POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

USO DI ONTOLOGIE PER L'ESTRAZIONE DI INFORMAZIONI STRUTTURATE DAI WIKI

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Marco Colombetti
Correlatore: Ing. Davide Eynard

Tesi di Laurea di:
Miglierina Carlo, matricola 672254

Anno Accademico 2008-2009

Questo lavoro è dedicato innanzitutto a san Riccardo Pampuri per avermi aiutato negli esami ed essere finalmente giunto alla laurea. Inoltre lo dedico a Tolkien, per aver creato quel meraviglioso mondo della Terra di Mezzo, e di aver scritto uno dei più bei libri di sempre: Il Signore degli Anelli

Indice

Capitolo 1

Introduzione

«Avrei tanto desiderato che tutto ciò non fosse accaduto ai miei giorni!», esclamò Frodo. «Anche io», annuì Gandalf, «come d'altronde tutti quelli che vivono questi avvenimenti. Ma non tocca a noi scegliere. Tutto ciò che possiamo decidere è come disporre del tempo che ci è dato»

dal libro "Il Signore degli Anelli".

Dalla prima connessione di due computer in remoto, avvenuta nel 1969, ad oggi il Web è molto cambiato. Nel 1982 viene definito il protocollo TCP/IP e che cosa è Internet e l'anno seguente nascono i primi server dei nomi dei siti. Nel 1989 sono connessi centomila computer, nel 1991 il CERN annuncia la nascita del Word Wide Web in cui si possono condividere documenti ipermediali, nel 1992 nasce l'HTML. Un anno dopo viene creato il primo browser pensato per il Web, chiamato Mosaic. Fino a qui si parla di fase uno di Web, ovvero siti con contenuti statici. Ora siamo nella fase due, i siti non condividono semplicemente informazioni, ma permettono di fare operazioni come l'acquisto o vendita di beni, controllare il proprio conto in banca e trasferire da esso del denaro. Inoltre il numero di computer connessi alla rete è intorno ai seicento milioni. Per maggior dettagli si veda [1].

Per quanto riguarda la condivisione delle informazioni, sono nati parecchi siti che cercano di rendere più semplice tale operazione, utilizzando tecnologie semplici da usare per chiunque. In questo contesto da anni in Internet si sono ampiamente diffusi i wiki, siti Web che permettono un editing collettivo delle informazioni. Essi usano un linguaggio molto semplice, in modo che chiunque possa facilmente modificare o creare pagine wiki. Questo ha permesso una più semplice diffusione della conoscenza in vari campi, creando o dando la possibiltà di creare pagine in cui vengono condivise informazioni.

Un esempio famoso è Wikipedia: esso mette a disposizione degli utenti degli strumenti semplici per la creazione di pagine in cui condividere ciò che uno conosce. Esse vengono create a partire dai link su pagine che trattano di argomenti simili a quello di cui si vuole parlare, così chi consulterà tali

temi può anche andare su altre per approfondire alcuni concetti presenti nel documento.

Tuttavia la semplicità di modifica e creazione ha portato un grosso limite: i dati non sono strutturati, impedendo ricerche avanzate e l'uso delle informazioni contenuti da parte degli elaboratori.

Questa tesi nasce con l'obiettivo di creare uno strumento in grado di estrarre informazioni dai wiki e di organizzarli in un'ontologia. Per ottenere tale scopo è stata creata una applicazione scritta in Java, che scarica i dati da wiki e li organizza in una base di conoscenza implementata in OWL; per avere un esempio concreto su cui lavorare è stata costruita un ontologia in cui salvare dati su personaggi de "Il Signore degli Anelli".

I vantaggi che si possono ottenere sono i seguenti: l'estrazione dei dati è automatica, evitando un faticoso e lungo lavoro manuale, essi sono ben strutturati, in modo che sia possibile fare ricerche avanzate di informazioni in maniera semplice tramite query SPARQL, infine si possono applicare regole (con SWRL o tramite il supporto per l'inferenza di Jena) che permettono di inferire della nuova conoscenza, aggiungendo nuove informazioni partendo da quelle già a disposizione.

Per fare ciò è stato sfruttato un nuovo campo del web, che già sta iniziando a diffondersi in internet: il web semantico, che corrisponde alla terza fase del Web. Esso cerca di associare ai documenti anche un contenuto semantico codificato in modo da essere comprensibile anche da sistemi informatici, e che questi possano poi utilizzare la conoscenza contenuta nei documenti.

La presente tesi è organizzata nel seguente modo:

- nel secondo capitolo si espongono le basi teoriche del progetto, insieme a una breve descrizione delle principali tecnologie utilizzate
- nel terzo capitolo si illustrano le scelte di progetto effettuate e si dà una descrizione dell'architettura dell'applicazione e dell'ontologia sottostante
- nel quarto capitolo si mostrano alcuni esempi dei risultati ottenuti
- nel quinto capitolo si espongono le conclusioni sul lavoro svolto e si danno alcuni suggerimenti sui possibili sviluppi futuri.

Capitolo 2

Basi teoriche e tecnologie

«Che peccato che Bilbo non abbia trafitto con la sua spada quella vile e ignobile creatura quando ne ebbe l'occasione!». «Peccato? Ma fu la Pietà a fermargli la mano. Pietà e Misericordia: egli non volle colpire senza necessità. (...)». «Mi dispiace», disse Frodo; « ma sono terrorizzato e non ho alcuna Pietà per Gollum».(...) «Merita la morte». «Se la merita! E come! Molti tra i vivi meritano la morte. E parecchi che sono morti avrebbero meritato la vita. Sei forse tu in grado di dargliela? E allora non essere troppo generoso nel distribuire la morte nei tuoi giudizi: sappi che nemmeno i più saggi possono vedere tutte le conseguenze. Ho poca speranza che Gollum riesca ad essere curato ed a guarire prima di morire.Ma c'è una possibilità. Egli è legato al destino dell'anello. Il cuore mi dice che prima della fine di questa storia l'aspetta un'ultima parte da recitare; e quando l'ora giungerà, la pietà di Bilbo potrebbe cambiare il corso di molti destini, e soprattutto del tuo.»

dal libro "Il Signore degli Anelli".

