo nell'indice, espresso come una costante della classe *Field.Store* (in questo caso *YES*);
- un valore che esprime la volontà di analizzare o meno il testo tramite l'analizzatore prima di memorizzarlo eventualmente nell'indice, espresso come una costante della classe *Field.Index* (in questo caso *TOKENIZED*, ad indicare che il testo deve essere analizzato);
- un valore che esprime la volontà di generare un vettore dei diversi termini del testo, ognuno con associato il relativo numero di occorrenze, espresso come una costante della classe *Field.TermVector* (in questo caso *YES*).

4. si aggiunge tale campo al modello di documento *Lucene* tramite il metodo *addDocument()*:

```
writer.addDocument(doc);
```

5. si esegue l'ottimizzazione dell'indice, effettuando tutte le operazioni precedenti fisicamente su disco:

```
writer.optimize();
```

6. al termine delle operazioni di aggiornamento, si rilascia la risorsa associata all'indice fisico:

```
writer.close();
```

4.3 Rappresentazione vettoriale dei documenti

In parallelo con il Paragrafo 3.4, si affronta ora la questione della realizzazione della rappresentazione vettoriale dei documenti. Come visto essa consiste nel creare uno spazio *n*-dimensionale in cui ogni dimensione è rappresentata da un particolare termine e il

modulo in quella direzione è rappresentata dal relativo numero di occorrenze nel documento.

Verso la fine del precedente paragrafo si è evidenziato che in fase di indicizzazione sono stati salvati anche i vettori dei termini di ogni documento, per cui si è potuta usare questa utilissima funzionalità per raggiungere lo scopo. La classe *RepresentationUtils* contiene il codice che realizza le rappresentazioni.

La struttura destinata a contenere tale rappresentazione non è stata creata ad hoc, ma è stata scelta tra quelle messe a disposizione dalla JDK: si tratta di un oggetto di tipo generico *Map*, facente parte del package *java.util*, costituito da un insieme di coppie chiave-valore i cui tipi sono definiti dall'utente. *Map* è un'interfaccia, e tra le varie sue implementazioni è stata scelta quella di tipo *TreeMap*, ordinata per chiave in modo da permettere una ricerca più veloce. La struttura logica ideata per la memorizzazione delle rappresentazioni è per l'appunto ad albero, in cui la radice è costituita da un oggetto *Map* di coppie con il campo chiave pari al titolo del documento considerato e con il campo valore costituito a sua volta da una *Map* di coppie con la chiave pari al termine contenuto considerato e con il valore pari al relativo numero di occorrenze. Sono state create due strutture di questo tipo, una per contenere i requisiti ed una per i report. In sostanza, dopo aver importato il package necessario:

```
import java.util.*;
```

si è definita la struttura in modo seguente (si riporta come esempio il caso dei requisiti):

```
Map<String, Map<String, Integer>> resultMap =
      new TreeMap<String, Map<String, Integer>>();
```

A questo punto non resta che riempirla, e ciò è stato fatto tramite i seguenti passi:

1. per ogni documento nell'indice si estrae il relativo modello:

   ```
   Document doc = reqsIndex.document(i);
   ```

2. dal documento si estrae il titolo per il campo chiave della mappa:

```
String reqName = doc.get("title");
```

3. si crea una sottostruttura *Map* destinata ad essere il campo valore della mappa principale:

```
Map<String, Integer> simpleMap = new TreeMap<String, Integer>();
```

4. dall'indice si estrae il vettore dei termini del documento considerato come oggetto di tipo *TermFreqVector*:

```
TermFreqVector reqReprVector =
      reqsIndex.getTermFreqVector(i, "contents");
```

   in cui il secondo parametro si riferisce al campo del documento di cui si vuole il vettore dei termini;

5. si estraggono dall'oggetto appena creato i termini e le relative occorrenze, memorizzandoli in array separati ma con lo stesso ordinamento degli elementi:

```
String[] terms = termFreqVector.getTerms();
int[] freqs = termFreqVector.getTermFrequencies();
```

6. si pone ognuna delle coppie nella sottomappa:

```
simpleMap.put(terms[i], (Integer) freqs[i]);
```

7. si memorizza ogni coppia titolo-mappa dei termini nella mappa principale:

```
resultMap.put(reqName, simpleMap);
```

4.4 Verifica delle associazioni tra requisiti e report

A questo punto si ha tutto il necessario per effettuare l'analisi comparata tra i requisiti ed i report. Questa è suddivisa operativamente in due fasi successive, illustrate di seguito.

4.4.1 Calcolo del livello di similitudine e scelta dei report significativi

La classe *AssociationUtils* mette a disposizione i metodi per verificare e quantificare le associazioni tra requisiti e report secondo la strategia top-*N*, memorizzando i risultati in due mappe. Tali mappe sono praticamente uguali, ma differiscono nell'organizzazione delle associazioni: la prima rappresenta un albero in cui per ogni requisito vi è la mappa dei report associati con i relativi livelli di similitudine, mentre la seconda dualmente è un albero in cui per ogni report vi è la mappa dei requisiti associati con i relativi livelli di similitudine. Ciò è stato fatto per velocizzare in seguito il recupero di queste informazioni in entrambi i punti di vista perché eventuali operazioni di riorganizzazione dei dati sono computazionalmente onerose. La struttura delle mappe è riportata di seguito:

```
Map<String, Map<String, Double>> firstAssoc =
      new TreeMap<String, Map<String, Double>>();
```

Senza entrare troppo nei dettagli del codice implementativo, simile per i metodi utilizzati a quanto già visto, si riporta di seguito l'algoritmo utilizzato:

1. tramite l'utilizzo di due strutture di tipo *Iterator* del package *java.util* si scandiscono le due mappe delle rappresentazioni vettoriali in modo sequenziale, cioè per ogni rappresentazione di requisito si attraversano tutte le rappresentazioni dei report. Di seguito sono riportati la creazione e l'uso di una tale struttura:

   ```
   Iterator<Entry<String, Map<String, Integer>>> firstIterator =
         firstMap.entrySet().iterator();
         while (firstIterator.hasNext()) {
               Entry<String, Map<String, Integer>> firstEntry =
         firstIterator.next();
   ```

   in cui l'oggetto di tipo *Entry* rappresenta una coppia della mappa.

2. per ogni entry si estraggono i due campi tramite i metodi appositi dell'oggetto *Entry*:

```
String firstKey = firstEntry.getKey();
Map<String, Integer> firstValue = firstEntry.getValue();
```

3. dalle rappresentazioni vettoriali si calcola il livello di similitudine tra il requisito ed il report considerati tramite il calcolo del *coefficiente di correlazione a coseno* (rif. Paragrafo 3.4.1).
4. Dato che per la strategia top-*N* si prendono in considerazione per ogni requisito solo gli *N* report con livello di similitudine maggiore, si controlla se l'associazione considerata può prendere posto all'interno delle relative mappe. Se già vi sono *N* documenti (nella mappa ordinata per requisiti) e il livello di similitudine è inferiore a tutti gli altri, semplicemente si scarta tale associazione, altrimenti se vi sono posti liberi essa viene inserita in maniera ordinata in entrambe le mappe oppure se ancora vi sono *N* documenti ma l'associazione merita di essere inserita si cerca e si cancella all'interno delle mappe l'associazione con livello di similitudine minore (sempre tra quelle relative alla mappa ordinata per requisiti) e poi si inserisce la nuova come visto prima.

4.4.2 Stima del livello di trattazione dei requisiti

La classe *ValutationUtils* contiene i metodi necessari al calcolo della stima finale sul grado di trattazione dei requisiti. Essa crea una mappa semplice, composta da un campo chiave di tipo *String* ed un campo valore di tipo *Integer*, in cui si riportano i nomi dei requisiti e le relative stime.

L’algoritmo procede alla creazione di un vettore risultante dalla somma vettoriale dei singoli report associati ad un requisito ed al calcolo del livello di similitudine tra quest’ultimo ed il vettore calcolato.

Si procede tramite i seguenti passi:

1. per ogni requisito si scandiscono i nomi dei report ad esso associati e si recuperano le relative rappresentazioni vettoriali;
2. ogni termine costituente la rappresentazione viene aggiunto ad una mappa semplice, avente termini come chiavi ed occorrenze come valori, in questo modo:
    - se il termine ancora non esiste nella mappa si aggiunge come nuova entry;
    - se già esiste si somma al numero di occorrenze presenti quello del report considerato.

    Alla fine del processo si ottiene la rappresentazione vettoriale del vettore risultante;
3. si calcola il livello di similitudine tra la rappresentazione vettoriale del requisito ed il vettore risultante ottenuto, esprimendola per comodità in percentuale.
4. si memorizzano il nome del requisito e la stima ottenuta nella mappa destinata a contenere i risultati finali.

A questo punto i risultati calcolati vengono presentati all’utente tramite l’interfaccia grafica, riportando a video due strutture ad albero:

- un albero con nodi di livello 1 pari ai nomi dei requisiti seguiti dalla stima sul grado di trattazione totale, e nodi di livello 2 pari ai nomi dei report associati con i relativi livelli di similitudine;
- un albero con nodi di livello 1 pari ai nomi dei report, e nodi di livello 2 pari ai nomi dei requisiti associati con i relativi livelli di similitudine.

I requisiti ed i report visualizzati al livello 2 sono stati ordinati in ordine di similitudine decrescenti tramite un algoritmo di ordinamento *quick-sort*, applicato da un apposito oggetto istanziato dalla classe *Quicksort*.

4.5 Supporto alla ricerca di requisiti emergenti

Per abilitare il supporto alla ricerca di eventuali requisiti emergenti, immediatamente dopo il calcolo delle stime sui requisiti il tool scandisce la mappa dei report e seleziona quelli che non sono stati associati ad alcun requisito; tali report vengono poi ulteriormente indicizzati in un nuovo indice per favorire le ricerche successive.

A questo punto il sistema è in grado di suggerire all'utente diverse parole che sembrano essere significative e, interagendo con esso, di fornire maggiori dettagli sulla distribuzione delle occorrenze di qualche parola in particolare. L'algoritmo è il seguente:

1. il sistema scandisce tutti i report dell'indice appena creato e colleziona tutti i termini in essi contenuti in una mappa semplice in maniera praticamente identica alla costruzione del vettore risultante del paragrafo precedente (cioè effettuando la somma delle occorrenze dei termini uguali);
2. da questo elenco di termini se ne selezionano i primi *n* ad occorrenza maggiore, dove *n* è un parametro stabilito dall'utente prima dell'elaborazione, in modo identico alla selezione dei report tramite strategia top-*N* del paragrafo precedente;
3. si presenta tramite l'ausilio dell'interfaccia grafica l'elenco delle parole con le relative occorrenze, ordinato in ordine di occorrenze decrescenti tramite l'algoritmo *quick-sort*;

Se l'utente specifica un termine da cercare, scegliendolo tra quelli suggeriti o fornendolo a proprio piacimento, il sistema avvia la fase di ricerca illustrata di seguito:

1. dato che tutti i termini memorizzati hanno subito uno stemming, prima di effettuare una ricerca è necessario effettuarlo anche sulla parola considerata. Ciò è ottenuto applicando l'analizzatore alla costruzione di una query costituita solo da un termine:

   ```
   QueryParser parser = new QueryParser(field, analyzer);
   Query query = parser.parse(word);
   String stemmedWord = query.toString();
   ```

   Prima si definisce un oggetto di tipo *QueryParser*, che costruisce query dopo aver analizzato i termini che le costituiscono, inizializzandolo con il nome del campo dell'indice su cui effettuare la successiva ricerca e con l'analizzatore da usare. Poi si esegue l'analisi ottenendo come risultato un oggetto di tipo *Query*, e da questo è possibile ricavare la parola analizzata tramite l'invocazione del metodo *toString()*.
2. si crea un oggetto di tipo *IndexSearcher*, responsabile delle ricerche all'interno dell'indice:

   ```
   IndexSearcher searcher = new IndexSearcher(index);
   ```

   dove *index* è la directory in cui è posizionato l'indice;
3. si effettua la ricerca tramite il metodo *search()*:

   ```
   Hits hits = searcher.search(query);
   ```

   che restituisce una struttura di tipo *Hits* contenente i risultati;
4. si scorrono i documenti contenuti in tale struttura prelevando il nome dei report e le relative occorrenze della parola cercata ed inserendoli in array:

   ```
   bugNames[i] = hits.doc(i).get("ID");
   wordFreqs[i] = bugs.get(bugNames[i]).get(interestedWord);
   ```

5. si ordinano gli array tramite l'algoritmo *quick-sort* in ordine di occorrenze decrescente ed infine si presentano i risultati ottenuti all'utente tramite l'interfaccia grafica.

4.6 Esempio di utilizzo

Come riepilogo delle varie possibilità che l'utente ha di utilizzare il sistema, si propone un esempio pratico di utilizzo del tool. Si supponga di voler effettuare un'analisi su di un progetto, e che si sia provveduto a salvare in file separati i requisiti e i report, archiviandoli in directory diverse. L'esempio che sarà utilizzato per illustrare l'uso del tool riguarderà un insieme di 14 requisiti e 268 report sui bug estratti dal progetto ArgoUML, di cui si parlerà approfonditamente nel Capitolo 5.
Per eseguire il tool è sufficiente posizionarsi nella directory in cui esso si trova e digitare dal prompt dei comandi "*java –jar DevelopAider.jar*". Nel seguito si vedranno in dettaglio i passi da compiere per sfruttare tutte le funzionalità che il tool mette a disposizione.

4.6.1 Configurazione dei parametri

All'apertura dell'applicazione si accede alla sezione dedicata alla configurazione, in cui è possibile specificare quasi tutti i parametri che il sistema userà a tempo di esecuzione (Figura 4.2):

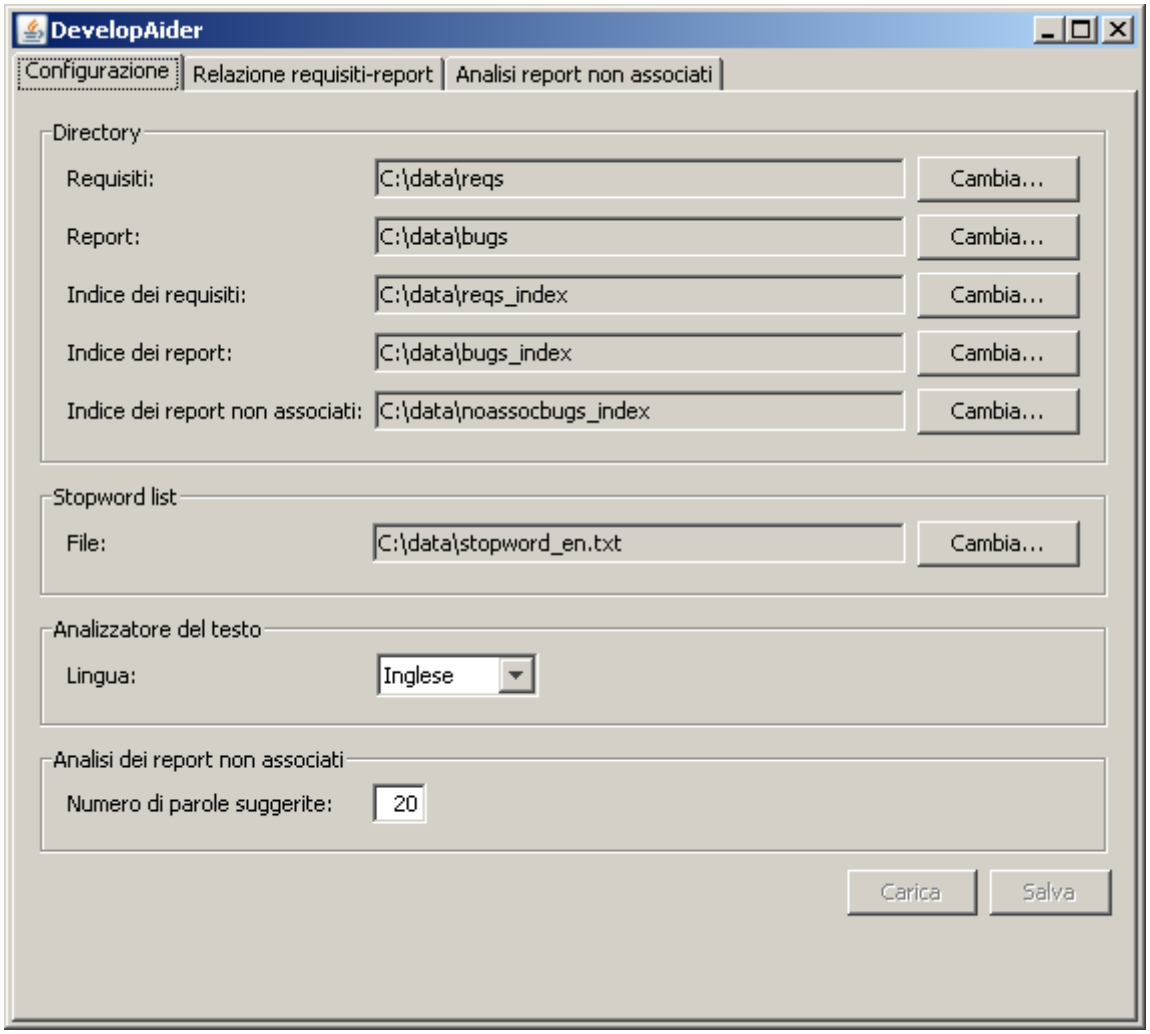


Figura 4.2 Finestra di configurazione

Come è possibile notare, sono già presenti delle informazioni di default nei campi; esse sono prese da un file di configurazione presente nella directory del programma, che può essere aggiornato con nuovi valori impostati dall'utente se desiderato.

Per cominciare, è possibile cambiare le opzioni riguardanti sia l'indicazione di directory che di file. Per quanto riguarda le directory, il tool ha bisogno di conoscerne cinque:

1. directory contenente i file dei requisiti da analizzare;
2. directory contenente i file dei report sui bug con cui condurre l'analisi;

3. directory destinata a contenere i file che costituiranno l'indice della base di dati relativa ai requisiti;
4. directory destinata a contenere i file che costituiranno l'indice della base di dati relativa a tutti i report sui bug;
5. directory destinata a contenere i file che costituiranno l'indice della base di dati relativa ai report sui bug che dall'analisi risulteranno non associati ad alcun requisito.