2.1 Il web semantico e le ontologie

Le ontologie nascono all'interno dell'area dell'informatica che si occupa del web semantico, che è stato definito come la prossima evoluzione del web. La novità di questo progetto sta nel fatto di poter assegnare un significato ai dati presenti sulla rete, e rappresentare tali conoscenze in modo che il computer possa leggerle e utilizzarle.

Dare significato ai dati vuol dire collegare il linguaggio alla realtà, cioè fare sì che a ogni termine sia collegato un oggetto reale. Tale collegamento è mediato dai concetti legati a tale termine. Ad esempio alla parola cane è legato il concetto di tale animale, con il quale si può collegare la parola cane a un cane reale. Questo è definito come triangolo aristotelico termine-concetto-oggetto o linguaggio-mente-realtà.

Oltre a un modo per rappresentare la conoscenza servono anche delle procedure di ragionamento, in questo ambito si usa il ragionamento deduttivo. Esso permette, partendo da delle premesse che esprimono delle informazioni

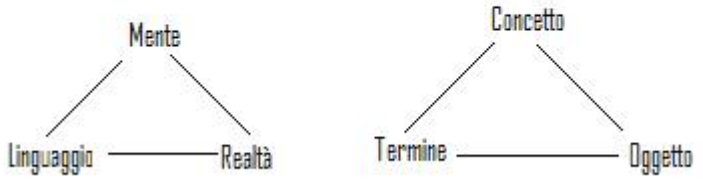

Figura 2.1: Triangoli Aristotelici Linguaggio-Mente-Realtà e Termine-Concetto-Oggetto

già conosciute, di ottenere nuove informazioni conservando l'eventuale verità delle premesse. Per fare questo procedimento si usa la logica matematica. Essa è il settore della matematica che studia i sistemi formali, che sono strumenti per definire in maniera rigorosa e completa un insieme di assiomi (principi che vengono ritenuti veri perché evidenti o perché assunti come punti di partenza per un particolare sistema) usati per la dimostrazione di teoremi. In particolare la logica si occupa della codificazione dei concetti intuitivi della dimostrazione e di computazione come parte dei fondamenti della matematica. Per la rappresentazione e per il processo di ragionamento si è fatto uso di una *description logic* (DL) che è un sottoinsieme della logica del primo ordine (*Frist Order Logic*, FOL).

La logica del primo ordine o teoria del primo ordine è un particolare sistema formale, cioè una teoria in cui è possibile esprimere enunciati e dedurre le loro conseguenze logiche in modo del tutto formale e meccanico. Esso è un linguaggio che gestisce meccanicamente enunciati, usando connettivi logici (cioè operatori logici come la disgiunzione $\vee$, la congiunzione $\wedge$ o la negazione $\neg$) quantificatori (per ogni $\forall$ ed esiste $\exists$) e relazioni (collegamento che sussiste tra un determinato numero di oggetti appartenenti a determinati insiemi). L'espressione "del primo ordine" indica che c'è un insieme di riferimento e i quantificatori possano riguardare solo gli elementi di tale insieme e non i sottoinsiemi.

Non si usa FOL (nonostante essa possa esprimere conoscenze molto articolate) perché è solo semi decidibile: cioè quando la conclusione è deducibile dalle premesse, il processo di ragionamento termina sempre dopo un numero finito di passi producendo una prova; in caso contrario potrebbe non terminare affatto.

Le DL sono comunque abbastanza espressive, sono decidibili e quindi dopo un numero finito di passi danno risposta. Dati questi vantaggi, bisogna trovare una DL che abbia una complessità computazionale accettabile: questo significa che il processo deduttivo deve terminare in un tempo

accettabile.

Per effettuare il processo di ragionamento deduttivo si usa un reasoner, uno strumento automatico che applica le DL e crea un nuovo modello della conoscenza chiamato "modello inferito" o "inferred model". In tale modello vale tutto quello che si è esplicitamente detto, in più si inseriscono le conclusioni tratte dal reasoner. Il modello che contiene solo ciò che è stato espresso esplicitamente si chiama "modello asserito" o "asserted model". Bisogna notare inoltre che un'ontologia non identifica un solo modello, ma identifica tutti i modelli possibili in cui quanto è espresso in essa è vero.

Tra i vari sistemi di logiche, e stato scelto un linguaggio basato su logica SHOIN(D_n), dove le varie lettere rappresentano le proprietà che tale logica possiede. Esse sono (i concetti di classe, proprietà e individui verranno spiegati più avanti):

- S: assiomi d'inclusione e di equivalenza di classi; classi denominate, classe universale ($\top$) e classe vuota ($\bot$), classi anonime formate con i costruttori di intersezione, unione, operatore universale quantificato, operatore esistenziale quantificato o non quantificato e negazione; assiomi di equivalenza o disequivalenza tra individui e appartenenza di un individuo a una classe, dichiarazione di transività delle proprietà
- H: assiomi di inclusione e di equivalenza tra proprietà
- O: nominali
- I: proprietà inversa
- N: restrizioni di cardinalità non qualificate
- D_n: attributi con valori in domini di dati

Le classi in una base di conoscenza rappresentano i concetti, mentre l'estensione (o interpretazione) di un individuo è un oggetto reale. L'insieme degli individui rappresenta l'estensione della classe.

La classe universale è la classe che contiene tutti gli individui, mentre la classe vuota (come suggerisce il nome) è quella che non contiene individui.

Si dice classe denominata (o named class) una classe che è identificata da un nome, mentre una classe anonima è invece descritta da un'espressione fatta con i costruttori di classi. Tali costruttori sono: l'unione, l'intersezione, la negazione, il costruttore esistenziale (qualificato o non qualificato) e il costruttore universale qualificato.

Due classi sono equivalenti (A$\equiv$B) quando le estensioni delle due classi si equivalgono in qualunque universo, cioè in qualunque interpretazione della base di conoscenza. Una classe è inclusa in un altra (A$\sqsubseteq$B) quando in qualsiasi interpretazione dell'ontologia l'insieme denotato da A è un sottoinsieme da quello denotato da B.

L'operatore unione di due classi A e B ($A \sqcup B$) costruisce la classe C che contiene tutti gli individui che appartengono ad almeno una delle due classi. Ad esempio se A rappresenta l'insieme di tutti gli uomini e B l'insieme delle donne, la classe C rappresenta tutti gli esseri umani. L'intersezione ($A \sqcap B$) invece crea la classe C che contiene gli individui che appartengono ad entrambe le classi. Se A è la classe dei genitori, B quella che contiene le persone di genere maschile allora la classe C rappresenta i padri. La classe ottenuta con la negazione di una classe A ($\neg A$) rappresenta la classe che contiene tutti gli individui della classe universale meno quelli che appartengono alla classe A.

Il costruttore esistenziale ($\exists R$ dove R è una proprietà) crea le classi degli individui che hanno la proprietà R, ad esempio la classe dei genitori si può costruire $\exists$isParentOf. Se è qualificata ($\exists R.C$ dove C è una classe), la classe è ristretta agli individui che sono in relazione tramite R con gli individui appartenenti alla classe C; considerando R come la relazione essere genitore di e C la classe delle donne, l'espressione $\exists R.C$ denota l'insieme di chi è padre di sole femmine. L'operatore quantificatore universale qualificato ($\forall R.C$) crea la classe degli individui che se hanno la proprietà R, sono in relazione con gli individui della classe C, se R e C sono le stesse dell'esempio di prima, la classe formata denota l'insieme che contiene tutti gli individui che o non sono padri o se lo sono, hanno solo figlie femmine.

La restrizione di cardinalità funziona similmente come il costruttore esistenziale, ma aggiunge dei vincoli sul numero di individui con cui gli elementi dell'insieme che si vuole costruire hanno una relazione tramite la proprietà R. In particolare (se n è un numero intero) $\leq$nR significa che la classe costruita contiene individui che sono in relazione con al massimo n individui tramite R; $\geq$nR gli individui sono in relazione con almeno n individui; infine =nR costruisce la classe che contiene tutti gli individui che sono in relazione con esattamente n individui. Ad esempio $\geq$3genitoreDi rappresenta la classe delle persone che hanno almeno tre figli.

I nominali sono quelle classi che contengono un solo individuo, sono della forma (se a rappresenta un generico individuo) {a}. Le classi si possono costruire facendo l'unione di più nominali. Tale tipo di costruttore di classi si chiama anche *one-of*: infatti, una variabile che appartiene a una classe così formata può assumere solo uno dei valori elencati.

Le proprietà sono una relazione binaria tra due individui e sono dette Object Property; esiste anche un tipo di proprietà che mette in relazione un individuo con un tipo di dato (un intero, una stringa o altro) e in questo caso si chiama Data Property.

Per un approfondimento su questi temi si veda [2].

2.2 Tecnologie specifiche

2.2.1 OWL

Le ontologie vengono rappresentate con Web Ontology Language (OWL), in particolare per questo lavoro è stata usata la versione 1.0. Un'ontologia OWL viene rappresentata in memoria sotto forma di grafo, e può essere serializzata in vari formati tra cui un'estensione di RDF. Esso viene esteso tramite costrutti aggiuntivi per modellare realtà più complesse, inoltre si vincola l'uso dei costrutti RDF in modo da rimanere nel primo ordine e si fa in modo che sia proposizionalmente chiuso.

2.2.2 RDF

Il *Resource Description Framework* (RDF) è lo strumento base per la codifica, lo scambio e il riutilizzo di metadati strutturati e consente l'interoperabilità tra applicazioni che si scambiano informazioni sul Web. Esso è un modello e non fornisce linguaggio, cioè è un modo di rappresentare metadati.

È costituito da due componenti:

- RDF Model and Syntax: espone la struttura del modello RDF, e descrive una possibile sintassi
- RDF Schema: espone la sintassi per definire schemi e vocabolari per i metadati

Tutto quello che viene descritto da RDF è chiamato risorsa; ogni risorsa viene identificata da un URI (Uniform Resource Identifier).

RDF può essere rappresentato in vari modi, ad esempio: con un grafico, N3 e RDF/XML. Nel grafico le risorse sono visualizzate come nodi e i predicati come archi. Nel N3 si rappresenta il modello RDF con le triple soggetto-predicato-oggetto, cioè un insieme di tre elementi, di cui il primo rappresenta il soggetto, il secondo il predicato o proprietà con cui il soggetto è legato al terzo elemento, l'oggetto. Nell'ultimo modo si serializza il tutto in un file XML. Per una descrizione completa si veda anche [3].