Supponendo che i file necessari si trovino nelle directory "*requirements*" (per i requisiti) e "*reports*" (per i report), contenute all'interno di "*C:\example*", e che si siano destinate per gli indici le directory seguenti (create in precedenza):

- "*C:\example\requirements_index*" per l'indice dei requisiti;
- "*C:\example\reports_index*" per l'indice dei report;
- "*C:\example\new_reports_index*" per l'indice dei report non associati;

è possibile specificare ognuna tramite apposite finestre di sistema attivate con il relativo pulsante **Cambia** (Figura 4.3):

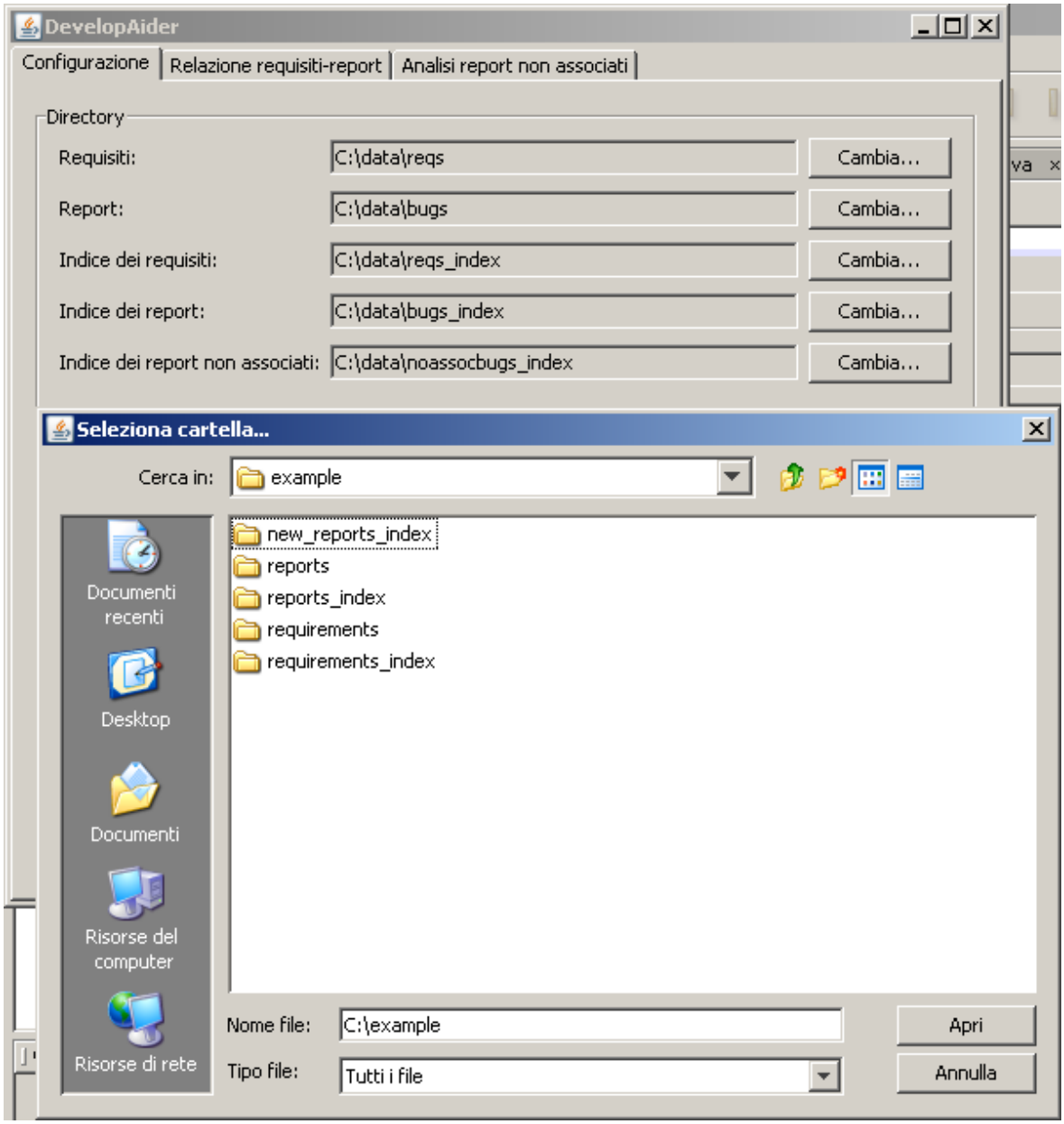


Figura 4.3 Finestra di selezione directory

Da notare che la finestra di sistema consente di visualizzare (e quindi selezionare) unicamente directory e non file.

La stessa operazione dovrà essere fatta per le directory dei vari indici da creare, ottenendo così infine il risultato in Figura 4.4:

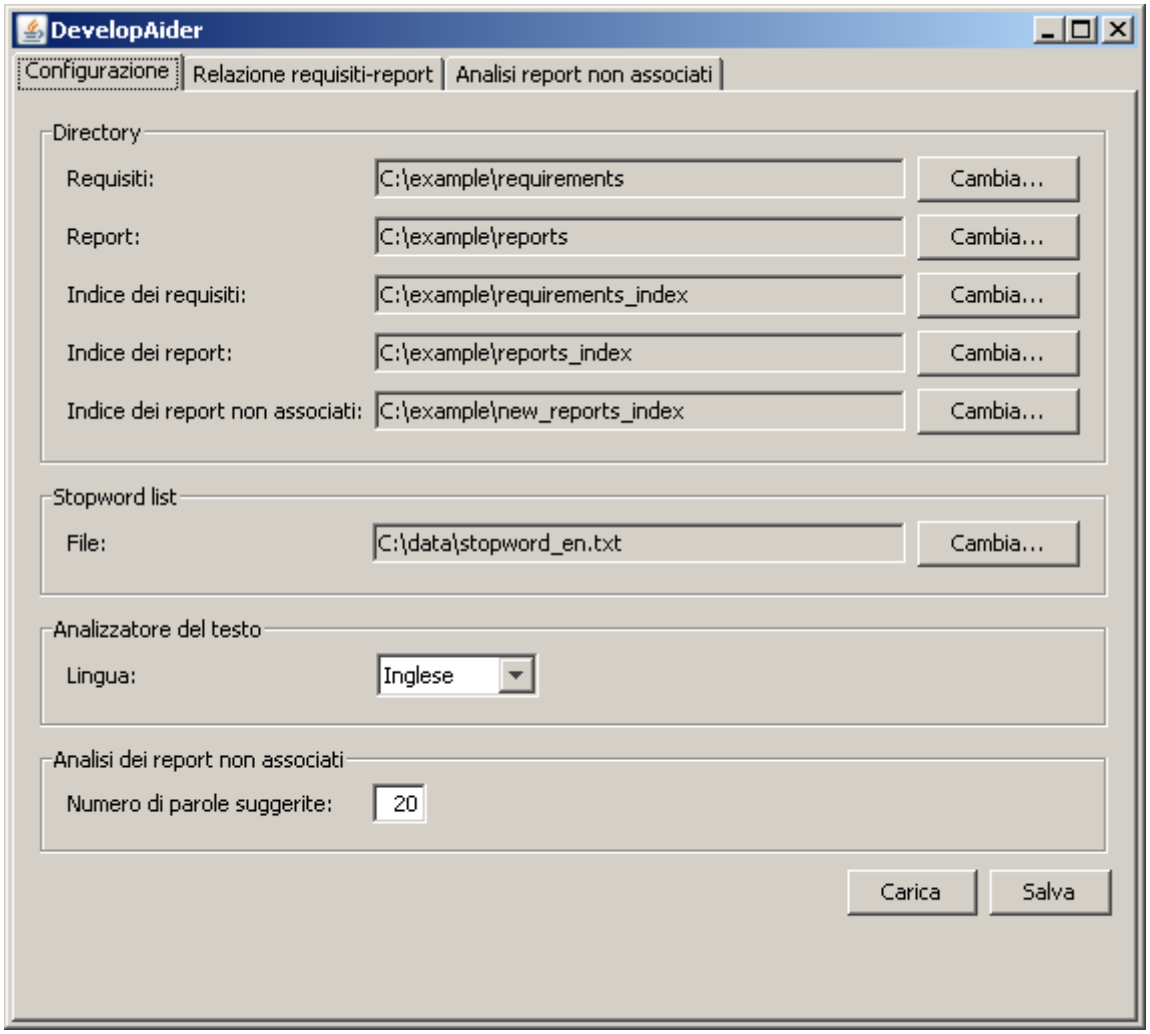


Figura 4.4 Finestra di configurazione con directory definite

Per quanto riguarda l'indicazione di file, bisogna specificare il percorso ed il nome di quello contenente l'elenco delle stopword da utilizzare nella fase di analisi. Generalmente ogni lingua ha il proprio elenco di stopword, e nella versione attuale ne esistono due, per l'italiano e l'inglese. L'esempio considerato utilizza la lingua inglese. L'interfaccia è praticamente uguale a quella vista per le directory, tranne per il fatto di avere in più la possibilità di poter selezionare file (Figura 4.5):

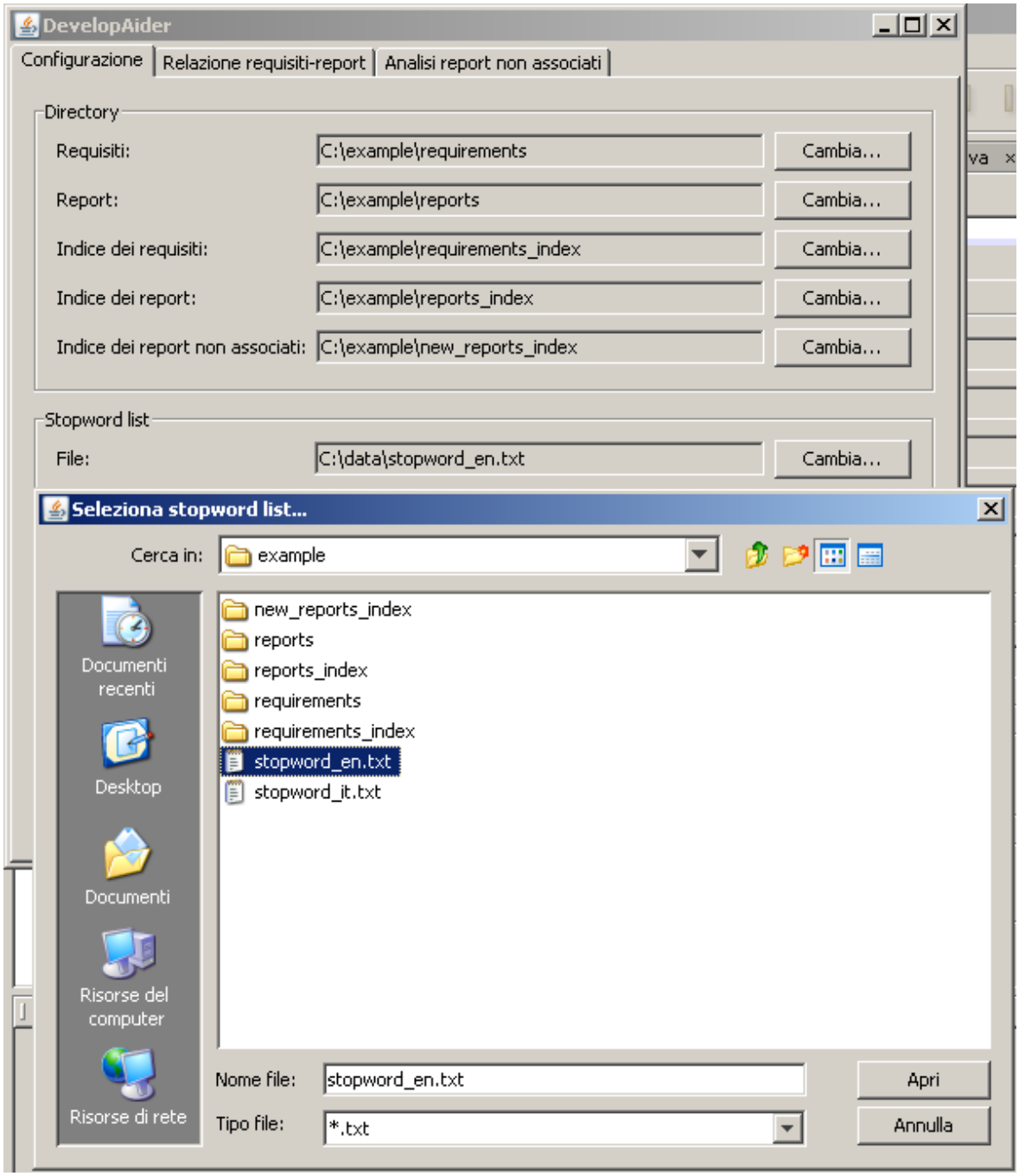


Figura 4.5 Finestra di selezione file

Altri parametri importanti da definire sono poi:

- la lingua utilizzata dall'analizzatore del testo in fase di indicizzazione dei documenti (per ora italiano o inglese). Da notare che per un'impostazione corretta

dell'analizzatore è necessario che il file delle stopword sia quello relativo alla lingua selezionata;

- il numero massimo di parole a frequenza di occorrenza maggiore che il sistema suggerirà in fase di analisi dei report non associati ad alcun requisito, estraendole da questi. In questo caso si lascerà l'impostazione di default pari a venti parole.

Infine, la configurazione dovrebbe presentarsi come in Figura 4.6:

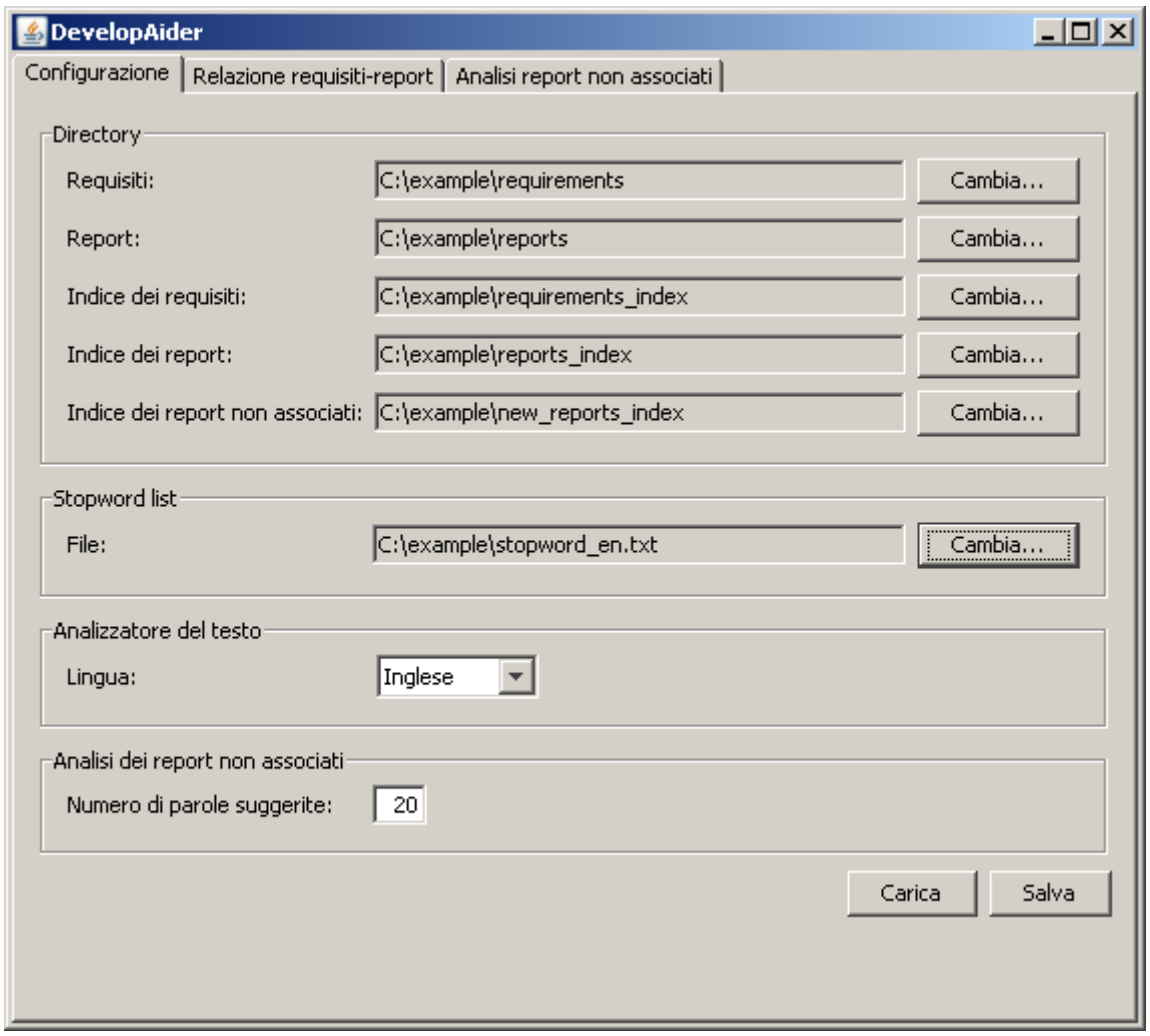


Figura 4.6 Configurazione per l'esempio

Se lo si desidera, è possibile salvare permanentemente i parametri definiti nel file di configurazione (sostituendo quella già presente) tramite il tasto **Salva**, oppure è possibile annullare le modifiche apportate ricaricando la configurazione salvata mediante l'ausilio del tasto **Carica**.

4.6.2 Analisi dei documenti

Definiti i parametri desiderati, è possibile passare alla finestra di analisi (Figura 4.7):

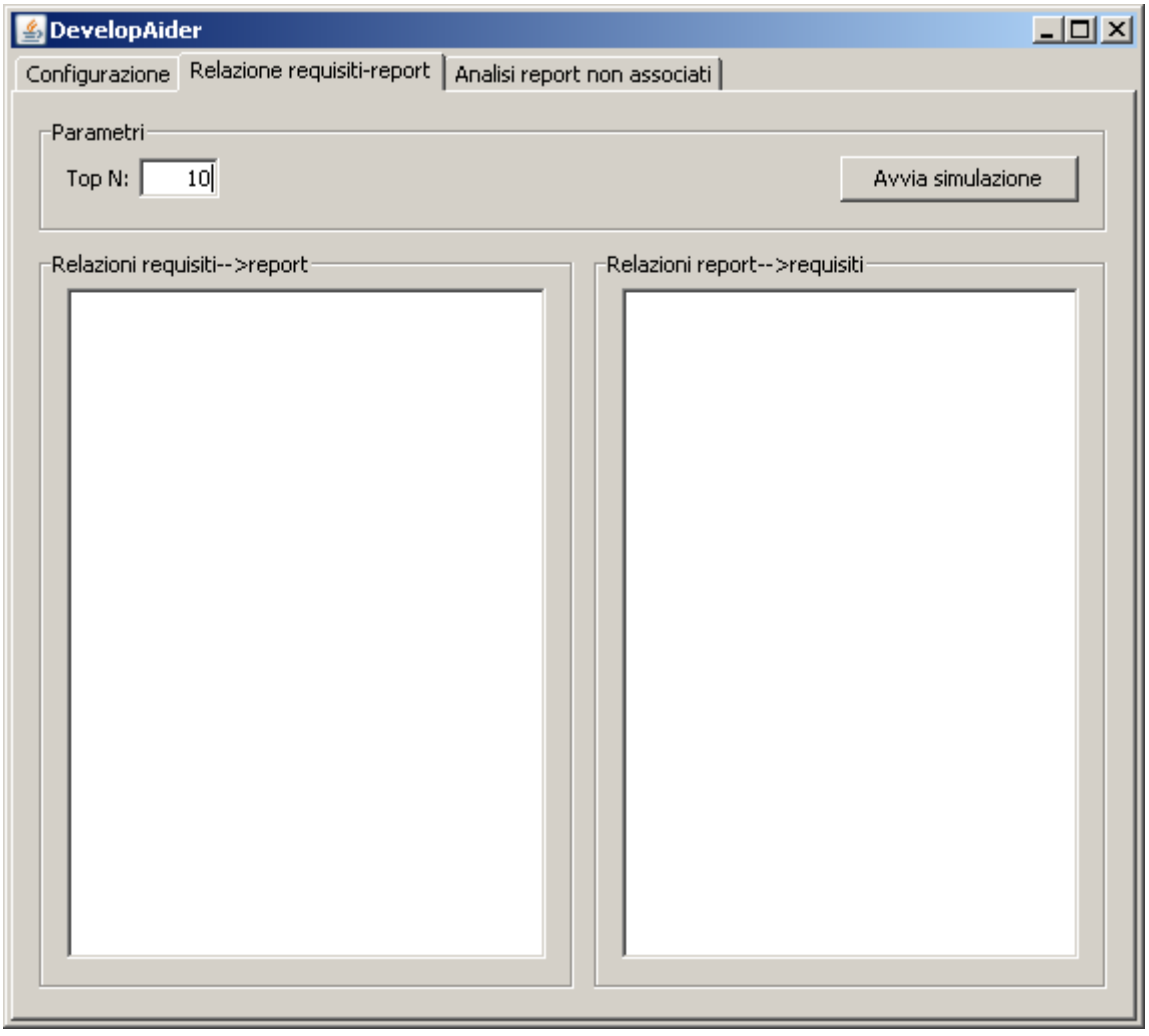


Figura 4.7 Finestra di analisi prima dell'elaborazione

Prima di avviare l'analisi dei requisiti, bisogna fissare il parametro *top N* (che come si è visto nei capitoli precedenti indica il numero massimo di report a maggiore similitudine da prendere in considerazione per ogni requisito); dopodiché si può avviare l'analisi con il pulsante **Avvia simulazione**. Nell'esempio considerato si è fissato un *top N* pari a 10, ed i risultati sono illustrati di seguito (Figura 4.8):