2.2.3 XML

L'*eXtensible Markup Language* (XML) è un linguaggio di annotazione (markup) che permette di creare creare gruppi di marcatori (tag set) personalizzati (ad esempio XHTML o MathML). É anche un formato standard usato per lo scambio di dati, metalinguaggio per creare documenti arricchiti da informazioni addizionali e un supporto per la costruzione di formati specifici per gli usi più disparati. Per definire in maniera univoca i tag, in modo che si possano unire più documenti XML che usano gli stessi tag per indicare concetti differenti senza fare confusione, si usano i *namespace* che

sono anch'essi identificati da URI. (Per maggior informazioni si veda [4] e [5]).

Visivamente è simile a HTML, ma ha regole sintattiche diverse, principalmente l'innestamento dei tag deve essere corretto, i tag devono essere sempre chiusi ed è ammesso un solo elemento radice. Oltre alle regole sintattiche XML consente di effettuare sui propri documenti anche un controllo di tipo semantico (cioè una verifica della validità delle tag rispetto a quello che il documento vuole rappresentare). Per consentire questo tipo di controllo si usano la *Document Type Definition* (DTD) o la *XML Schema Definition* (XSD).

Qui sotto è riportato esempio di documento XML:

```
<?xml version=‘‘1.0’’ encoding=‘‘ISO-8859-1’’?>
<utenti>
       <utente>
               <nome>Luca</nome>
               <cognome>Ruggero</cognome>
               <indirizzo>Milano</indirizzo>
        </utente>
        <utente>
               <nome>Max</nome>
               <cognome>Rossi</cognome>
               <indirizzo>Roma</indirizzo>
        </utente>
</utenti>
```

La prima riga indica la versione di XML in uso e specifica la codifica ISO per la corretta interpretazione dei dati, il tag utenti indica l'elemento radice.

Il DTD e il XSD sono documenti in cui si descrivono i tag e la struttura degli elementi. Il primo è semplice e fa parte dello standard ma di contro è poco espressivo, crea difficoltà all'inserimento dei namespace, non consente di imporre vincoli al contenuto testuale e, soprattutto, agli attributi, non permette di creare testi tipizzati e non è scritto in XML. Invece XSD è molto espressivo, ha un supporto per l'inserimento dei namespace, permette la tipizzazione dei dati ed è esspresso in XML. Come svantaggio XSD è più complesso e prolisso. Nell'esempio, il DTD (o il XSD) di tale documento impone che l'elemento radice si chiami utenti, all'interno ci devono essere uno o più elementi chiamati utente. Questi ultimi devono contenere gli elementi nome, cognome e indirizzo. Il documento DTD in particolare sarebbe:

```
<!ELEMENT utenti (utente+)>
<!ELEMENT utente (nome,cognome,indirizzo)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT indirizzo (#PCDATA)>
```

dove #PCDATA indica che l'elemendo contiene una stringa. Il simbolo più indica che tale campo all'interno dell'elemento utenti deve essercene almeno uno, ma può essere ripetuto più volte.

Il documeento XSD conterrebbe le righe:

```
<xsd:element name=‘‘utenti’’>
<xsd:complexType>
<xsd:element name=‘‘utente’’ maxOccurs=‘‘unbounded’’>
<xsd:sequence>
<xsd:element name= ‘‘nome’’ type= ‘‘xsd:string’’/>
<xsd:element name= ‘‘cognome’’ type= ‘‘xsd:string’’/>
<xsd:element name= ‘‘indirizzo’’ type= ‘‘xsd:string’’/>
</xsd:sequence>
</xsd:element>
</xsd:complexType>
</xsd:element>
```

il campo maxOccurs indica quante volte al massimo può comparire l'elemento in cui tale attributo compare, il valore unbounded indica che ci possono essere infinite ripetizioni. Per maggiori informazioni si veda [6] e [7].

Un documento XML viene controllato da un parser, strumento che controlla la sintassi. Se lo confronta anche con il documento DTD o XSD è detto parser validante. Esso inoltre crea una rappresentazione in grado di essere elaborata dalle varie applicazioni che lavorano su tale documento.

2.2.4 SPARQL: il linguaggio per le query

SPARQL Protocol and RDF Query Language (o semplicemente SPARQL) è un linguaggio query per RDF, usato anche su ontologie; esse sono sintatticamente molto simili a quelle fatte in SQL per i database. Le query SPARQL hanno la forma:

```
PREFIX pref:<namespace>
SELECT ?x ?y
WHERE { ?x pref:prop ?y}
```

dove ?x e ?y sono delle variabili, pref è il prefisso (che corrisponde al namespace specificato dopo il comando PREFIX) e prop è una proprietà qualsiasi. All'interno di query SPARQL ci possono essere più triple, che vanno a rappresentare un modello di grafo che deve essere soddisfatto dai dati presenti nella base di conoscenza. Inoltre esistono altri comandi:

- DISTINCT: da inserire dopo SELECT, serve per evitare le ripetizioni nei risultati della query.

- GROUP BY ?x: da inserire dopo il blocco WHERE, serve a raggruppare i risultati secondo la variabile ?x
- ORDER BY ?x: come GROUP BY, in più ordina i risultati rispetto alla variabile ?x
- FILTER da inserire all'interno del blocco WHERE, serve per filtrare i risultati ottenuti secondo particolari vincoli.

2.2.5 Tecnologie per l'inferenza

Per fare inferenza sull'ontologia si possono usare varie tecnologie, tra le quali: SWRL e Jena. Queste servono per creare regole, ma esse non sono l'unico modo per fare inferenza, infatti esistono, ad esempio, la sussunzione (cioè il meccanismo per verificare che una classe sia una sottoclasse di un'altra) e la chiusura di proprietà, cioè trovare un estensione di un proprietà che abbia certe caratteristiche, come ad esempio che sia transitiva.

Il *Semantic Web Rule Language* o SWRL è una versione compatibile a OWL di Datalog, un linguaggio a regole. Le regole SWRL permettono di creare regole usate per fare inferenza sulla base di conoscenza, cioè di fare un processo di ragionamento deduttivo per ottenere nuova conoscenza partendo da quello che già si sa.

Le regole costruite tramite Jena sono simili a quelle scritte in SWRL. In entrambi i casi le regole hanno lo stesso formato: sono un'implicazione in cui l'antecedente è costituito da una serie di proprietà unite da congiunzioni logiche ("and") e quando l'antecedente è vero (cioè se tutte le proprietà sono verificate) viene inserita nell'ontologia la proprietà contenuta nel conseguente. Se P1, P2 e P3 sono delle propietà, lotr è il prefisso che denota il namespace dell'ontologia, le regole hanno il formato (secondo la sintassi di Jena):

```
[rule1:  (?a lotr:P1 ?b) (?b lotr:P2 ?c) -> (?a lotr:P3 ?c)]
```

dove ?a,?b e ?c sono delle variabili che denotano individui.

2.3 I wiki

Le informazioni presenti nell'ontologia sono state estratte da una categoria particolare di siti: i wiki. Essi sono un siti web o documenti ipertestuali che vengono creati e aggiornati dai loro utilizzatori usando un linguaggio particolare, che varia a seconda del software utilizzato per creare il singolo wiki. Tale tecnologia si poggia su un software che crea le pagine web partendo dal codice scritto dagli utenti. Una caratteristica distintiva dei wiki è la facilità con cui le pagine possono essere create e aggiornate.

I Wiki sono un mezzo completamente ipertestuale, con una struttura di navigazione non lineare. Tipicamente ogni pagina contiene un gran numero di link ad altre pagine; nei wiki di dimensioni notevoli esiste comunque una navigazione gerarchica, ma non deve essere necessariamente usata. I collegamenti (link) vengono creati usando una sintassi particolare.

La maggior parte dei wiki usa il modello CamelCase per la nomenclatura dei link, che viene prodotta mettendo in maiuscolo la lettera iniziale di ogni parola contenuta in una frase ed eliminando gli spazi. CamelCase oltre a facilitare i collegamenti, induce la scrittura dei link ad una forma che devia dallo spelling standard.

Solitamente in un wiki le nuove pagine sono create semplicemente inserendo il link appropriato in una pagina che tratta un argomento correlato. Se il link non esiste, esso viene evidenziato come link rotto (broken link). Cercando di seguire quel link viene aperta una finestra di modifica, che permette all'utente di inserire il testo della nuova pagina. Questo meccanismo assicura che le pagine cosiddette "orfane" (cioè che non hanno link che puntano ad esse) siano create raramente, mantenendo un alto livello di connessione.

Generalmente, non esiste una verifica preventiva sulle modifiche, e la maggior parte dei wiki è aperta a tutti gli utenti. Per evitare che malintenzionati inseriscano informazioni sbagliate, cancellino quelle esistenti o procurino danni di altro tipo, i wiki normalmente hanno una cronologia delle modifiche in modo che sia possibile tornare a una versione precedente nel caso vengano fatte modifiche indesiderate.

Essi consentono di creare pagine con dati semi strutturati, e spesso in pagine wiki sono presenti template e infobox, che semplificano l'inserimento delle informazioni. In alcuni casi esse semplificano anche l'operazione di estrazione dei dati, motivo per cui si è fatto uso di tali siti. Per maggiori informazioni si veda anche [8].

Questo è un esempio di codice wiki, preso dalla pagina di wikipedia che parla di come si fanno i link interni.

TOCright Aiuto I "'wikilink"' sono link, ossia [[Collegamento ipertestuale/collegamenti ipertestuali]], che per agevolare la consultazione di [[Wikipedia]] collegano le une con le altre le diverse pagine dell'enciclopedia (sotto il dominio "it.wikipedia.org"). Mentre i collegamenti a pagine dello stesso progetto sono detti wikilink (spesso abbreviato in "'wlink"'), i collegamenti diretti alle corrispondenti pagine delle Wikipedie in altre lingue sono detti [["Aiuto:interwiki/interwiki"]] e vanno nel "[[portlet]]" "Altre lingue" (sta in basso a sinistra, sotto gli strumenti). Una pagina che non contiene wikilink è detta [[Aiuto:Glossario#Pagina_senza_uscita/senza uscita]]; una a cui non puntano wikilink [[Aiuto:Pagina orfana/orfana]].

*Anche se in linea teorica l'intero progetto di Wikipedia è assimilabile ad un [[ipertesto]], l'attuale "policy" in materia, non scritta, ma desumibile da altre pagine, non permette di creare wikilink in maniera indiscriminata; ma indica assieme al [[Wikipedia:buon senso/buon senso]] le seguenti norme/suggerimenti: * dovrebbero essere trasformati in wikilink "'solo i termini più significativi per l'argomento"' trattato nella pagina o, più in generale, i termini che presentano interesse o difficoltà (parole straniere, tecnicismi, locuzioni latine...). * un termine può presentarsi diverse volte in una voce, ma il wikilink*

non va inserito ad ogni ricorrenza, bensì solo una volta, alla prima occasione o quando diventa il tema centrale del discorso; sono ammesse ovviamente eccezioni nelle pagine più lunghe o qualora diventasse scomodo per l'utente risalire al wikilink scorrendo la pagina. vedi anche|Aiuto:Manuale di stile!manuale di stile

== Istruzioni ==

Per ottenere un wikilink, è sufficiente inserire la parola fra <u>doppie</u> "parentesi quadre" "' <nowiki>[[ ]]</nowiki> "' <ref>ATTENZIONE le singole parentesi quadre ("'[ ]"') sono utilizzate per i [[Wikipedia:collegamenti esterni/collegamenti esterni]]</ref> <small>(ALT + GR + è; ALT + GR + +)</small> in questo modo: "' <nowiki>[[</nowiki>"'Dante Alighieri "' <nowiki>]]</nowiki> "' è l'autore della "' <nowiki>[[</nowiki> "' Divina Commedia "' <nowiki>]] </nowiki>"' . se la parola corrisponderà al titolo esatto della pagina si ottiene: :[[Dante Alighieri]] è l'autore della [[Divina Commedia]]. altrimenti verrà visualizzato un [[Wikipedia:link rosso/link rosso]] :[[Dante Arighieri]] è l'autore della [[Divina COMMEDIA]] come se la pagina non esistesse ancora. Poiché il click su un link rosso produce l'invito a creare la nuova pagina con quel titolo, talvolta si inseriscono volutamente link rossi come stimolo al completamento dell'enciclopedia.