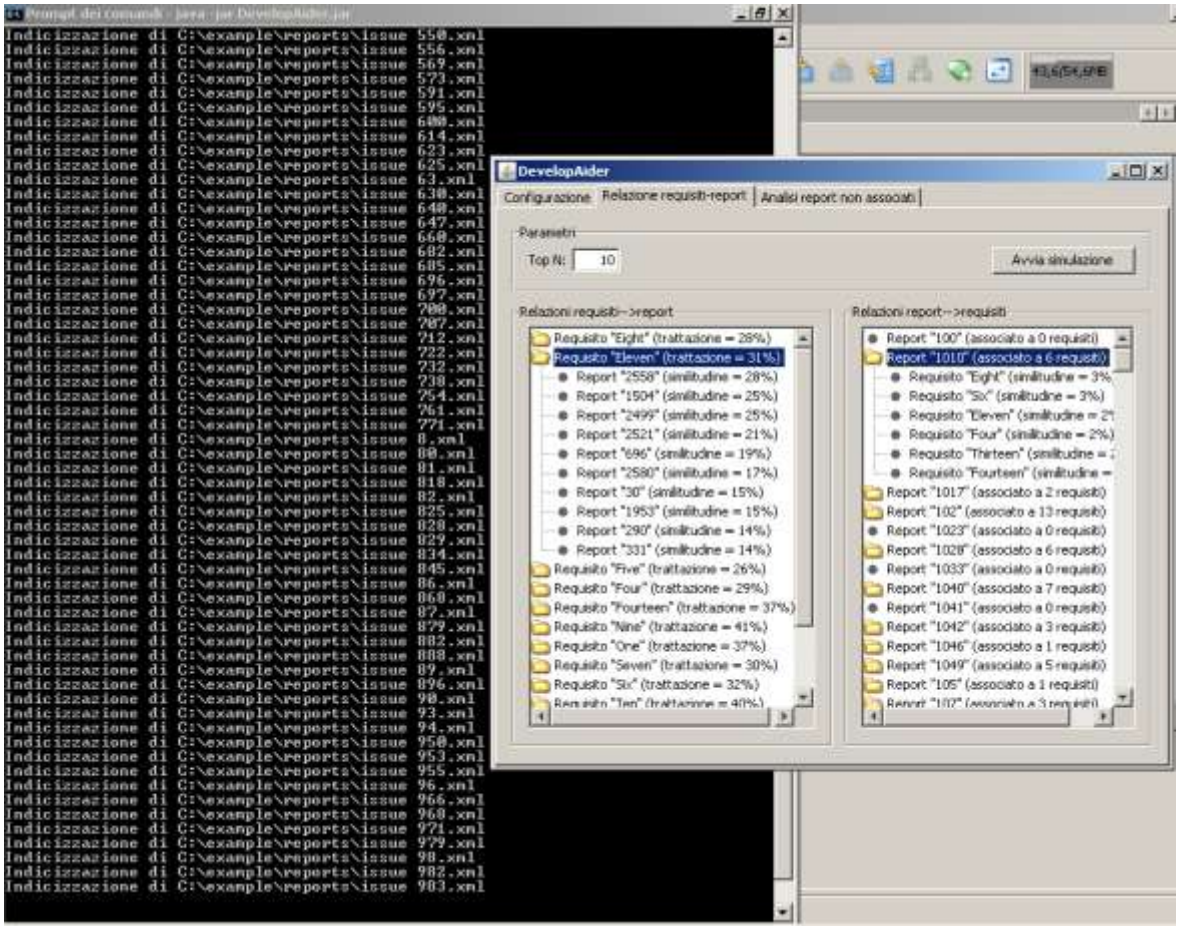


Figura 4.8 Finestra di analisi dopo l'elaborazione

La visualizzazione dei risultati è composta di due strutture gerarchiche ad albero:

- la prima, a sinistra, è relativa alle informazioni dal punto di vista dei requisiti. Il primo livello dell'albero è composto da nodi che rappresentano i requisiti ed il loro livello di trattamento complessivo stimato; facendo doppio clic su uno di essi si

aprono i nodi di secondo livello (terminali) che mostrano i report associati a tale requisito ed il relativo livello di similitudine calcolato. I report sono visualizzati in ordine di similitudine decrescente;

- la seconda, a destra, è relativa alle informazioni dal punto di vista dei report. Sostanzialmente la visualizzazione è identica al caso precedente, ma qui si elenca per ogni report quanti e quali requisiti sono ad esso associati per evidenziare la presenza di eventuali requisiti ridondanti o di report non associati.

4.6.3 Estrazione di informazioni dai report non associati

La fase di analisi al punto precedente si occupa anche di organizzare in un apposito indice i report che non hanno trovato associazione con alcun requisito, ed estrarre da questi i termini a più alto numero di occorrenze, come supporto all'ingegnere dei requisiti per la scoperta di eventuali requisiti emergenti. Basandosi sull'impostazione del numero massimo di termini da suggerire, definita nella schermata di configurazione o prelevata dal file di configurazione, è possibile trovare un elenco di tali termini nella finestra "Analisi dei report non associati", che nell'esempio considerato con venti parole suggerite si presenta come in Figura 4.9:

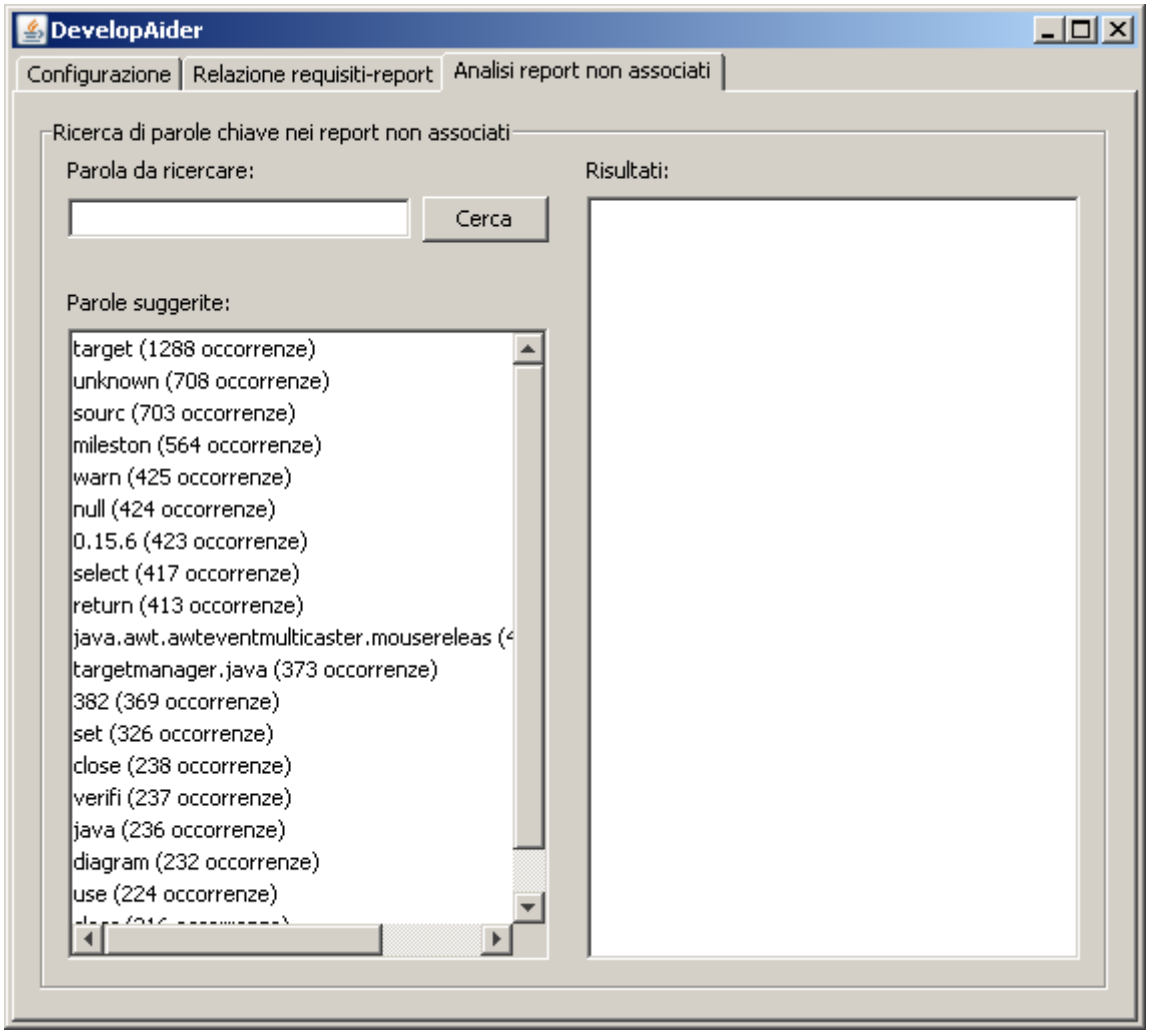


Figura 4.9 Finestra di analisi dei termini nei report non associati

I termini sono presentati in ordine di numero di occorrenze decrescente.

Da qui si possono richiedere maggiori informazioni su un particolare termine, utilizzando due strategie:

1. selezionando uno dei termini suggeriti dal sistema. Il termine viene riportato automaticamente nella casella di testo soprastante, e basta poi premere il tasto **Cerca** (Figura 4.10);

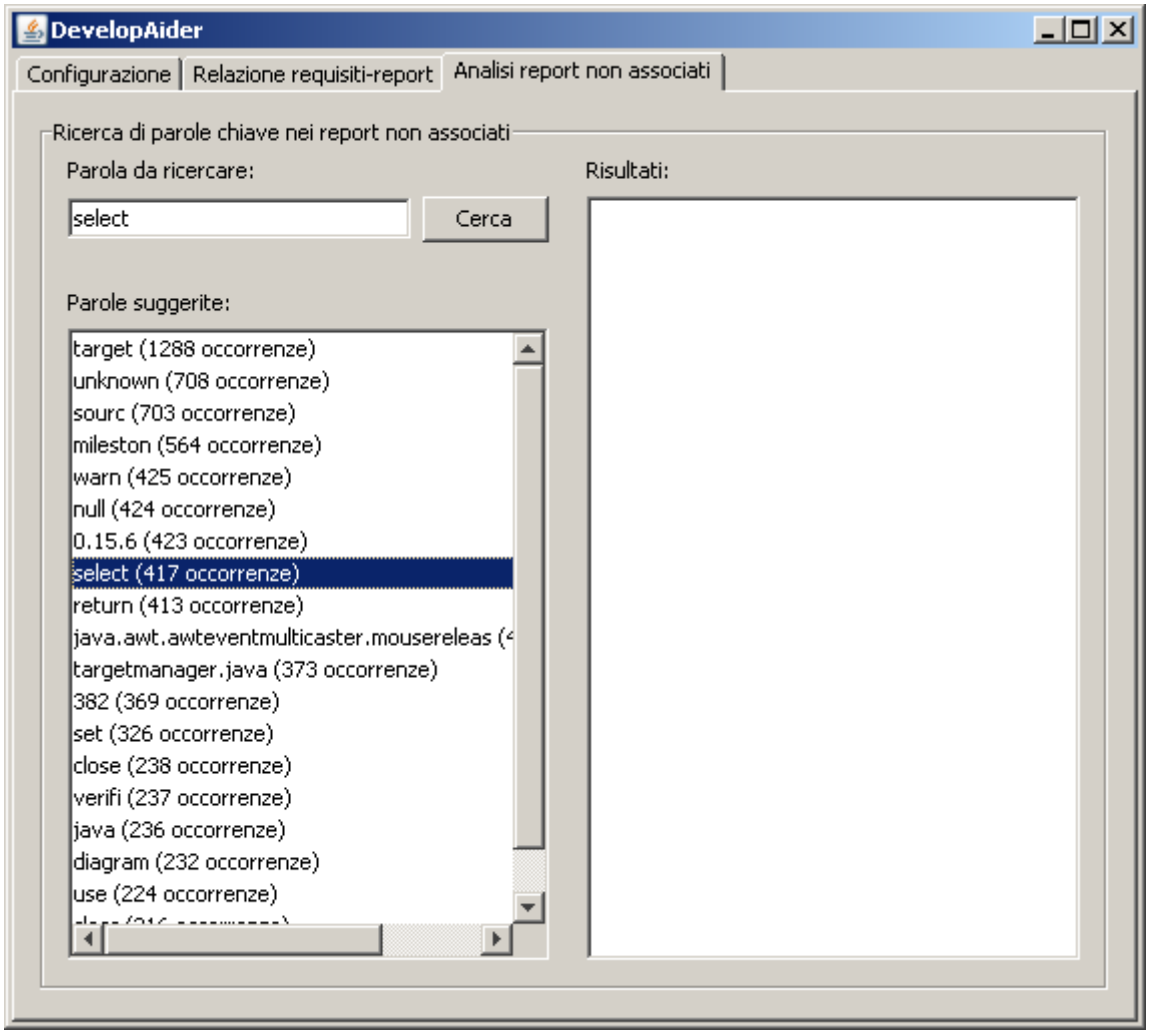


Figura 4.10 Selezione di un termine suggerito dal sistema

2. specificando direttamente un termine a cui si è interessati nella casella di testo e premendo il tasto **Cerca** (Figura 4.11).

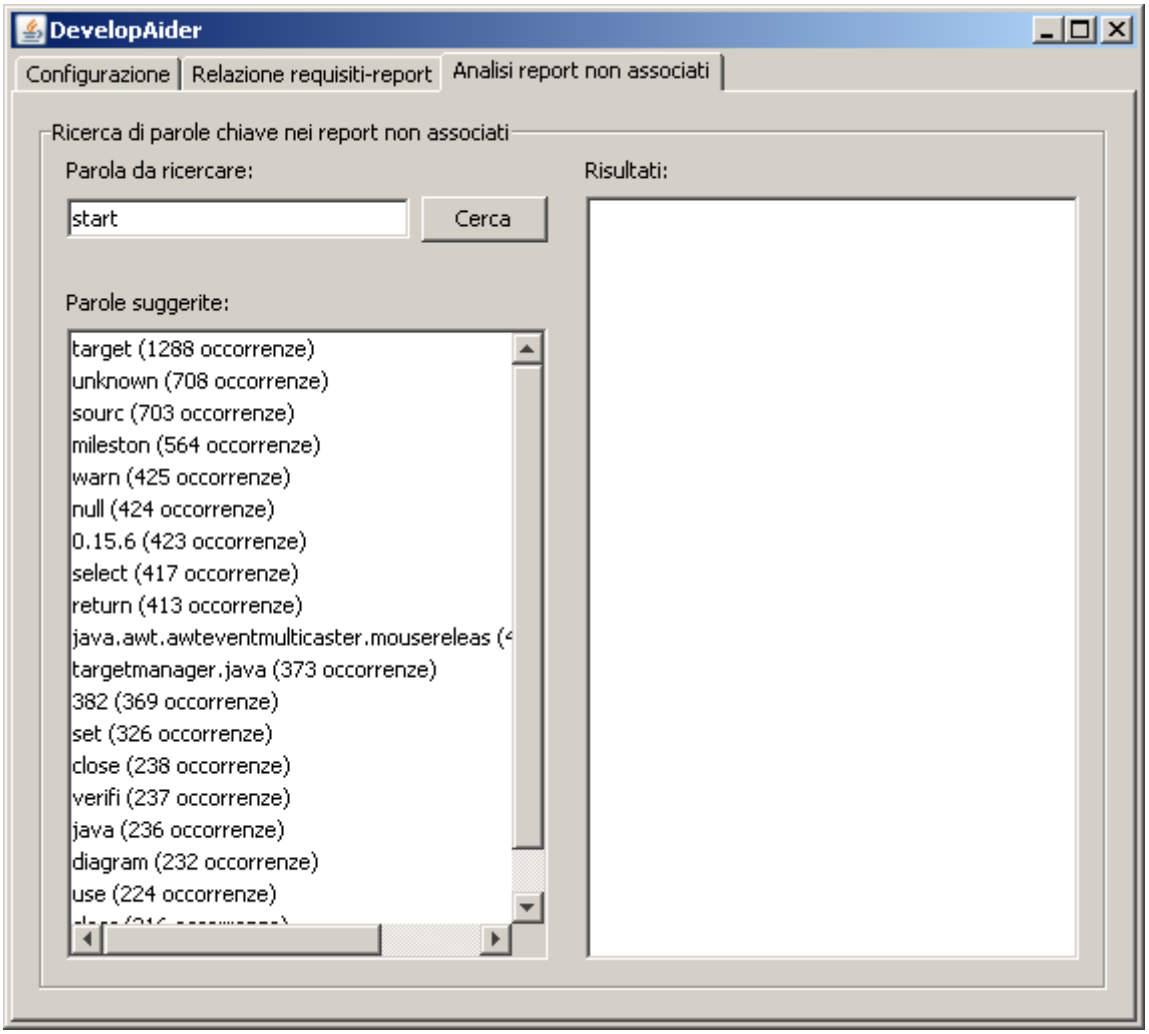


Figura 4.11 Definizione di un termine particolare

In entrambi i casi il sistema fornirà nella parte destra della finestra un riepilogo delle informazioni sulla parola chiave richiesta, riportando nuovamente il numero di occorrenze totali e l'elenco dei report (tra quelli non associati) contenenti tale parola corredati dal relativo numero di occorrenze del termine (Figura 4.12):

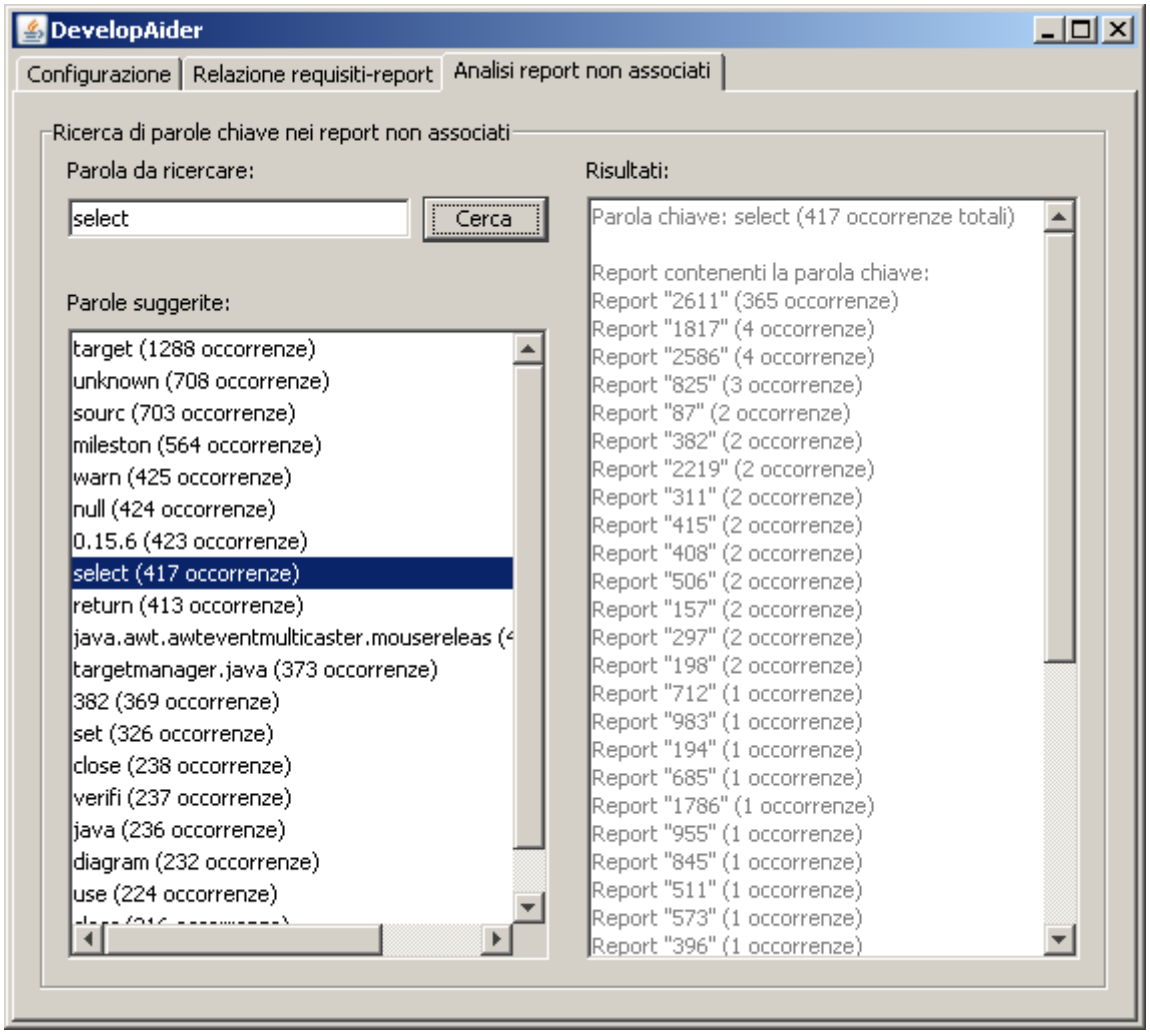


Figura 4.12 Finestra di ricerca dopo l'analisi di un termine

Da notare che anche qui, per facilità di lettura, l'elenco è ordinato per numero di occorrenze decrescente.

Capitolo 5 – Sperimentazione

5.1 Introduzione

Finora si è discusso ampiamente degli aspetti teorici, metodologici ed implementativi del tool realizzato nel presente lavoro di tesi. A questo punto è d'uopo condurne una sperimentazione analizzando più in dettaglio i risultati di una esecuzione del programma su casi reali, in modo da valutare la qualità delle scelte metodologiche fatte in fase di progetto.

Come visto nei capitoli precedenti, lo strumento software *DevelopAider* analizza i requisiti di un progetto ed i relativi report sui bug fornendo diverse informazioni; tra queste, il livello di trattamento globale stimato per ogni requisito e la relativa misura di similitudine con ogni report ad esso associato sono quelle sicuramente principali e più interessanti, in quanto le altre sono derivate proprio da queste e comunque forniscono supporto per scopi secondari.

Il punto di partenza è la considerazione che un'analisi quantitativa (ma anche solo qualitativa) sulla relazione esistente tra un requisito ed un report è una questione non banale perché prevede un confronto a livello semantico, e come visto ampiamente nel corso del Capitolo 2 l'information retrieval basato su metodi semantici non riesce a dare risultati soddisfacenti commisurati allo sforzo progettuale necessario. Il tool basa le sue misure semplicemente sul calcolo della similitudine delle rappresentazioni in uno spazio vettoriale dei documenti considerati, calcolandone il prodotto vettoriale. Questo è in realtà un confronto sintattico: non solo il livello di similitudine calcolato potrebbe non

rispecchiare quanto il report tratta del requisito considerato, ma addirittura potrebbe essere stabilita un'associazione tra un requisito ed un report che in realtà affrontano argomenti diversi.

Da ciò si comprende che un'analisi realmente efficace può essere condotta solo da una persona fisica, e che il presente tool può essere utilizzato solo per avere suggerimenti e linee guida.

5.2 Criterio di valutazione

Dalle considerazioni precedenti scaturisce quindi il bisogno di effettuare una verifica dell'effettiva associazione semantica tra un requisito ed un report individuata dal tool.

Come visto nel Paragrafo 2.3, nell'information retrieval in generale (ma anche nel caso particolare del *DevelopAider*) la valutazione è condotta di solito in termini del livello di *precisione* e di *richiamo* ottenuti. È stato detto che queste metriche sembrano opposte perché inversamente proporzionali, ma in realtà sono entrambe necessarie per comprendere fino in fondo quanto un metodo di recupero sia efficace. Infatti da un lato la *precisione* misura quanto di ciò che è stato recuperato e quindi considerato significativo sia in effetti attinente all'argomento in esame. Nell'applicazione svolta, ciò significa stabilire quanti dei report considerati attinenti ad un dato requisito lo siano effettivamente. Se si suppone che ad un certo requisito siano stati associati dall'applicazione N_P report, ma di questi solo $n \leq N_P$ hanno realmente attinenza con il requisito, allora la *precisione P* ottenuta per quest'ultimo è

$$P = \frac{n}{N_P}$$

in cui ovviamente *P* sarà un valore reale compreso tra 0 e 1. Quanto più la precisione si avvicina all'unità tanto più efficace è il metodo con cui le associazioni vengono stabilite.

Dall'altro lato il *richiamo* misura quanto di tutto ciò che è effettivamente attinente ad un dato argomento sia stato recuperato considerandolo significativo. In pratica, se su N_R report totali relativi ad un requisito l'applicazione ne recupera solo $n \leq N_R$, sottoinsieme dei report N_P totali considerati significativi da questa come visto prima, allora il *richiamo R* per il particolare requisito è

$$R = \frac{n}{N_R}$$

Anche in questo caso *R* sarà un valore reale compreso tra 0 e 1, e valori prossimi all'unità indicano che il metodo tende a recuperare quasi tutti i documenti significativi.

Le due misure di qualità del metodo sono inversamente proporzionali in quanto un aumento della *precisione* implica quasi sempre una maggiore selettività dell'algoritmo di scelta dei documenti, che inevitabilmente scarta una quantità maggiore di report effettivamente significativi, mentre un aumento del *richiamo* dovuto ad un rilassamento dei vincoli di scelta e/o ad un incremento imposto del numero di documenti da recuperare aumenta la probabilità di promuovere report non attinenti all'argomento considerato.

L'andamento relativo tipico dei due parametri è riportato in Figura 5.1.

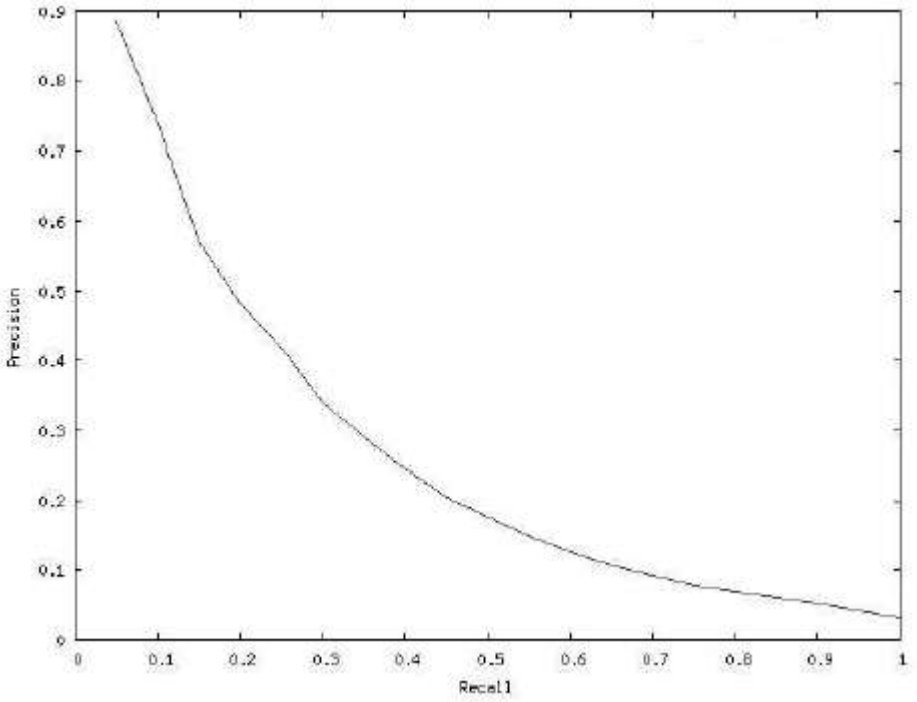


Figura 5.1 Confronto degli andamenti di precisione e richiamo

Un'equa valutazione dovrebbe basarsi su un bilanciamento di questi due fattori in modo da massimizzare le informazioni utili recuperate minimizzando al contempo gli errori di valutazione nella scelta. Se è possibile correlare gli andamenti della precisione e del richiamo ad un parametro del sistema, si ottiene un grafico simile a quello riportato in Figura 5.2:

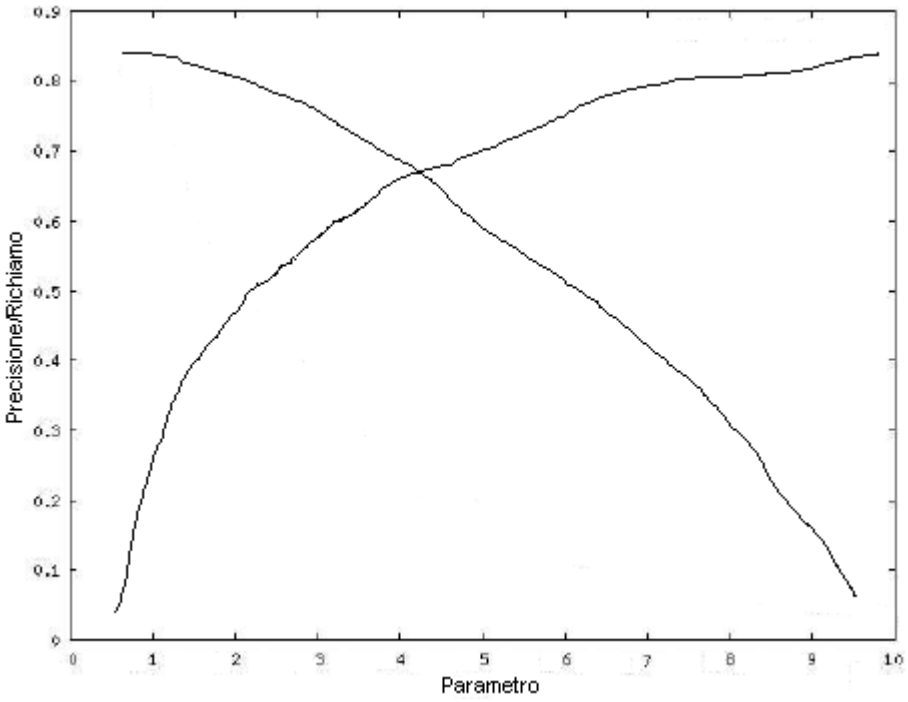


Figura 5.2 Dipendenza parametrica di precisione e richiamo

e la scelta più idonea del parametro è tipicamente suggerita dal punto di intersezione delle due curve.

Purtroppo però la misura del *richiamo* non sempre è fattibile – come in questo caso – perché si avrebbe bisogno di conoscere nel dettaglio tutti i documenti nella base di dati, e normalmente il numero di report sui bug per un'applicazione di taglia media supera lungamente le migliaia. È però possibile valutare il tool in termini della sola *precisione*, mandandolo in esecuzione diverse volte e raffrontando le *precisioni* globali ottenute ad ogni passo.

Più in dettaglio, si esegue il tool con diversi valori del parametro top-*N* (rif. Paragrafo 4.4.1), e ad ogni passo si effettuano le seguenti operazioni:

1. per ogni requisito si prendono in considerazione tutti gli *N* report ad esso associati automaticamente dal sistema, e si assegna ad ognuno di loro un valore di *precisione*:

- il valore 1, se il report effettivamente affronta l'argomento del requisito associato;
- il valore 0, se essi non sono legati dal punto di vista semantico.

2. per ogni requisito si calcola la *precisione* media al livello top-*n*, effettuando una media dei valori di precisione appena calcolati dei report ad esso associati;
3. si effettua la media delle *precisioni* calcolate per ogni requisito, ottenendo la *precisione* globale al livello top-*n*.

Alla fine del processo si ottiene la *precisione* che il tool è in grado di fornire a diversi livelli di top-*n* per il particolare progetto da testare, ed è quindi possibile rendersi conto delle prestazioni massime raggiungibili e del livello di top-*n* tipico a cui queste sono raggiunte. Data l'impossibilità di realizzare una sperimentazione completa a causa di vincoli contingenti, è stato analizzato un caso di studio a cui è stata applicata tale procedura.

5.3 Caso di studio - ArgoUML

DevelopAider è uno strumento ad ampio raggio, estensivamente applicabile, purché ovviamente sia accessibile il materiale riguardante l'SRS (o perlomeno i requisiti funzionali) del software oggetto dell'analisi e la base di dati contenente tutti i report sui bug riscontrati. Purtroppo non sempre le buone pratiche e le linee guida dell'ingegneria del software sono rispettate (come in alcuni progetti open-source) o comunque spesso particolari informazioni progettuali e manutentive sono considerate riservate, con il risultato che cercare in rete un progetto completo adatto come caso di test diventa

un'impresa abbastanza ardua: è piuttosto comune trovare server Bugzilla o Issuezilla[5] nei progetti open-source, ma è praticamente impossibile trovare documentazione progettuale in merito.

Il problema è stato parzialmente risolto dal progetto *ArgoUML*[6]. Si tratta di un tool open-source che permette di costruire modelli UML di progetti creando i nove tipi di diagrammi dello standard UML 1.4 (Figura 5.3):

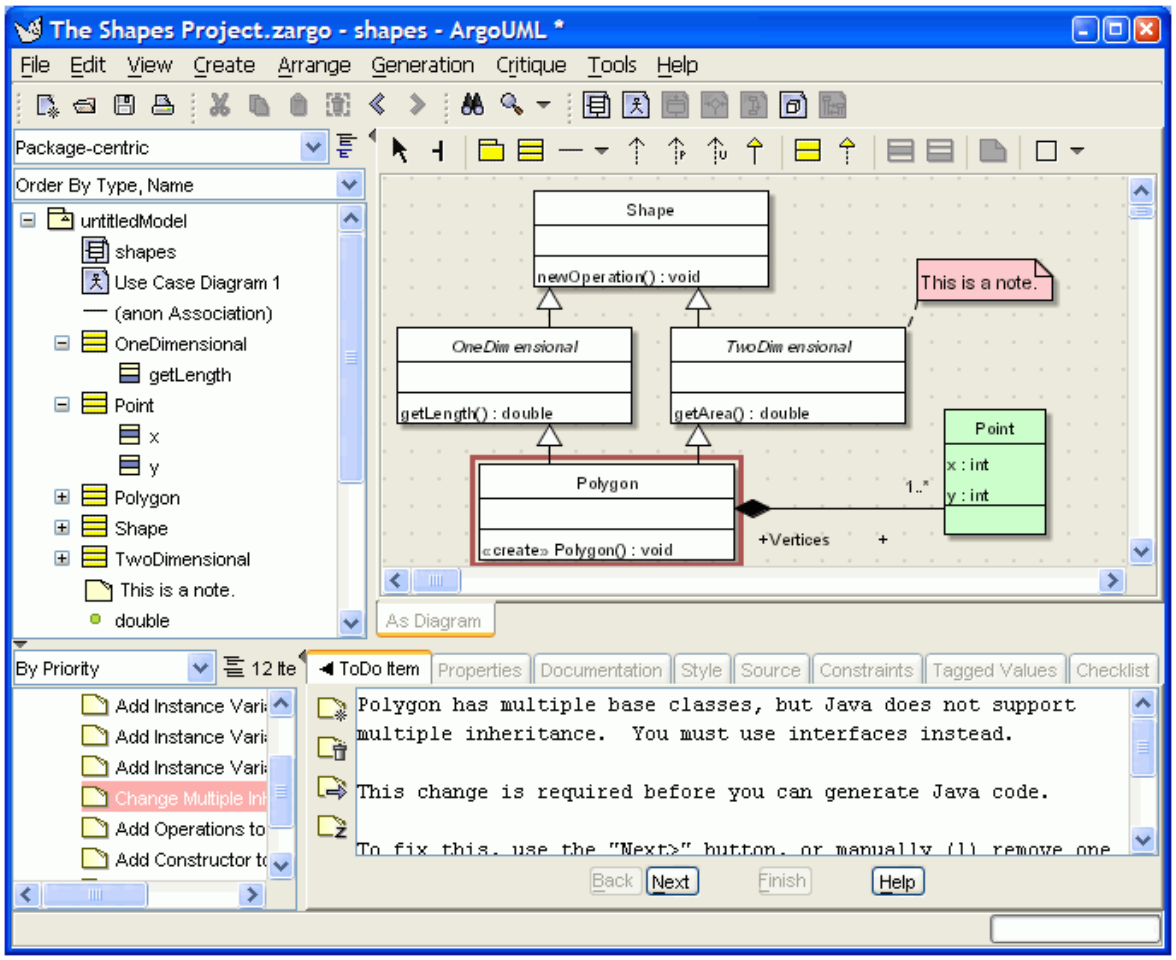


Figura 5.3 Finestra principale di ArgoUML

[5] Variante del server Bugzilla utilizzato in diversi progetti open-source
[6] http://argouml.tigris.org

Il sito mette a disposizione una base di dati contenente diversi report sui bug e mantenuta da un server Issuezilla (che permette di visualizzarli sia in formato HTML che XML), e pubblica anche i requisiti. Purtroppo però questi ultimi non sono quelli classici di progetto dell'ingegneria del software elencanti le funzionalità dell'applicazione, bensì dei requisiti legati all'implementazione (aspetto, compatibilità, ecc.), quindi la valutazione effettuata su questo materiale potrebbe non rispecchiare le effettive prestazioni del tool su progetti aventi documentazione completa.

Nel dettaglio sono stati presi in considerazione un totale di quattordici requisiti, di cui:

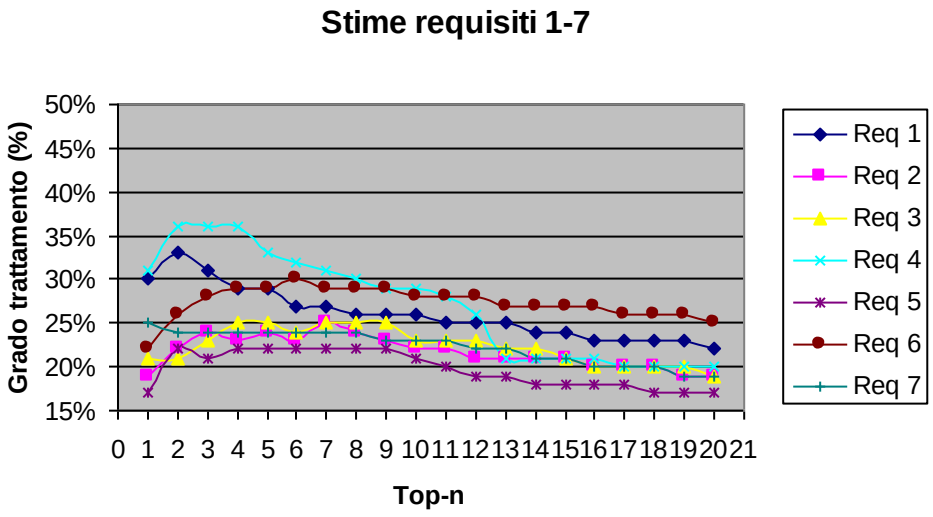
- otto riguardanti l'interfaccia utente grafica (GUI), relativa alle funzioni generali del ciclo di vita dei diagrammi, quali aggiornamenti dei diagrammi, protezione dei campi di testo, ecc.;
- due riguardanti l'aderenza allo standard UML 1.4;
- tre riguardanti la compatibilità con le varie versioni di Java e delle JVM;
- uno riguardante il supporto allo sviluppo dell'applicazione.