Se invece il titolo della pagina a cui si vuol fare riferimento è diverso dalla parola che si vuol far comparire nel testo, bisogna procedere nel seguente modo: <tt>[["Titolo della pagina""'!"'testo]]</tt> Cioè: fra le parentesi si scrive prima il "titolo esatto" della pagina e poi il "testo" che si vuole visualizzare come link, separati da una barra verticale ("pipe"): "|" <small> (SHIFT + \)</small>. Ad esempio :[[" Thomas Mann""'!"' Thomas]] e [[" Heinrich Mann " "'!"' Mann]] diventano rispettivamente: :[[Thomas Mann|Thomas]] e [[Heinrich Mann|Mann]]

==Wikilink diretti a sezioni e sottosezioni==

I wikilink possono anche puntare direttamente a [[aiuto:sezioni|sezioni]] o sottosezioni, identificate dai titoli messi fra <tt><nowiki>== ==</nowiki></tt><ref>Il link è valido per qualsiasi sezione e sottosezione, cioè anche per i titoli individuati da tre o più uguali (=): è sufficiente che il nome della sezione o della sottosezione indicato corrisponda al titolo di una sezione o sottosezione effettivamente presente nella pagina oggetto del link.</ref>; la sintassi è la seguente:

<tt><nowiki>[[</nowiki>"Titolo pagina"# "Titolo (sotto)sezione"| Testo]]</tt>

Il '#' indica che il testo <u>dopo</u> è proprio il titolo della sezione voluta. Se questo dovesse risultare modificato, o comunque non corrispondente, il wikilink punterà non più direttamente alla sezione, alla pagina come fosse un wikilink qualsiasi.

La sequenza == nomesez == serve per creare una nuova sezione del documento dal titolo *nomesez*. In Figura 2.2 è mostrata la pagina ottenuta con il codice riportato sopra.

Figura 2.2: Una pagina di Wikipedia

Capitolo 3

Progetto e implementazione

Sam:«È come nelle grandi storie, padron Frodo, quelle che contano davvero, erano piene di oscurità e pericolo, e a volte non volevi sapere il finale, perché come poteva esserci un finale allegro, come poteva il mondo tornare com'era dopo che erano successe tante cose brutte, ma alla fine è solo una cosa passeggera, quest'ombra, anche l'oscurità deve passare, arriverà un nuovo giorno, e quando il sole splenderà sarà ancora più luminoso, quelle erano le storie che ti restavano dentro, anche se eri troppo piccolo per capire il perché, ma credo, padron Frodo, di capire ora, adesso so, la persone di quelle storie avevano molte occasioni di tornare indietro e non l'hanno fatto... andavano avanti, perché loro erano aggrappati a qualcosa.»

Frodo:«Noi a cosa siamo aggrappati Sam?»

Sam:«C'è del buono in questo mondo, padron Frodo... è giusto combattere per questo!»

dal film "Le Due Torri".

Per lo sviluppo di questa applicazione è stato necessario fare alcune scelte a livello di progettazione e implementazione. Il presente capitolo descrive tali scelte. Nella prima sezione si spiega quale linguaggio di programmazione e quali tecnologie sono state utilizzate per implementazione. Nella seconda viene descritta l'architettura dell'applicazione e nella terza è descritta l'architettura dell'ontologia.

3.1 Scelte progettuali

3.1.1 Linguaggio di programmazzione

Il linguaggio di programmazione che si è deciso di utilizzare per lo sviluppo dell'applicazione è Java.

Questo linguaggio è stato scelto perché ha vari vantaggi: è orientato agli oggetti, è indipendente dalla piattaforma, contiene strumenti e librerie per il networking ed è eseguito in modo sicuro. La programmazione orientata agli oggetti aiuta perché permettere di dividere il problema in parti più piccole e più semplici; inoltre rende semplice la modifica di alcune funzioni o parti del

codice che svolgono una particolare operazione. Infatti si può cambiare un metodo o un intera classe e, a patto di non modificare la firma dei metodi e dei attributi, lasciare invariato il resto del codice.

Java viene eseguito in maniera sicura in quanto ogni applicazione viene avviata in una sandbox, cioè impedisce che del codice malevolo o dal comportamento indesiderato possa avere ripercussioni negative su tutto il sistema.

L'indipendenza della piattaforma significa che l'esecuzione dell'applicazione deve avere un comportamento simile su hardware o sistemi operativi diversi. Cioè si può eseguire (in teoria) ovunque. Questo è ottenuto eseguendo il codice su una macchina virtuale (Java Virtual Machine) che fa da interfaccia tra il programma scritto in Java e l'elaboratore. Tale macchina virtuale traduce le istruzioni Java in istruzioni da passare all'elaboratore in cui si trova: in questo modo essa non solo consente l'esecuzione di codice Java su macchine e sistemi operativi differenti, ma fa anche da sandbox impedendo l'esecuzione di codice malevolo sulla macchina reale.