Per quanto riguarda i report sui bug, il server Issuezilla ne mette a disposizione circa cinquemila, legati a diverse versioni dell'applicazione ed aventi cinque differenti priorità. Dato l'elevato numero di documenti che si sarebbero dovuti collezionare ed analizzare manualmente, si è deciso di restringere lo studio ad un campione scelto casualmente consistente in cento report di diverse priorità ma facenti capo ad una versione particolare dell'applicazione, la *milestone* 0.15.6: ciò al fine di garantire una certa omogeneità degli argomenti trattati dai report scelti. È stato calcolato che i risultati derivanti dalla scelta di tale campione si discostano da quelli effettivi basati su tutti i report relativi alla milestone considerata, per un intervallo di confidenza pari al 99%, di un valore pari all'8%: ciò vuol dire che se ad esempio si rileva una *precisione* dell'80%, questa può in realtà essere compresa tra il 72% e l'88%, variazione che non influenza molto la stima ricavata.

Riguardo il numero di iterazioni da eseguire con *DevelopAider* si è ritenuto sufficiente partire da un top-*n* pari ad uno ed avanzare per unità fino ad un top-*n* pari a venti.

Ad ogni livello si considerano significativi, per ogni requisito, solo i report con i livelli di associazione maggiori, per cui le iterazioni al livello top-*n* concettualmente si rifanno (relativamente alle computazioni sulle singole associazioni) a quelle al livello top-*n-1* con l'aggiunta di un report per ogni requisito. Risulta quindi inutile e dispendioso in termini di spazio e praticità di lettura riportare i risultati ottenuti per tutti i livelli top-*n* considerandoli uno alla volta: si riporteranno semplicemente quelli ottenuti nel caso top-*20*, aggiungendo tutti i valori delle *precisioni* globali ad ogni livello.

Riguardo l'output prodotto da *DevelopAider*, per comodità di lettura in Appendice B è riportata la tabella con le associazioni riscontrate, in cui in ogni cella si specifica l'ID del report selezionato dal tool ed il relativo livello di similitudine con il requisito associato, espresso in percentuale (Tabella B.1). Inoltre, sempre in Appendice B sono riportate le stime sul grado di trattamento totale dei requisiti secondo il tool (Tabella B.2). Di seguito si forniscono solo due diagrammi che illustrano l'andamento delle stime calcolate al variare del top-*n* (Figura 5.4).



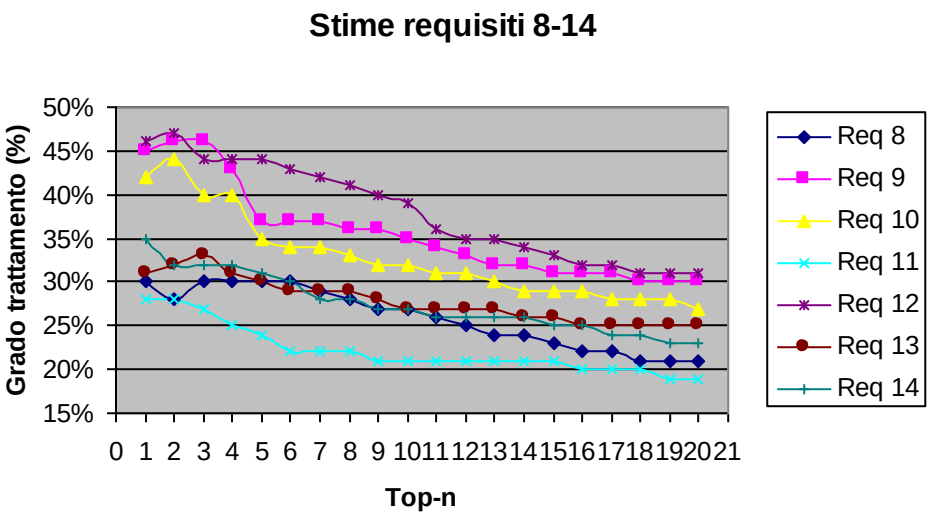


Figura 5.4 Andamento delle stime al variare del top-n

In generale, tranne che per alcuni requisiti particolari, gli andamenti sono piuttosto lineari. Dall'analisi di questi si nota che inizialmente il livello di trattamento dei requisiti cresce, mentre poi si abbassa gradualmente: da ciò emerge che inizialmente i report sono effettivamente relativi all'argomento considerato, ma poi ne vengono introdotti altri non correlati, per cui la stima globale tende a diminuire. Ciò è dimostrato dalle verifiche manuali sulla precisione, di cui di seguito si riporta la tabella completa (Tabella 5.1):

	Req 1	Req 2	Req 3	Req 4	Req 5	Req 6	Req 7
Top 20	1	1	1	0	1	0	1
	1	1	1	1	1	0	1
	1	1	1	1	1	1	1
	0	1	1	0	0	0	0
	0	0	0	1	0	0	0
	1	1	1	1	1	0	1
	0	1	1	1	1	0	1
	1	1	1	1	1	0	0
	0	1	1	0	0	0	1
	1	1	1	1	1	1	0
	0	0	0	0	0	0	1
	0	0	0	0	0	0	0
	0	1	1	0	1	0	1
	0	0	1	0	0	0	0

	0	0	1	0	1	1	1
	0	1	0	0	0	1	1
	0	0	0	0	0	0	1
	0	0	1	0	0	0	0
	0	1	0	0	0	1	1
	0	1	0	0	0	1	1

	Req 8	**Req 9**	**Req 10**	**Req 11**	**Req 12**	**Req 13**	**Req 14**
Top 20	0	1	1	0	0	0	1
	1	1	1	1	1	1	1
	1	1	1	1	1	0	1
	1	1	1	1	1	1	1
	0	1	1	1	0	1	1
	1	0	0	0	1	1	1
	0	1	0	0	1	1	1
	0	0	0	0	1	0	1
	0	0	0	1	1	0	1
	1	0	0	0	1	0	0
	0	0	1	1	0	0	0
	0	0	1	0	1	1	0
	0	1	0	0	0	0	1
	0	0	0	1	0	0	1
	0	0	0	0	0	1	0
	0	0	0	1	0	0	0
	0	1	0	0	0	0	0
	0	0	0	1	1	1	0
	0	1	0	1	0	1	1
	0	0	0	0	0	0	1

Tabella 5.1 Precisioni dei singoli report a top-20

Come si può vedere quasi la metà delle associazioni individuate dal tool sono in realtà falsi positivi, a dimostrazione che i risultati di un'analisi sintattica non sempre approssimano correttamente quelli di un'analisi semantica.

Infine, le *precisioni* calcolate per ogni top-*n* sono riportate di seguito. Inoltre per comodità di lettura si riportano anche i valori graficati in un diagramma (Tabella 5.2).

	Precisione
Top 1	57,14%
Top 2	75,00%
Top 3	80,95%
Top 4	76,79%
Top 5	70,00%
Top 6	70,24%
Top 7	69,39%
Top 8	66,96%
Top 9	64,29%
Top 10	63,57%
Top 11	59,74%
Top 12	56,55%
Top 13	55,49%
Top 14	53,06%
Top 15	51,90%
Top 16	50,45%
Top 17	48,32%
Top 18	47,22%
Top 19	47,37%
Top 20	46,43%

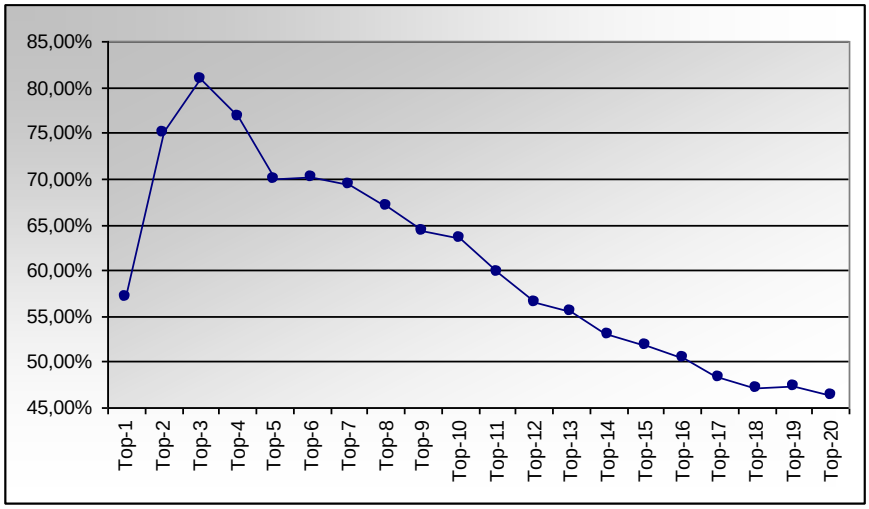


Tabella 5.2 Precisione globale per vari top-n e relativo diagramma

Dal grafico si nota che l'applicazione fornisce una precisione massima di circa l'81% a top-*3* e poi decresce, e questo è in effetti il comportamento atteso per via della sempre maggiore introduzione di report non attinenti all'aumentare del top-*n*.

5.4 Valutazioni finali

Le tabelle esposte nel paragrafo precedente sono di grande aiuto per trarre delle conclusioni, seppure preliminari, sull'effettiva affidabilità del tool *DevelopAider*.

La questione principale e più spinosa era quella sull'efficacia delle scelte inerenti le associazioni, ma uno sguardo alla Tabella 5.2 fa subito comprendere che tutto sommato il tool risponde in maniera corretta purché si scelga il valore giusto del top-*n*: il valore

massimo di *precisione* nel caso di studio *ArgoUML* si raggiunge per un'esecuzione a top-*3* ed è pari a circa l'80%. Tuttavia la sua efficacia si mantiene abbastanza elevata fino a top-*6* o top-*7* con *precisioni* prossime al 70%: ciò è dovuto al fatto che inizialmente sono aggiunti all'analisi report affini all'argomento del requisito. Superata questa soglia, la *precisione* globale comincia a scendere monotonicamente, e questo è il chiaro sintomo che in generale i report significativi per i requisiti sono ormai terminati.

In conclusione, in base al caso di test usato per la valutazione, il tool risulta effettivamente utile ad uno sviluppatore esperto dei progetti analizzati, in quanto quest'ultimo dovrebbe essere in grado di stimare il top-*n* più idoneo per un'analisi più efficace.

Capitolo 6 - Conclusioni

6.1 Considerazioni finali

Nel presente lavoro di tesi è stato trattato il problema della valutazione dello stato di avanzamento di un progetto software tramite l'analisi del grado di trattamento dei requisiti che lo definiscono.

Nel Capitolo 1 è stata data una definizione formale del termine *requisito*, è stata spiegata la sua utilità ed illustrato il suo ciclo di vita tipico. È stato poi spiegato il problema tipico dell'ingegnere del software: comprendere se i requisiti individuati sono stati effettivamente affrontati, ed in caso affermativo se possono essere considerati soddisfatti. Inoltre i requisiti possono anche evolvere nel loro ciclo di vita, e sono state quindi date, tra le altre, le definizioni di *requisiti duplicati* e *requisiti emergenti*: l'individuazione di tali categorie di requisiti impegna non poco l'ingegnere del software nella delicata fase di manutenzione. Il sistema realizzato cerca di essere un valido supporto per l'ottimizzazione degli sforzi dei tecnici in questo ambito.

Il Capitolo 2 ha dato un quadro vario ed esauriente sui due campi applicativi che più di tutti sono coinvolti nel trattamento dei requisiti in maniera avanzata:

- la *tracciabilità dei requisiti*;
- l'*information retrieval*.

Il primo aspetto è legato alla generazione ed alla memorizzazione di informazioni che permettano di stabilire legami logici tra requisiti stessi, e tra requisiti ed entità di implementazione. Sono stati illustrati i vari tipi di tracciabilità (pre-RS e post-RS,

backward e forward) e le problematiche che inevitabilmente si presentano (diversa interpretazione da parte delle persone); infine si è provato a fornire alcuni modelli di riferimento per la stesura di uno schema di tracciabilità efficace, in cui ognuno enfatizza aspetti diversi della tracciabilità ed è rivolto a diverse categorie di utenti.

Il secondo aspetto si occupa del recupero delle informazioni da testi per comprendere l'argomento trattato, ed è stato utilizzato dal sistema realizzato per estrarre il contenuto del materiale da analizzare per confrontarlo e valutarlo. Sono stati distinti i diversi punti di vista dell'information retrieval (sintattico e semantico) ed i vari parametri utilizzati per la sua valutazione, e sono state illustrate le fasi proprie dell'analisi del testo e della sua rappresentazione. Infine sono state presentate le strutture dati per la memorizzazione delle informazioni e le strategie di ricerca delle stesse maggiormente utilizzate.

Nel Capitolo 3 è stato illustrato il metodo utilizzato nel lavoro di tesi per risolvere le problematiche enunciate, consistente in quattro fasi:

1. organizzazione del materiale a disposizione in forma testuale (requisiti software e report sui bug pubblicati) secondo le tecniche proprie dell'information retrieval;
2. rielaborazione sintattica dello stesso e rappresentazione in uno spazio vettoriale;
3. verifica del trattamento dei requisiti tramite confronti tra le rappresentazioni;
4. rielaborazione del materiale non utilizzato per l'estrazione di parole utili al fine di individuare requisiti emergenti.

Nel capitolo 4 si è parlato dell'implementazione del sistema realizzato, attraversando tutti i punti metodologici del capitolo precedente e fornendo una spiegazione delle operazioni centrali per la loro realizzazione. Inoltre è stata dedicata una sezione alla spiegazione dell'interfaccia grafica del tool, fornendo un esempio d'uso. Tutto il materiale di corredo al software, come requisiti, casi d'uso, ecc., sono stati riportati in Appendice A.

Nel Capitolo 5 è stato affrontato l'aspetto della valutazione del sistema. Prima di tutto si è discusso del suo punto di debolezza conosciuto a priori, consistente in un'analisi sintattica piuttosto che semantica, ma le ricerche nel campo hanno constatato che gli sforzi connessi allo sviluppo di un'analisi semantica non producono risultati apprezzabilmente migliori rispetto a quelli di una sintattica. Successivamente sono stati illustrati in dettaglio i parametri coinvolti nella valutazione del sistema, e come sono stati utilizzati nel caso del sistema realizzato. Infine è stato condotto uno studio su un caso di test considerato abbastanza significativo, e sono stati confrontati i risultati sperimentali (riportati in Appendice B) con quelli ottenuti dalle verifiche manuali: è stato constatato che benché l'efficacia del sistema possa variare da requisito a requisito (e quindi potenzialmente tra vari casi di studio) essa si mantiene sempre mediamente a livelli alti (con una precisione superiore al 70%) e quindi il sistema risulta essere effettivamente di aiuto al processo di manutenzione software.

6.2 Sviluppi futuri

Il campo dell'ingegneria dei requisiti e la relativa fase di manutenzione sono aspetti molto delicati del processo di sviluppo software perché, facendo da ponte tra le necessità di persone (o gruppi di persone) e la rappresentazione formale delle funzionalità di un sistema, risentono dell'ambiguità del linguaggio naturale e quindi della possibile incomprensione tra i clienti e gli ingegneri del software. Anche il campo di supporto dell'information retrieval risente della difficoltà di trattamento del linguaggio naturale, e la conseguenza di ciò è l'impossibilità di condurre un'analisi totalmente automatica.

Come detto al paragrafo precedente, nel lavoro di tesi è stato fornito un supporto basato su un'analisi sintattica, che non vuole essere la soluzione definitiva, ma solo un mezzo per valutare una particolare strategia ed un punto di partenza per ulteriori ricerche. Sarebbe molto utile ed interessante provare soluzioni che prevedano l'uso di tecniche di analisi semantica e di recupero di informazioni di tipo probabilistico, raffrontandone poi i risultati con quelli prodotti dal sistema realizzato in questa sede. Sarebbe inoltre auspicabile in futuro coinvolgere ambiti a livelli più alti come quelli dell'*ingegneria della conoscenza*, facendo uso di *tecniche euristiche* per l'apprendimento e di *logiche fuzzy* per le decisioni.

Appendice A – UML

La presente Appendice fornisce la documentazione formale del tool *DevelopAider* secondo lo standard *UML*, ed è da considerarsi come completamento degli argomenti trattati nel Capitolo 4.

A.1 Requisiti funzionali

Requisito numero: F-P01 **Tipo: Funzionale**

Requisito: Modifica configurazione

Descrizione: L'applicazione, prima di poter procedere all'esecuzione delle procedure di analisi, ha bisogno del settaggio di alcuni parametri per un corretto e soddisfacente funzionamento. Il sistema deve quindi dare la possibilità di definire tali parametri, e successivamente su richiesta memorizzarli in maniera permanente in modo da poterli riutilizzare in successive esecuzioni. I parametri inseriti dovranno essere congruenti con le scelte effettuate, ed il sistema dovrà essere in grado di gestire eventuali errori nella loro definizione.

Input richiesto: Informazioni relative alla collocazione delle risorse fisiche: requisiti funzionali del software, report sui bug, strutture fisiche allocate su disco, ecc.
Informazioni relative all'analisi testuale: dizionario delle stopword, lingua utilizzata
Informazioni relative al supporto per la scoperta di requisiti emergenti: numero di parole chiave suggerite

Output desiderato: Accettazione dei parametri inseriti e, se desiderato, memorizzazione degli stessi in maniera permanente

Criterio di accettazione: Dati inseriti corretti e congruenti con le scelte effettuate

Aspettative collegate: Esecuzione più versatile e controllata dell'applicazione

Requisiti collegati: F-P02, F-P03, F-P04, F-P05, F-P06

Stato	**Release**	**Priorità**	**Stabilità**	**Livello comprensione**	**Categoria**
BOZZA	**1**	**RICHIESTO**	**STABILE**	**COMPLETO**	**SOFTWARE**
Versione:		**Data:**		**Compilato da:**	
1		**10/7/2007**		**Corbo Leonardo**	
Note:					

Requisito numero: F-P02 **Tipo: Funzionale**

Requisito: Richiamo configurazione

Descrizione: Il sistema deve essere in grado di richiamare i parametri precedentemente memorizzati in maniera permanente. Tali parametri richiamati andranno a sostituire quelli eventualmente modificati localmente, riportando il sistema presumibilmente in una condizione di default.

Input richiesto: Nessuno

Output desiderato: Modifica dei parametri relativi alla configurazione con quelli risalenti all'ultimo salvataggio

Criterio di accettazione: Nessuno

Aspettative collegate: Nessuna

Requisiti collegati: F-P01, F-P03, F-P04, F-P05, F-P06

Stato	**Release**	**Priorità**	**Stabilità**	**Livello comprensione**	**Categoria**
BOZZA	**1**	**RICHIESTO**	**STABILE**	**COMPLETO**	**SOFTWARE**
Versione:		**Data:**		**Compilato da:**	
1		**10/7/2007**		**Corbo Leonardo**	
Note:					

Requisito numero: F-P03 **Tipo: Funzionale**

Requisito: Visualizzazione trattamento requisiti

Descrizione: Il sistema deve essere in grado di effettuare un'analisi comparata tra i requisiti software di un'applicazione da verificare ed i relativi report sui bug rilasciati ufficialmente da un server Bugzilla, fornendo una stima sul grado di avanzamento dell'intero progetto tramite stime individuali su ogni requisito che lo identifica.

Input richiesto: Informazioni relative alla configurazione: collocazione delle risorse fisiche, dizionario delle stopword, lingua da utilizzare
Informazioni relative all'analisi: numero massimo di report da considerare per ogni requisito

Output desiderato: Visualizzazione per ogni requisito della stima sul grado di trattamento globale e dei report associati con il relativo livello di similitudine, ordinati in ordine di similitudine decrescente

Criterio di accettazione: Dati inseriti corretti e congruenti con le scelte effettuate

Aspettative collegate: Nessuna

Requisiti collegati: F-P01, F-P02, F-P04

Stato	**Release**	**Priorità**	**Stabilità**	**Livello comprensione**	**Categoria**
BOZZA	**1**	**RICHIESTO**	**STABILE**	**COMPLETO**	**SOFTWARE**
Versione:		**Data:**		**Compilato da:**	
1		**10/7/2007**		**Corbo Leonardo**	
Note:					

Requisito numero: F-P04 **Tipo: Funzionale**

Requisito: Visualizzazione associazioni report

Descrizione: Il sistema deve essere in grado di effettuare un'analisi comparata tra i requisiti software di un'applicazione da verificare ed i relativi report sui bug rilasciati ufficialmente da un server Bugzilla, fornendo le associazioni esistenti tra essi ordinate per report a supporto della scoperta di eventuali requisiti duplicati e/o emergenti.