3.1.2 Ambiente di sviluppo e librerie

Come piattaforma di sviluppo si è deciso di usare di Eclipse, un comodo programma open source. Comodo sia dal punto di vista dell'interfaccia utente in quanto usa una pagina per ogni classe, permette una facile importazione di nuove librerie o di classi già esistenti sia dal punto di vista funzionale poiché possiede una funzione di auto completamento che suggerisce i metodi disponibili quando si usa un'istanza di una classe.

Oltre alle librerie già presenti, è stata importata la libreria Jena (versione 2.5.7) che permette di lavorare con le ontologie. Tale libreria mette a disposizione classi per la costruzione dell'ontologia, il suo popolamento, l'esecuzione di query e il reasoning attraverso le Jena rules. Queste ultime permettono di fare reasoning, dando la possibilità di creare regole per ottenere nuove informazioni dalla base di conoscenza. Inoltre è stata importata la libreria jdom-1.0, che permette l'elaborazione di documenti XML. In Figura 3.1 è rappresentata finestra di Eclipse.

3.1.3 Editor di ontologie

Per la visualizzazione e modifica dell'ontologia abbiamo scelto di utilizzare Protégé. Esso presenta varie schede in cui mostra le classi, le proprietà e gli individui. Inoltre mette a disposizione strumenti per fare query usando il linguaggio SPARQL e scrivere regole SWRL. Queste ultime due caratteristiche tuttavia non sono state utilizzate ma si sono sfruttate le funzioni di Jena. Questo perché Protégé creava dei problemi su queste funzionalità; inoltre in questo modo si utilizza unicamente l'applicazione Java. In Figura 3.2 è rappresentata una finestra di Protégé.

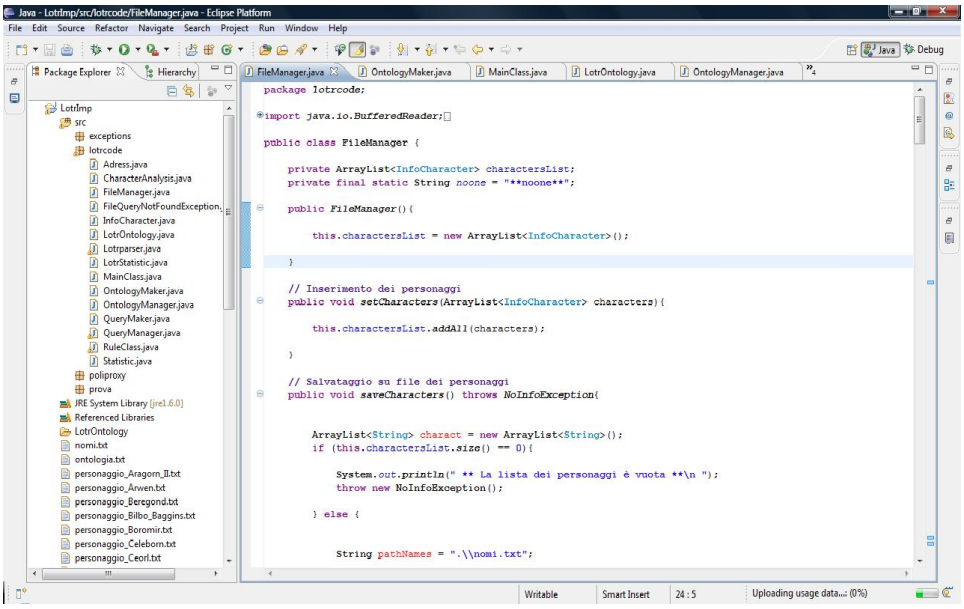

Figura 3.1: Eclipse

3.1.4 Fonte dei dati

Le informazioni sono state scaricate sfruttando gli infobox presenti nei siti. Essi sono dei template creati per facilitare la creazione di tabelle riassuntive sulle informazioni presenti nella pagina. Ogni campo ha un nome, uguali nelle varie pagine del sito che trattano dello stesso argomento (ad esempio nelle pagine dei personaggi di un libro o un film). In Figura 3.3 è mostrato un esempio di infobox.Per maggiori informazioni si veda [9].

Per ottenere i dati desiderati, sono stati usati i siti:

- http://tolkiengateway.net
- http://lotr.wikia.com

La scelta di questi due siti è stata fatta perché essi contengono template su un numero abbastanza grande di personaggi e con sufficienti informazioni. Altri siti trovati o avevano informazioni su un numero ridotto sui personaggi, o informazioni troppo lacunose. Altri non avevano nè template nè infobox.

3.2 Architettura dell'applicazione

3.2.1 L'applicazione per il recupero dei dati

L'applicazione è composta da diverse classi per gestire i vari aspetti del progetto (download dei dati, costruzione e popolamento ontologia e altri).