Input richiesto: Informazioni relative alla configurazione: collocazione delle risorse fisiche, dizionario delle stopword, lingua da utilizzare
Informazioni relative all'analisi: numero massimo di report da considerare per ogni requisito

Output desiderato: Visualizzazione per ogni report dei requisiti associati con il relativo livello di similitudine, ordinati in ordine di similitudine decrescente

Criterio di accettazione: Dati inseriti corretti e congruenti con le scelte effettuate

Aspettative collegate: Nessuna

Requisiti collegati: F-P01, F-P02, F-P03

Stato	**Release**	**Priorità**	**Stabilità**	**Livello comprensione**	**Categoria**
BOZZA	**1**	**RICHIESTO**	**STABILE**	**COMPLETO**	**SOFTWARE**
Versione:		**Data:**		**Compilato da:**	
1		**10/7/2007**		**Corbo Leonardo**	
Note:					

Requisito numero: F-P05 **Tipo: Funzionale**

Requisito: Suggerimento keyword per requisiti emergenti

Descrizione: Il sistema deve essere in grado di isolare i report che non hanno associazioni con alcun requisito e suggerire, estraendole da questi, un certo numero di parole chiave utili per la scoperta di eventuali requisiti emergenti.

Input richiesto: Informazioni relative alla configurazione: collocazione delle risorse fisiche, dizionario delle stopword, lingua da utilizzare, numero di parole chiave da suggerire (*N*)

Output desiderato: Elenco delle *N* parole chiave a frequenza totale di occorrenza maggiore estratte dai report non associati, ordinato in ordine di frequenza decrescente

Criterio di accettazione: Dati inseriti corretti e congruenti con le scelte effettuate

Aspettative collegate: Nessuna

Requisiti collegati: F-P01, F-P02, F-P03, F-P06

Stato	**Release**	**Priorità**	**Stabilità**	**Livello comprensione**	**Categoria**
BOZZA	**1**	**RICHIESTO**	**STABILE**	**COMPLETO**	**SOFTWARE**
Versione:		**Data:**		**Compilato da:**	
1		**10/7/2007**		**Corbo Leonardo**	
Note:					

Requisito numero: F-P06 **Tipo: Funzionale**

Requisito: Analisi keyword per requisiti emergenti

Descrizione: Il sistema deve essere in grado di isolare i report che non hanno associazioni con alcun requisito e rilevare la presenza al loro interno di una particolare parola chiave scelta tra quelle suggerite dal sistema o indicata esplicitamente dall'utente per la scoperta di eventuali requisiti emergenti.

Input richiesto: Informazioni relative alla configurazione: collocazione delle risorse fisiche, dizionario delle stopword, lingua da utilizzare, numero di parole chiave da suggerire (*N*)
Informazioni relative alla ricerca: parola chiave da cercare

Output desiderato: Elenco dei report non associati contenenti la parola chiave desiderata con le relative frequenze di occorrenza, e ordinati in ordine di frequenza decrescente

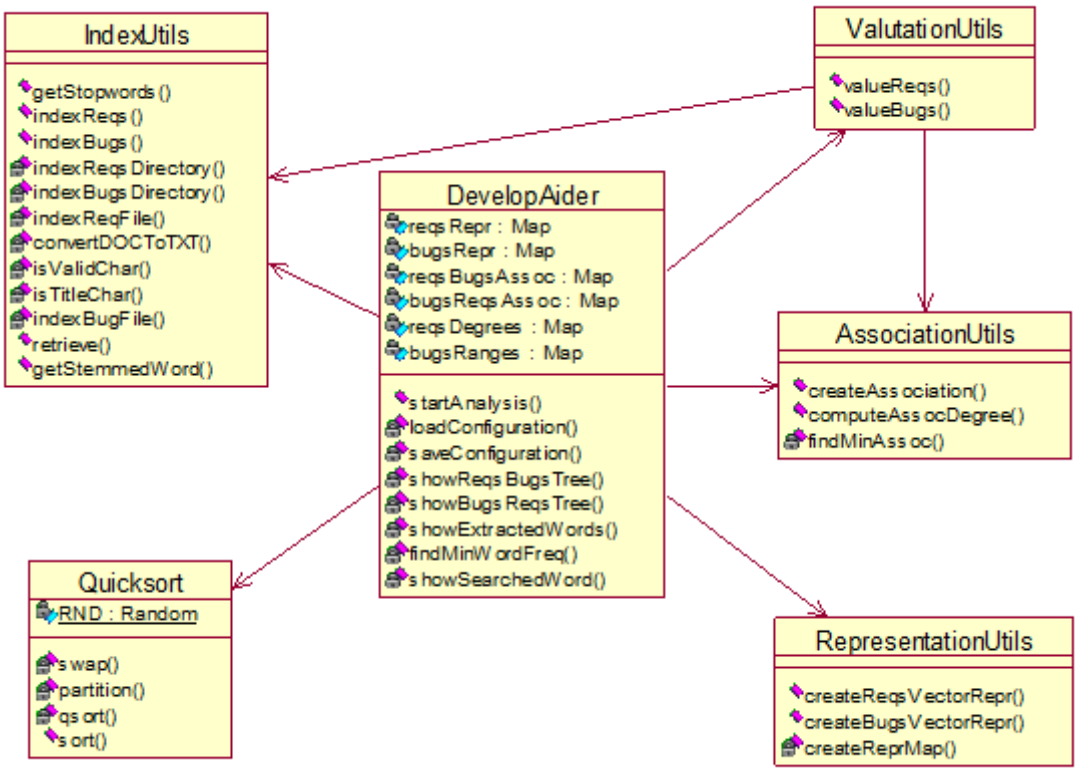
Criterio di accettazione: Dati inseriti corretti e congruenti con le scelte effettuate

Aspettative collegate: Nessuna

Requisiti collegati: F-P01, F-P02, F-P03, F-P05

Stato	**Release**	**Priorità**	**Stabilità**	**Livello comprensione**	**Categoria**
BOZZA	**1**	**RICHIESTO**	**STABILE**	**COMPLETO**	**SOFTWARE**
Versione: **1**		**Data:** **10/7/2007**		**Compilato da:** **Corbo Leonardo**	
Note:					

A.2 Diagramma delle classi



A.3 Casi d'uso

A.3.1 Modifica configurazione

Titolo: Modifica configurazione

Autore: Corbo Leonardo

Descrizione: Procedura tramite cui è possibile definire buona parte dei parametri utilizzati dal sistema per un suo corretto e soddisfacente funzionamento. Successivamente quest'ultimo permette su richiesta di memorizzarli in maniera permanente in modo da poterli riutilizzare in successive esecuzioni. I parametri inseriti dovranno essere congruenti con le scelte effettuate, ed il sistema dovrà essere in grado di gestire eventuali errori nella loro definizione.

Attori: Utente

Pre-condizioni: Nessuna

Post-condizioni: Modifica dei parametri del sistema ed eventuale memorizzazione permanente

Scenario principale: Modifica locale parametri effettuata

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.

2. L’utente modifica i dati riguardanti la locazione fisica delle risorse da utilizzare e delle strutture allocate, utilizzando dei browser di cartelle:
    - directory contenente i file dei requisiti software;
    - directory contenente i file dei report sui bug;
    - directory atta a contenere l’indice dei requisiti che sarà creato;
    - directory atta a contenere l’indice dei report che sarà creato;
    - directory atta a contenere l’indice dei report non associati che sarà creato.
3. L’utente modifica i dati riguardanti l’analisi testuale, utilizzando browser di file e campi con opzioni prestabilite:
    - file contenente la lista delle stopword;
    - lingua da utilizzare.
4. L’utente modifica i dati relativi al supporto per la scoperta di requisiti emergenti, specificando il numero di parole chiave che il sistema deve suggerire.
5. Il sistema verifica la correttezza dei dati specificati dall’utente e gli consente di procedere nelle sue operazioni.
6. L’utente utilizza le altre funzionalità del sistema accedendo alle altre finestre.

Scenario 1: Parametro errato

1. L’utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L’utente modifica i dati riguardanti la locazione fisica delle risorse da utilizzare e delle strutture allocate, utilizzando dei browser di cartelle:

- directory contenente i file dei requisiti software;
- directory contenente i file dei report sui bug;
- directory atta a contenere l'indice dei requisiti che sarà creato;
- directory atta a contenere l'indice dei report che sarà creato;
- directory atta a contenere l'indice dei report non associati che sarà creato.

3. L'utente modifica i dati riguardanti l'analisi testuale, utilizzando browser di file e campi con opzioni prestabilite:
   - file contenente la lista delle stopword;
   - lingua da utilizzare.
4. L'utente modifica i dati relativi al supporto per la scoperta di requisiti emergenti, specificando il numero di parole chiave che il sistema deve suggerire.
5. Il sistema controlla la correttezza dei dati specificati dall'utente e non gli consente di procedere nelle sue operazioni perché ha inserito un parametro errato (ad esempio un numero di parole suggerite negativo).
6. L'utente deve correggere il dato reinserendolo.

Scenario alternativo: Modifica permanente parametri effettuata

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati riguardanti la locazione fisica delle risorse da utilizzare e delle strutture allocate, utilizzando dei browser di cartelle:
   - directory contenente i file dei requisiti software;

- directory contenente i file dei report sui bug;
- directory atta a contenere l'indice dei requisiti che sarà creato;
- directory atta a contenere l'indice dei report che sarà creato;
- directory atta a contenere l'indice dei report non associati che sarà creato.

3. L'utente modifica i dati riguardanti l'analisi testuale, utilizzando browser di file e campi con opzioni prestabilite:
   - file contenente la lista delle stopword;
   - lingua da utilizzare.
4. L'utente modifica i dati relativi al supporto per la scoperta di requisiti emergenti, specificando il numero di parole chiave che il sistema deve suggerire.
5. Il sistema verifica la correttezza dei dati specificati dall'utente e gli consente di procedere nelle sue operazioni.
6. L'utente seleziona di memorizzare permanentemente le informazioni specificate.
7. Il sistema salva le informazioni in un file di configurazione, eliminando quelle precedentemente contenute o creando un file nuovo se inesistente.
8. L'utente utilizza le altre funzionalità del sistema accedendo alle altre finestre.

A.3.2 Recupero configurazione

Titolo: Recupero configurazione

Autore: Corbo Leonardo

Descrizione: Procedura tramite cui è possibile richiamare i parametri precedentemente memorizzati dal sistema in maniera permanente. Tali parametri richiamati andranno a sostituire quelli eventualmente modificati localmente, riportando il sistema presumibilmente in una condizione di default.

Attori: Utente

Pre-condizioni: Esistenza del file di configurazione e dei relativi parametri memorizzati

Post-condizioni: Ripristino dei parametri di configurazione agli ultimi valori salvati

Scenario principale: Richiamo parametri

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente seleziona di richiamare i parametri memorizzati nel file di configurazione.
3. Il sistema apre il file di configurazione e carica i parametri in esso contenuti.
4. Il sistema aggiorna i valori dei parametri nella finestra di configurazione per darne immediata notifica all'utente.
5. L'utente utilizza le altre funzionalità del sistema accedendo alle altre finestre.

Scenario 1: File di configurazione inesistente

1. L’utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L’utente seleziona di richiamare i parametri memorizzati nel file di configurazione.
3. Il sistema tenta di aprire il file di configurazione ma non lo trova.
4. Il sistema avverte l’utente dell’errore e continua a mantenere i valori dei parametri attualmente impostati.
5. L’utente decide di utilizzare i parametri attuali, oppure definisce nuovi parametri e li salva su un nuovo file di configurazione.

A.3.3 Visualizzazione trattamento requisiti

Titolo: Visualizzazione trattamento requisiti

Autore: Corbo Leonardo

Descrizione: Procedura tramite cui è possibile effettuare un'analisi comparata tra i requisiti software di un'applicazione da verificare ed i relativi report sui bug rilasciati ufficialmente da un server Bugzilla. Al termine si ottiene una stima sul grado di avanzamento dell'intero progetto tramite stime individuali su ogni requisito che lo identifica.

Attori: Utente

Pre-condizioni: Parametri per l'analisi inseriti correttamente

Post-condizioni: Visualizzazione per ogni requisito della stima sul grado di trattamento globale e dei report associati con il relativo livello di similitudine

Scenario principale: Visualizzazione ottenuta

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.

5. Il sistema verifica la correttezza dei parametri inseriti, conduce l'analisi e presenta a video all'utente una struttura ad albero che riporta per ogni requisito la stima sul grado di trattamento globale ed i report associati con il relativo livello di similitudine, ordinati in ordine di similitudine decrescente.

Scenario 1: Parametro errato

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.
5. Il sistema rileva un errore in qualche parametro inserito, e ferma il processo dandone avviso all'utente.

A.3.4 Visualizzazione associazioni report

Titolo: Visualizzazione associazioni report

Autore: Corbo Leonardo

Descrizione: Procedura tramite cui è possibile effettuare un'analisi comparata tra i requisiti software di un'applicazione da verificare ed i relativi report sui bug rilasciati ufficialmente da un server Bugzilla. Al termine si ottengono le associazioni esistenti tra essi ordinate per report a supporto della scoperta di eventuali requisiti duplicati e/o emergenti.

Attori: Utente

Pre-condizioni: Parametri per l'analisi inseriti correttamente

Post-condizioni: Visualizzazione per ogni report dei requisiti associati con il relativo livello di similitudine

Scenario principale: Visualizzazione ottenuta

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.

5. Il sistema verifica la correttezza dei parametri inseriti, conduce l'analisi e presenta a video all'utente una struttura ad albero che riporta per ogni report i requisiti associati con il relativo livello di similitudine, ordinati in ordine di similitudine decrescente.

Scenario 1: Parametro errato

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.
5. Il sistema rileva un errore in qualche parametro inserito, e ferma il processo dandone avviso all'utente.

A.3.5 Suggerimento keyword per requisiti emergenti

Titolo: Suggerimento keyword per requisiti emergenti

Autore: Corbo Leonardo

Descrizione: Procedura tramite cui è possibile isolare i report che non hanno associazioni con alcun requisito e suggerire, estraendole da questi, un certo numero di parole chiave utili per la scoperta di eventuali requisiti emergenti.

Attori: Utente

Pre-condizioni: Parametri per l'analisi inseriti correttamente

Post-condizioni: Visualizzazione di un elenco di parole chiave a frequenza totale di occorrenza maggiore estratte dai report non associati

Scenario principale: Visualizzazione ottenuta

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare (tra cui in particolare il numero di parole suggerite per la scoperta di requisiti emergenti *N*) e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.
5. Il sistema verifica la correttezza dei parametri inseriti e conduce l'analisi.
6. L'utente passa alla finestra relativa al supporto per la scoperta di requisiti emergenti.

7. Il sistema presenta a video all'utente un elenco delle *N* parole chiave a frequenza totale di occorrenza maggiore estratte dai report non associati, ordinato in ordine di frequenza decrescente.

Scenario 1: Parametro errato

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare (tra cui in particolare il numero di parole suggerite per la scoperta di requisiti emergenti *N*) e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.
5. Il sistema rileva un errore in qualche parametro inserito, e ferma il processo dandone avviso all'utente.

A.3.6 Analisi keyword per requisiti emergenti

Titolo: Analisi keyword per requisiti emergenti

Autore: Corbo Leonardo

Descrizione: Procedura tramite cui è possibile isolare i report che non hanno associazioni con alcun requisito e rilevare la presenza al loro interno di una particolare parola chiave scelta tra quelle suggerite dal sistema o indicata esplicitamente dall'utente per la scoperta di eventuali requisiti emergenti.

Attori: Utente

Pre-condizioni: Parametri per l'analisi inseriti correttamente

Post-condizioni: Visualizzazione di un elenco di report non associati contenenti la parola chiave desiderata con le relative frequenze di occorrenza.

Scenario principale: Visualizzazione di una parola suggerita

1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare (tra cui in particolare il numero di parole suggerite per la scoperta di requisiti emergenti *N*) e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.
5. Il sistema verifica la correttezza dei parametri inseriti e conduce l'analisi.

6. L’utente passa alla finestra relativa al supporto per la scoperta di requisiti emergenti.
7. Il sistema presenta a video all’utente un elenco delle *N* parole chiave a frequenza totale di occorrenza maggiore estratte dai report non associati, ordinato in ordine di frequenza decrescente.
8. L’utente seleziona una delle parole suggerite di cui desidera vedere la distribuzione delle frequenze di occorrenza e ne dà conferma al sistema.
9. Il sistema effettua l’analisi della parola scelta e visualizza a video l’elenco dei report non associati contenenti quest’ultima con le relative frequenze di occorrenza, ordinati in ordine di frequenza decrescente.

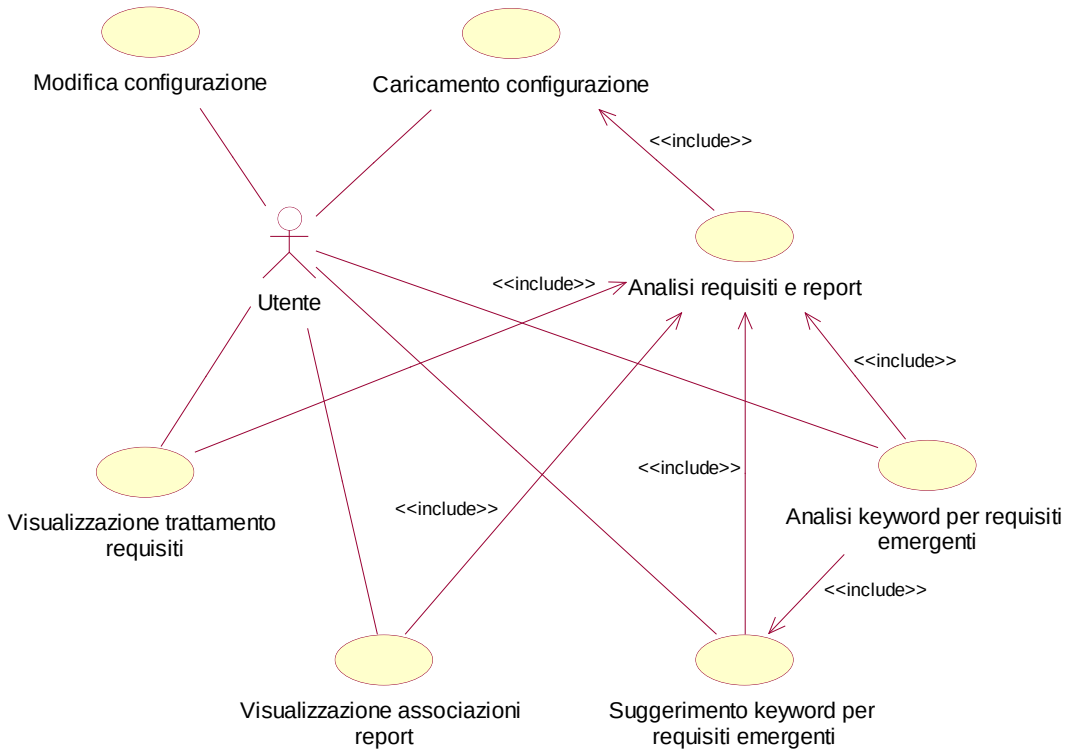
Scenario 1: Parametro errato

1. L’utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L’utente modifica i dati che vuole cambiare (tra cui in particolare il numero di parole suggerite per la scoperta di requisiti emergenti *N*) e lascia inalterati gli altri.
3. L’utente passa alla finestra relativa all’analisi delle relazioni tra requisiti e report.
4. L’utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l’analisi.
5. Il sistema rileva un errore in qualche parametro inserito, e ferma il processo dandone avviso all’utente.

Scenario alternativo: Visualizzazione di una parola specifica

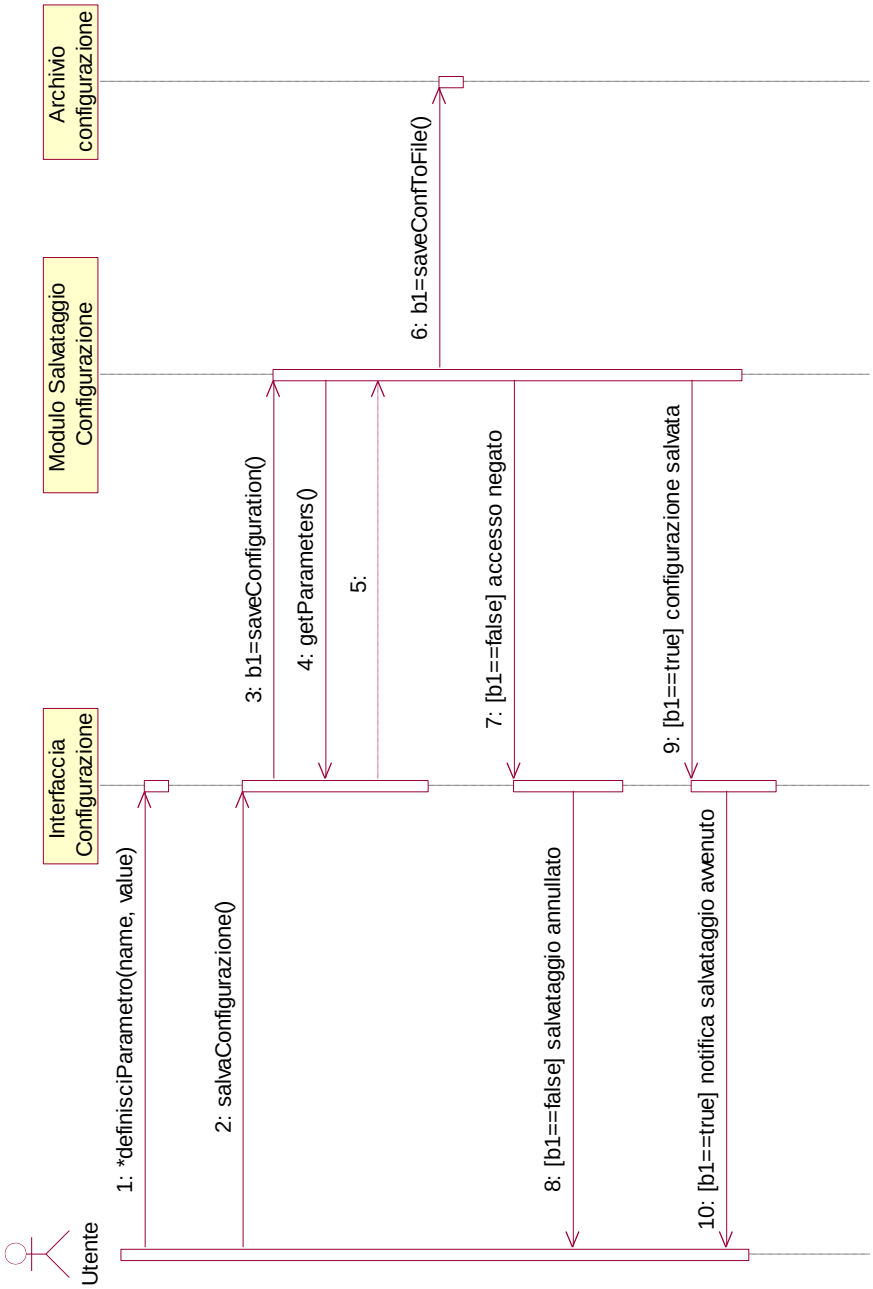
1. L'utente accede alla finestra di configurazione, che riporta tutti i parametri attualmente impostati per farne prendere visione.
2. L'utente modifica i dati che vuole cambiare (tra cui in particolare il numero di parole suggerite per la scoperta di requisiti emergenti *N*) e lascia inalterati gli altri.
3. L'utente passa alla finestra relativa all'analisi delle relazioni tra requisiti e report.
4. L'utente imposta il valore desiderato del *top-N* (numero massimo di report presi in considerazione per ogni requisito), e dà conferma scegliendo di avviare l'analisi.
5. Il sistema verifica la correttezza dei parametri inseriti e conduce l'analisi.
6. L'utente passa alla finestra relativa al supporto per la scoperta di requisiti emergenti.
7. Il sistema presenta a video all'utente un elenco delle *N* parole chiave a frequenza totale di occorrenza maggiore estratte dai report non associati, ordinato in ordine di frequenza decrescente.
8. L'utente decide di non avvalersi dei suggerimenti del sistema e specifica in un apposito campo una parola particolare, dandone alla fine conferma al sistema.
9. Il sistema effettua l'analisi della parola inserita e visualizza a video l'elenco dei report non associati contenenti quest'ultima con le relative frequenze di occorrenza, ordinati in ordine di frequenza decrescente.

A.4 Diagramma dei casi d'uso

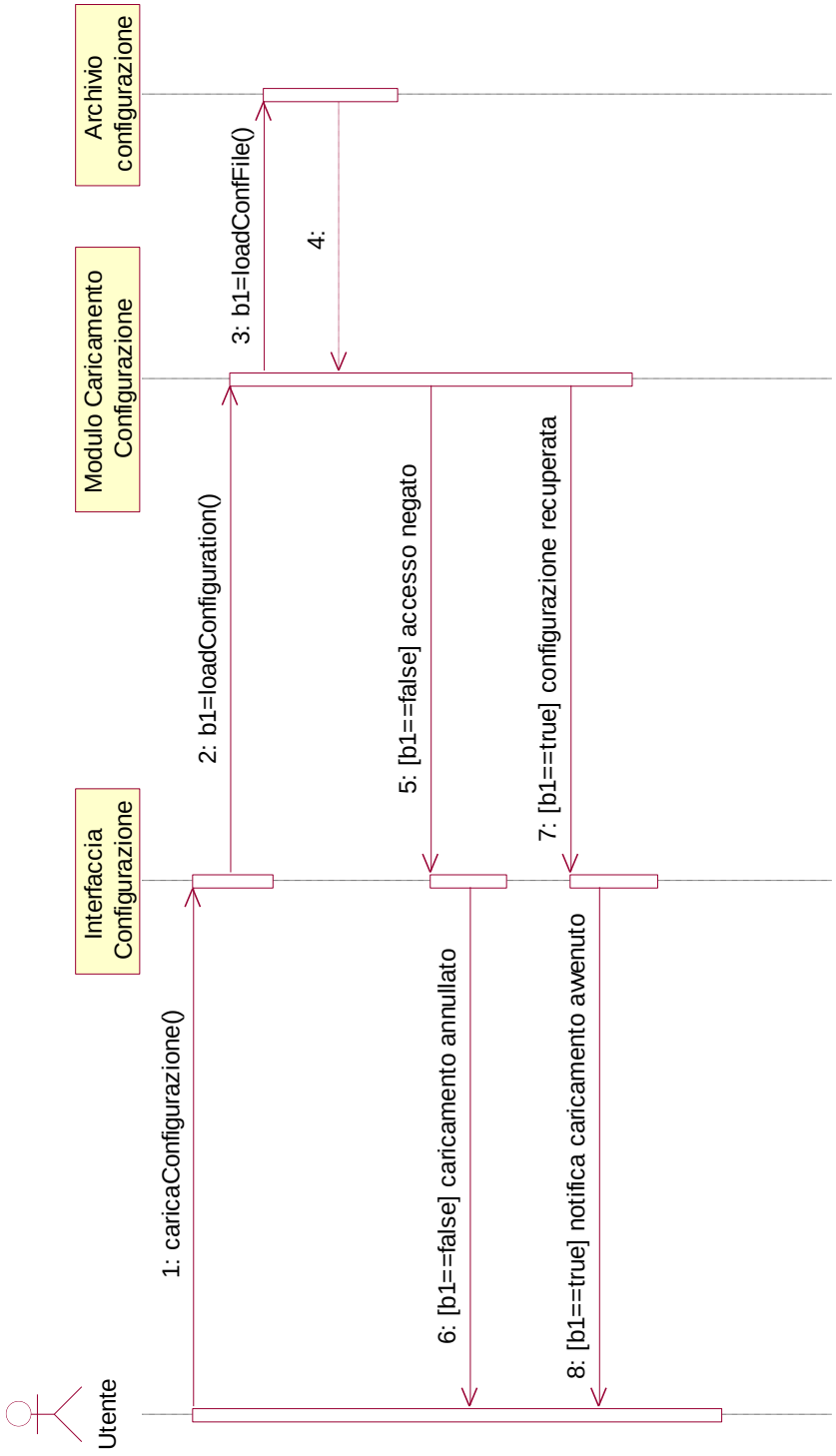


A.5 Diagrammi di sequenza

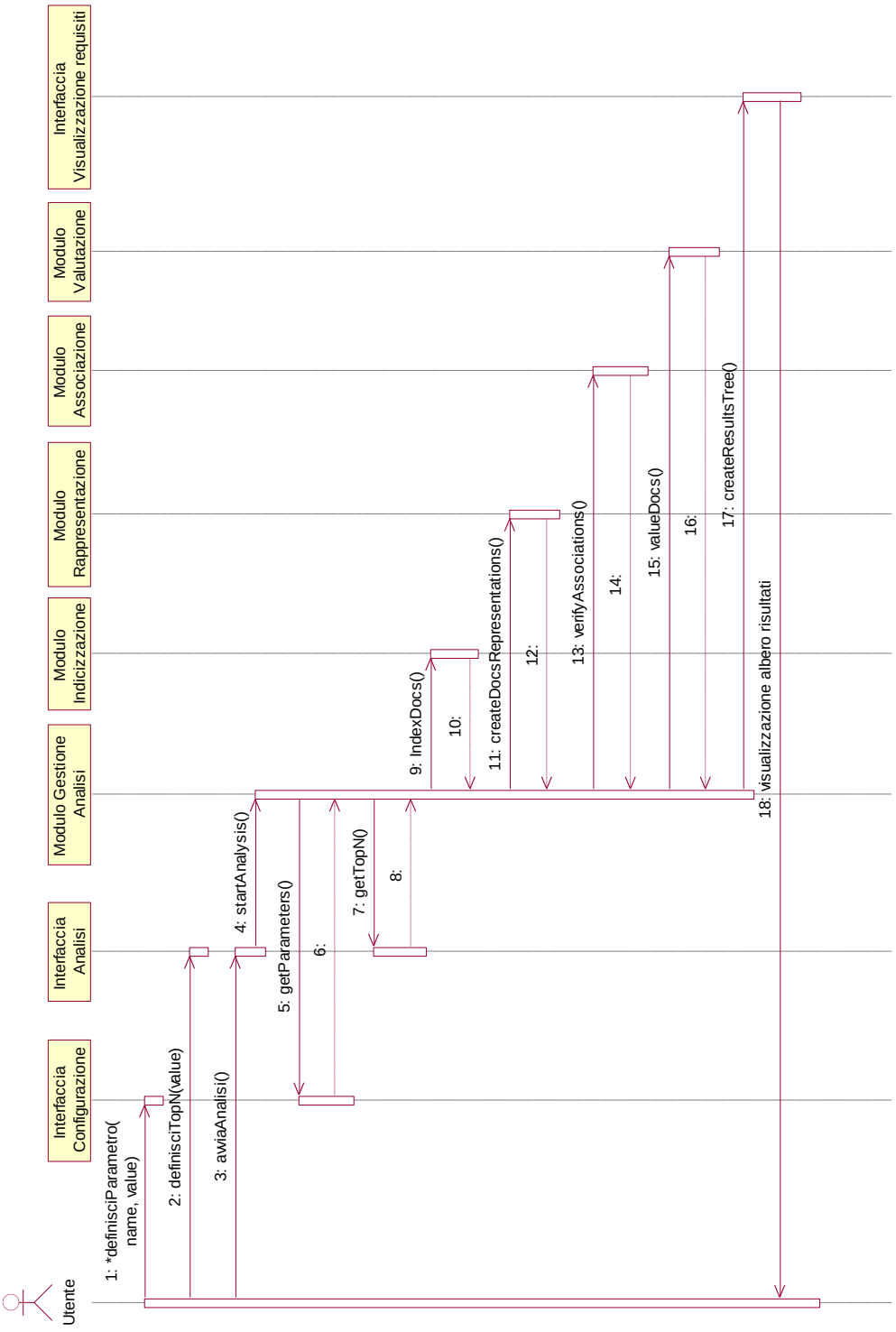
A.5.1 Modifica configurazione



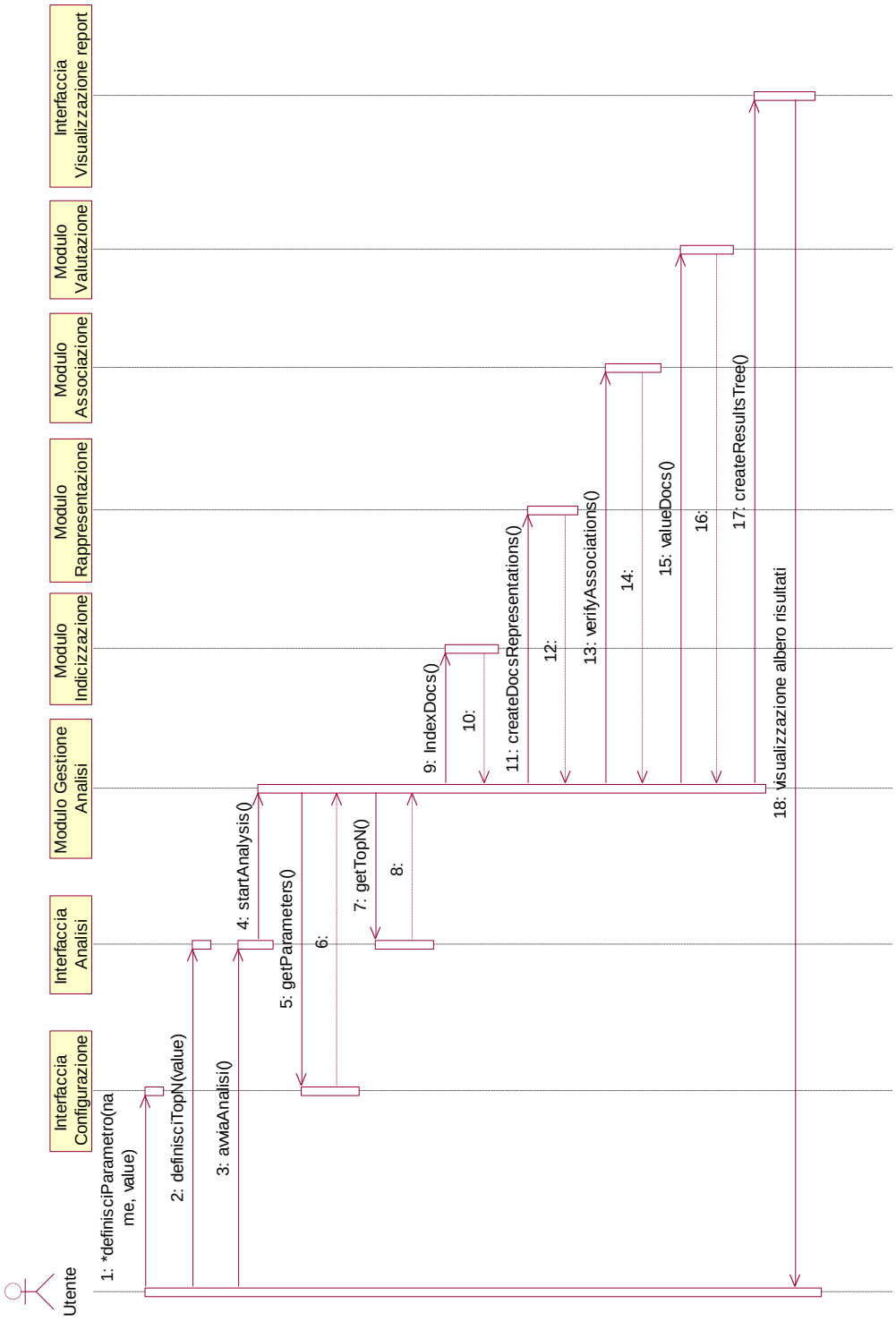
A.5.2 Caricamento configurazione



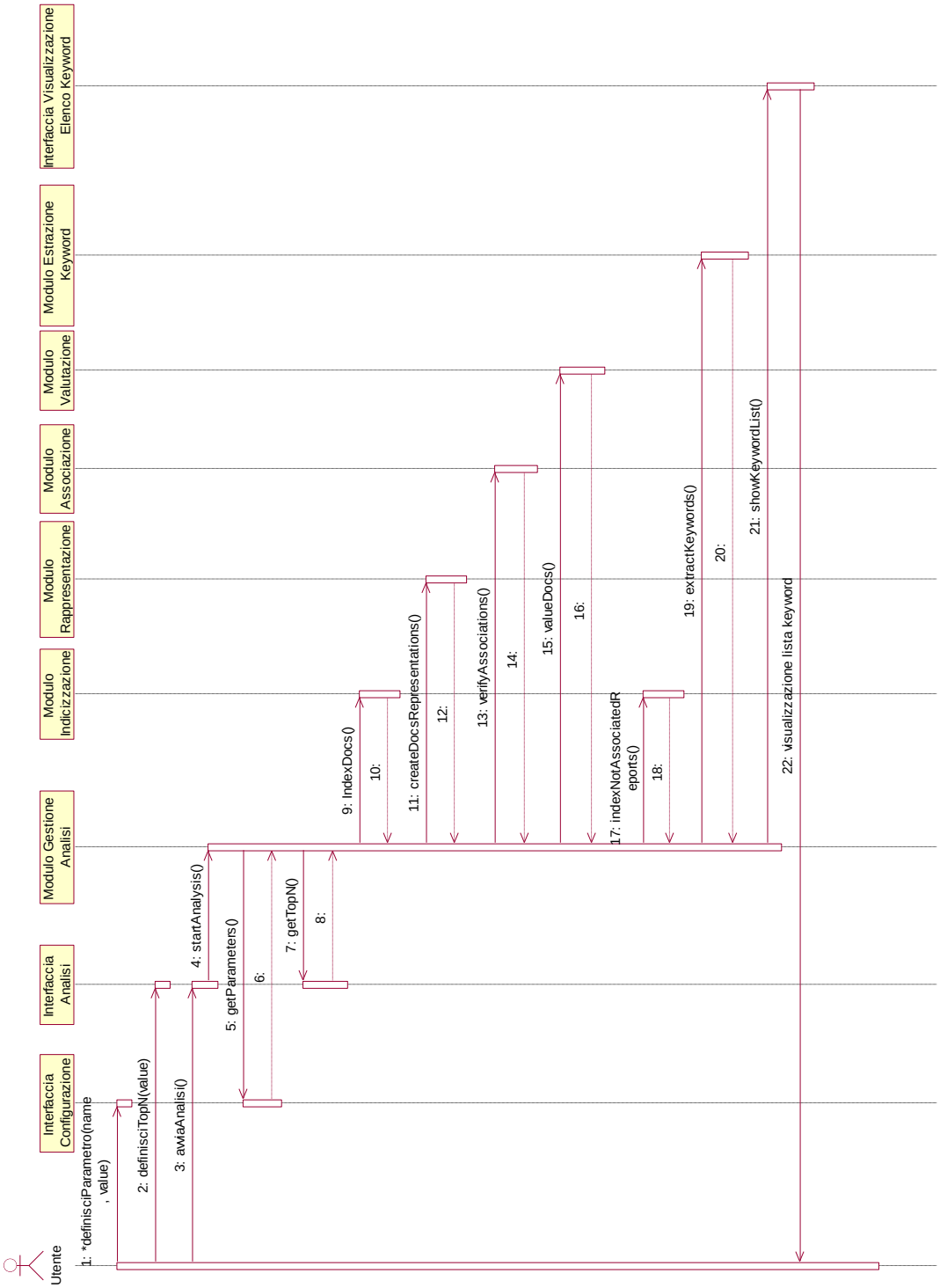
A.5.3 Visualizzazione trattamento requisiti



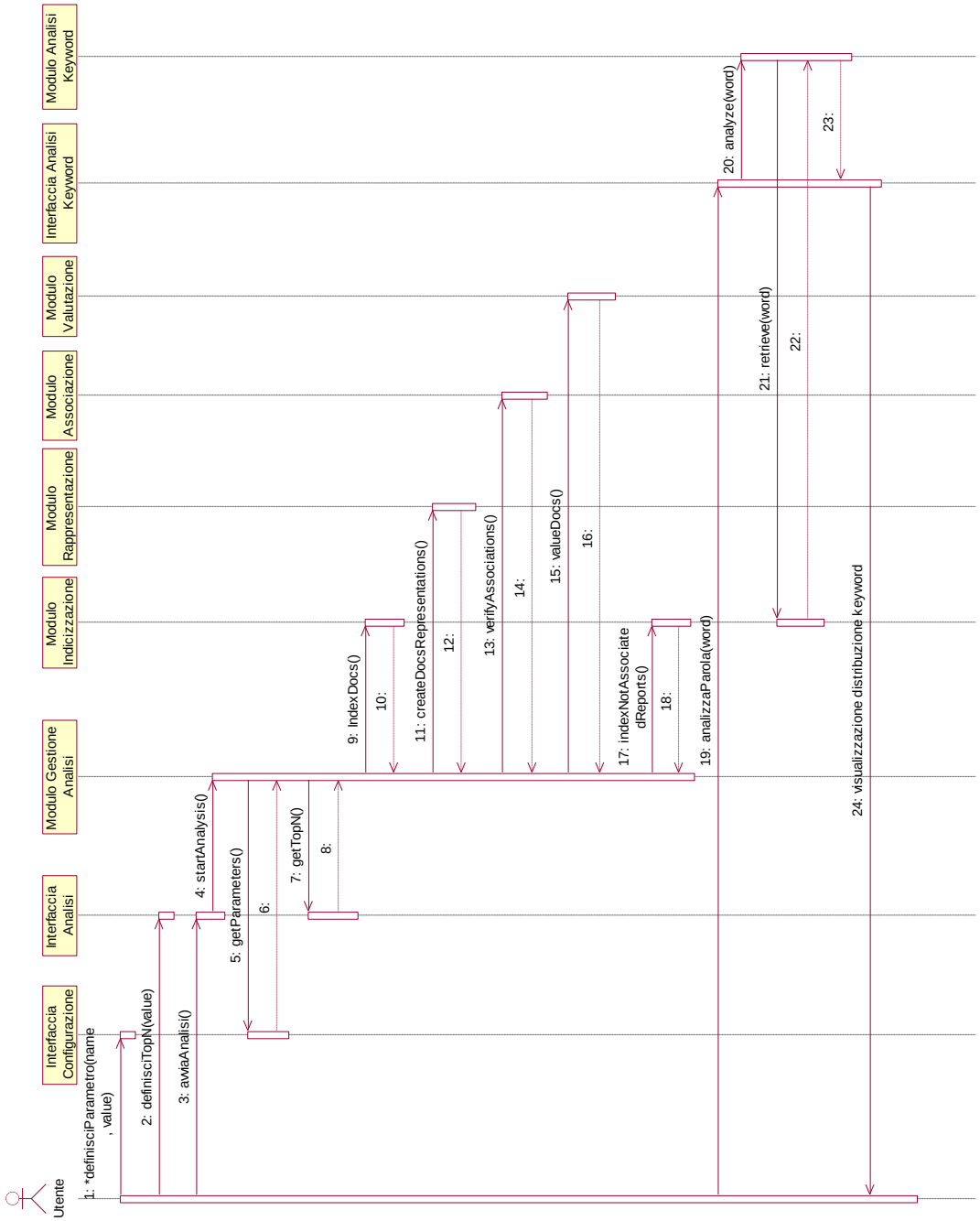
A.5.4 Visualizzazione associazioni report



A.5.5 Suggerimento keyword per requisiti emergenti



A.5.6 Analisi keyword per requisiti emergenti



Appendice B – Test di valutazione

B.1 Associazioni sperimentali a top-20

	Req 1	**Req 2**	**Req 3**	**Req 4**	**Req 5**
Top 20	2558 (30%)	595 (19%)	595 (21%)	1908 (31%)	1961 (17%)
	2105 (14%)	2105 (14%)	276 (16%)	373 (25%)	595 (16%)
	771 (12%)	135 (14%)	2558 (14%)	1549 (17%)	276 (13%)
	968 (12%)	276 (14%)	2105 (13%)	212 (16%)	2105 (10%)
	1191 (10%)	623 (14%)	623 (12%)	402 (15%)	2558 (10%)
	402 (10%)	771 (14%)	771 (11%)	281 (12%)	135 (10%)
	682 (10%)	2558 (13%)	135 (11%)	2558 (11%)	2182 (9%)
	595 (9%)	950 (9%)	1549 (11%)	86 (11%)	281 (9%)
	496 (9%)	402 (8%)	950 (10%)	396 (10%)	1908 (8%)
	396 (9%)	396 (8%)	402 (8%)	1191 (9%)	402 (8%)
	2595 (9%)	496 (7%)	496 (8%)	712 (7%)	771 (7%)
	265 (9%)	265 (7%)	265 (7%)	595 (7%)	1049 (6%)
	828 (9%)	1549 (7%)	1961 (7%)	2586 (6%)	96 (6%)
	2182 (8%)	968 (7%)	818 (7%)	2309 (6%)	979 (5%)
	2520 (8%)	2520 (6%)	415 (7%)	35 (6%)	682 (5%)
	331 (8%)	1961 (6%)	506 (7%)	309 (6%)	321 (5%)
	2309 (8%)	2182 (5%)	968 (6%)	569 (6%)	496 (5%)
	2006 (8%)	1028 (5%)	2182 (6%)	2520 (6%)	623 (5%)
	1075 (8%)	1010 (5%)	1028 (6%)	2006 (6%)	265 (5%)
	217 (8%)	382 (5%)	2520 (6%)	1947 (5%)	1549 (5%)

	Req 6	**Req 7**	**Req 8**	**Req 9**	**Req 10**
Top 20	2558 (22%)	2558 (25%)	1908 (30%)	2558 (45%)	2558 (42%)
	373 (17%)	968 (13%)	402 (16%)	496 (22%)	496 (21%)
	212 (16%)	388 (10%)	1549 (16%)	828 (19%)	968 (16%)
	2182 (15%)	496 (10%)	373 (16%)	968 (18%)	828 (15%)
	968 (14%)	623 (9%)	2558 (15%)	1267 (15%)	1267 (14%)
	496 (14%)	265 (9%)	86 (13%)	265 (14%)	506 (13%)
	265 (12%)	2105 (9%)	712 (11%)	331 (13%)	265 (13%)
	1908 (12%)	828 (9%)	331 (10%)	506 (13%)	402 (12%)
	2595 (12%)	331 (9%)	265 (9%)	2595 (13%)	388 (11%)
	402 (12%)	2182 (9%)	1191 (9%)	402 (13%)	217 (11%)
	331 (12%)	2309 (9%)	415 (8%)	388 (12%)	682 (11%)
	1075 (11%)	415 (8%)	212 (8%)	771 (12%)	331 (11%)
	506 (11%)	217 (8%)	2595 (8%)	845 (11%)	2595 (10%)
	2006 (10%)	506 (8%)	257 (7%)	217 (10%)	845 (9%)
	640 (10%)	2595 (8%)	396 (7%)	595 (10%)	2309 (9%)
	281 (10%)	1267 (8%)	1961 (7%)	2309 (10%)	1075 (9%)
	388 (10%)	402 (8%)	388 (7%)	2182 (10%)	2182 (8%)

	396 (9%) 1191 (9%) 1267 (9%)	682 (7%) 738 (7%) 1049 (7%)	1936 (7%) 623 (7%) 771 (7%)	1075 (10%) 682 (10%) 2520 (9%)	771 (8%) 1191 (7%) 2520 (7%)

	Req 11	**Req 12**	**Req 13**	**Req 14**
Top 20	2558 (28%)	2558 (46%)	2558 (31%)	2558 (35%)
	331 (14%)	496 (22%)	2595 (17%)	968 (15%)
	388 (11%)	968 (18%)	195 (14%)	265 (14%)
	968 (10%)	265 (17%)	968 (14%)	2595 (13%)
	2309 (9%)	1191 (17%)	331 (13%)	388 (13%)
	1267 (9%)	388 (16%)	828 (13%)	331 (13%)
	506 (8%)	506 (16%)	1317 (13%)	1267 (12%)
	265 (8%)	828 (15%)	569 (12%)	828 (11%)
	2520 (8%)	217 (15%)	217 (12%)	506 (11%)
	828 (8%)	331 (14%)	265 (12%)	623 (11%)
	1075 (8%)	1267 (13%)	402 (11%)	402 (9%)
	135 (7%)	195 (12%)	595 (11%)	697 (8%)
	2595 (7%)	682 (12%)	682 (11%)	2309 (8%)
	2182 (7%)	845 (11%)	506 (11%)	496 (8%)
	1191 (6%)	402 (11%)	1049 (10%)	712 (8%)
	496 (6%)	1936 (11%)	276 (10%)	845 (8%)
	845 (6%)	2595 (10%)	2182 (10%)	682 (7%)
	1317 (6%)	1075 (9%)	96 (9%)	257 (7%)
	217 (6%)	595 (9%)	496 (9%)	415 (7%)
	1936 (6%)	712 (9%)	2309 (9%)	1075 (6%)

Tabella B.1 Livelli di associazione a top-20

B.2 Stime sui requisiti sperimentali

	Req 1	**Req 2**	**Req 3**	**Req 4**	**Req 5**	**Req 6**	**Req 7**
Top 1	30%	19%	21%	31%	17%	22%	25%
Top 2	33%	22%	21%	36%	22%	26%	24%
Top 3	31%	24%	23%	36%	21%	28%	24%
Top 4	29%	23%	25%	36%	22%	29%	24%
Top 5	29%	24%	25%	33%	22%	29%	24%
Top 6	27%	23%	24%	32%	22%	30%	24%
Top 7	27%	25%	25%	31%	22%	29%	24%
Top 8	26%	24%	25%	30%	22%	29%	24%
Top 9	26%	23%	25%	29%	22%	29%	23%
Top 10	26%	22%	23%	29%	21%	28%	23%
Top 11	25%	22%	23%	28%	20%	28%	23%
Top 12	25%	21%	23%	26%	19%	28%	22%
Top 13	25%	21%	22%	21%	19%	27%	22%

Top 14	24%	21%	22%	21%	18%	27%	21%
Top 15	24%	21%	21%	21%	18%	27%	21%
Top 16	23%	20%	20%	21%	18%	27%	20%
Top 17	23%	20%	20%	20%	18%	26%	20%
Top 18	23%	20%	20%	20%	17%	26%	20%
Top 19	23%	19%	20%	20%	17%	26%	19%
Top 20	22%	19%	19%	20%	17%	25%	19%

	Req 8	**Req 9**	**Req 10**	**Req 11**	**Req 12**	**Req 13**	**Req 14**
Top 1	30%	45%	42%	28%	46%	31%	35%
Top 2	28%	46%	44%	28%	47%	32%	32%
Top 3	30%	46%	40%	27%	44%	33%	32%
Top 4	30%	43%	40%	25%	44%	31%	32%
Top 5	30%	37%	35%	24%	44%	30%	31%
Top 6	30%	37%	34%	22%	43%	29%	30%
Top 7	29%	37%	34%	22%	42%	29%	28%
Top 8	28%	36%	33%	22%	41%	29%	28%
Top 9	27%	36%	32%	21%	40%	28%	27%
Top 10	27%	35%	32%	21%	39%	27%	27%
Top 11	26%	34%	31%	21%	36%	27%	26%
Top 12	25%	33%	31%	21%	35%	27%	26%
Top 13	24%	32%	30%	21%	35%	27%	26%
Top 14	24%	32%	29%	21%	34%	26%	26%
Top 15	23%	31%	29%	21%	33%	26%	25%
Top 16	22%	31%	29%	20%	32%	25%	25%
Top 17	22%	31%	28%	20%	32%	25%	24%
Top 18	21%	30%	28%	20%	31%	25%	24%
Top 19	21%	30%	28%	19%	31%	25%	23%
Top 20	21%	30%	27%	19%	31%	25%	23%

Tabella B.2 Stime trattamento requisiti a top-20

Bibliografia

1. Ian SOMMERVILLE

   “Software Engineering”, 5th edition

2. M. EDWARDS, S. HOWELL

   “A methodology for Requirements Specification and Traceability for Large Real-Time Complex Systems”

   Technical Report, Naval Surface Warfare Center, 1992

3. J.D. PALMER

   “Traceability, Software Requirements Eng.”

   R.H. Thayer and M. Dorfman eds., 1997

4. O. GOTEL, A. FINKELSTEIN

   “An analysis of the requirements traceability problem”

   IEEE 1994

5. O. GOTEL, A. FINKELSTEIN

   “Contribution Structures”

   Proc. Second Int'l Symp. Requirements Eng., 1995

6. B. RAMESH

”Reference models for requirements traceability”

IEEE transactions on software engineering, vol. 27, no. 1, january 2001

7. A. SCHEER

“Business Process Engineering: Reference Models for Industrial Enterprises”

Springer-Verlag, 1998

8. G. CANFORA

“Tracing Object-Oriented Code into Functional Requirements”

9. CLELAND-HUANG, SETTIMI, DUAN, ZOU

“Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability”

Proceedings of the 2005 13th IEEE International Conference on Requirements Engineering (RE’05)

10. Ricardo BAEZA-YATES, Berthier RIBEIRO-NETO

“Modern Information Retrieval”

Addison-Wesley, 1999

11. C. J. van RIJSBERGEN

“Information Retrieval”, Second edition

12. Jussi KARLGREN

“The Basics of Information Retrieval: Statistics and Linguistics”

13. Mehmed KANTARDZIC

"Data Mining: Concepts, Models, Methods, and Algorithms"

John Wiley & Sons, 2003

14. Michael W. BERRY

"Survey of Text Mining - Clustering, Classification, and Retrieval"

Springer-Verlag, 2003

15. William B. FRAKES, Ricardo BAEZA-YATES

"Information Retrieval: Data Structures & Algorithms"

16. M. F. PORTER

"An algorithm for suffix stripping", 1980

17. Otis GOSPODNETIC, Erik HATCHER

"Lucene in action"

Manning

Ringraziamenti

Dopo tanto lavoro in cui mi sono sforzato di essere il più formale possibile, è difficile riuscire a scrivere ciò che mi viene in mente con parole naturali. In questo momento guardo furtivamente fuori della mia stanza, e mi ritornano in mente i momenti del periodo di tesi e dell'intero corso di studi.

Penso ad **Aaron** (formalmente **Prof. Visaggio**) che nonostante i suoi tanti impegni di lavoro ha cercato di seguire il mio cammino dall'inizio alla fine e ha dovuto per due o tre volte leggere e sopportare delle mie e-mail in cui gli riversavo addosso le mie crisi di panico (a cui non rispondeva: chissà cosa pensava!). Penso a **Luigi** (**Ing. Cerulo**), sempre silenzioso ma con le poche parole giuste di suggerimento al bisogno. Li ringrazio per avermi supportato e spero che in futuro possano instaurarsi legami che vadano oltre quelli professionali.

A ruota segue la persona che mi ha dato la forza di completare questo lavoro: **Anna Maria**. Anche se gli impegni di lavoro la stressano e la sua vita è sacrificata, non si lamenta mai ed anzi mi ha sempre confortato ed aiutato nei momenti bui (e non sono stati pochi!) con l'amore che solo lei è in grado di darmi: non è un caso che la senta già come mia moglie e non abbia vergogna scrivendole "**TI AMO!**".

Poi c'è **mia madre**, che opera nell'ombra non dicendomi o chiedendomi mai nulla, ma supportandomi e sopportandomi ogni giorno, soffrendo con me e per me, pregando la sera, e riempiendomi di tante importanti piccolezze. Anche **mio padre**, a modo suo, mi è stato vicino sollevandomi da impegni che avrei dovuto assolvere, e so in cuor mio che anche se non riesce a dimostrarlo tiene molto a me come alle mie sorelle…a proposito, quasi dimenticavo (scherzo!) **Antonella** e **Maria Pia**: a loro va, indistintamente, tutto il mio

affetto come fratello e come oggetto delle loro attenzioni in quest'ultimo periodo. Maria Pia mi è stata di grande aiuto dandomi consigli tecnici e lavorando con me alla raccolta dei dati, ed Antonella ha sempre usato il suo prezioso tempo per assistermi quando venivo a Benevento e sbrigare le mie commissioni. Aggiungo a loro come sorella acquisita **Fiammetta**, con cui ho avuto profonde esperienze di vita che mi hanno permesso di riflettere ed alla quale spero di essere stato sufficientemente d'aiuto negli studi e nella quotidianità, e penso anche al fidanzato **Marco**.

Può sembrare stupido, ma in questi momenti penso anche a due persone molto care, **mia nonna** e **zio Nino** che, benché non potranno essere presenti al momento della seduta, saranno felici di sapere che finalmente il loro nipote ha raggiunto una tappa importante della propria vita, tappa in cui loro hanno sempre sperato.

Un forte pensiero va alla **famiglia di Anna Maria**, che ha sempre creduto nelle mie potenzialità senza giudicarmi e mi vuole bene come se ne fossi membro da sempre.

Ora cominciano ad affollarsi nella mia testa tutti gli amici, vicini e lontani, che la vita personale ed universitaria mi ha permesso di conoscere e che mi hanno sostenuto nel periodo dei miei studi. **Giorgio** e **Pierino** (formalmente **Piero**), le persone più vere e genuine che abbia mai incontrato, con cui ho instaurato un vero rapporto di amicizia che va oltre le distanze. **Luca**, lo scherzoso coinquilino di Giorgio e Pierino, oggi mio rispettato collega di lavoro nonché stimato cognato. **Fulvio**, l'amico d'infanzia che le vicissitudini della vita hanno allontanato da me. **Leo**, il più simpatico e disponibile scansafatiche esistente. **Michele**, una persona piena di risorse umane che ha sempre reso disponibili a tutti. Le Anna Maria's Angels: **Enza**, **Luana** e l'unica **Francesca** (che come la Panda™ se non ci fosse bisognerebbe inventarla!). **Aldo**, l'amico viaggiatore che ha sempre un pensiero per me, insieme alla consorte **Teresa** che amorevolmente lo sopporta. **Antonio Lucarelli**, il geniaccio dell'università con un cuore grande grande. **Franca** e **Francesco**, i

fidanzatini che hanno e portano sempre allegria. **Antonio Russo**, il più grande pasticcione che la storia ricordi, e uno degli amici più bravi ed affezionati che ho. **Cesare**, l'amico delle superiori che mi ha instancabilmente seguito e cercato in tutti questi anni. **Savino** e **Luca**, miei amici ormai lontani fisicamente, ma sempre presenti nei miei pensieri.

Un ultimo pensiero va a tutti i colleghi di lavoro che da amici mi hanno esortato a portare a termine questo corso di studi: **Mario Sireno**, **Antonio Di Franco**, **Antonio Brescia**, **Rocco Palazzo** ed il nuovo arrivato, **Luca Signorile**.

A tutti quelli che ho nominato, ed a quelli che non ho menzionato perché buona parte dei miei neuroni non si è più rigenerata, con gli occhi lucidi rivolgo un semplice, ma sentito **GRAZIE!**