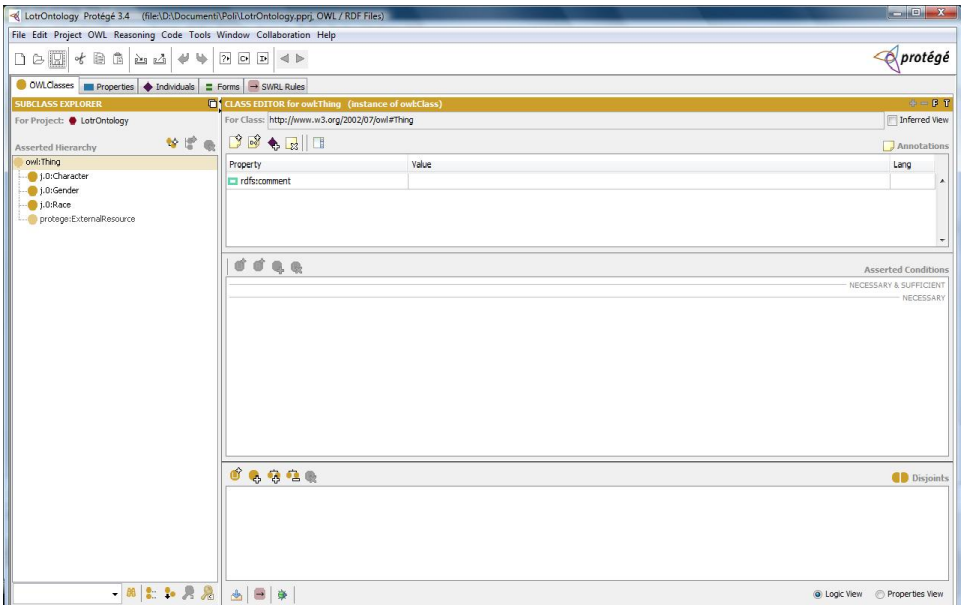

Figura 3.2: Protégé

Frodo Baggins	
Biographical information	
Other names	Mr. Underhill
Titles	Bearer of the One Ring
Date of birth	22 September, TA 2968
Date of death	Unknown (Last sighting 29 September, TA 3021)
Weapon	Sting, & Barrow-blade
Physical description	
Race	Hobbit
Culture	Shire-hobbit
Gender	Male

Figura 3.3: Esempio di infobox

Una prima piccola classe è stata creata per la gestione del proxy: infatti se ci si connette alla rete del politecnico di Milano è necessario usare un proxy imposto dall'università stessa, ed anche altre reti potrebbero imporre di usare un proxy particolare. Essa permette di scegliere quale proxy utilizzare, tra quello di default, quello del Politecnico di Milano o uno impostato ex novo.

Per poter scaricare le informazioni e analizzarle sono state create le classi:

- **Adress**: costruisce i vari indirizzi URL delle pagine web dei personaggi

analizzati

- **LotrParser**: scarica le pagine dai siti
- **CharacterAnalysis**: estrae le informazioni sui personaggi
- **InfoCharacter**: memorizza tali informazioni

In particolare la classe **LotrParser** mette a disposizione due funzioni: **get InformationXml** e **getInformationHtml**: esse prendono come parametri d'ingresso l'indirizzo della pagina web sotto forma di stringa e permettono, rispettivamente, di ricavare la stringa con il codice in XML e in HTML; per effettuare la conversione in HTML è stato utilizzato il parser messo a disposizione dalle librerie Java, mentre per XML si è fatto uso della libreria jdom-1.0 che mette a disposizione un parser XML tramite la classe **SAXBuilder**. Non si è fatto uso di un unica funzione poiché un sito rispetta la sintassi XML ed è quindi possibile usare un parser XML, mentre l'altro sito utilizza HTML, quindi non si può usare lo stesso metodo. Per ripulire i dati in ingresso è stato sviluppato il metodo **clearString** che sostituisce gli spazi con l'underscore, e se lo spazio è il primo carattere esso viene eliminato; questo viene fatto perché nei namespace delle varie entità non sono ammessi gli spazi. Questo metodo, inoltre toglie o sostituisce caratteri che potrebbero creare problemi quando si crea l'ontologia, in quanto potrebbero essere caratteri che XML riconosce come parte del codice (come i simboli $<$ o $>$).

La classe **Adress** viene utilizzata per la creazione degli indirizzi tramite i metodi **getAdress** e **getUrl**, ai quali vengono passati i nomi dei personaggi de "Il signore degli anelli". Questi metodi costruiscono l'URL della pagina wiki corrispondente a un personaggio partendo dal nome ed aggiungendo un prefisso e un suffisso che sono comuni a tutti. Naturalmente, essendo gli URL dei due siti differenti, le funzioni di trasformazione utilizzate sono una per sito.

Nella classe **CharacterAnalysis** il metodo **getCharacterInfo** usa i due metodi della classe **LotrParser getInformationXml** e **getInformationHtml** per ottenere le stringhe con dentro il codice XML o HTML della pagina corrispondente al personaggio che di volta in volta analizza. Dopo l'analisi di un singolo personaggio è stato necessario introdurre una sleep della durata di due secondi e mezzo poiché i server dei due siti chiudono la connessione se ricevono troppe richieste in sequenza.

Inoltre è stata creata la classe **FileManager** per poter salvare le informazioni su file .txt, uno per ogni personaggio, in modo che non sia necessario riscaricare le informazioni da Internet ogni volta; così facendo, dopo il primo utilizzo è possibile usare l'applicazione anche off-line. Un vantaggio collaterale di questo strumento è una maggiore velocità dell programma, in quanto scaricare le informazioni dai siti richiede parecchio tempo.

Per la creazione e popolamento dell'ontologia sono state create tre classi: **OntologyMaker**, **OntologyManager** e **LotrOntology**. La prima, che contiene il metodo main, crea l'ontologia, cioé crea le classi, le object property e le data property che esprimono le varie caratteristiche dei personaggi. Essa deve essere eseguita una sola volta per costruire l'ontologia ed è per questo motivo che ha il metodo main e non è lanciata dalla **MainClass**. La classe **LotrOntology** popola l'ontologia e aggiunge le proprietà tra i vari individui, appoggiandosi alla classe **OntologyManager**. Quest'ultima infatti è usata come interfaccia tra l'ontologia e l'applicazione Java, usando le funzioni messe a disposizione da Jena.

Per fare ciò l'applicazione prima inserisce tutti gli individui di cui si hanno i template nella classe character (per la descrizione dell'ontologia si veda la Sezione 3.2.2), quindi verifica se i personaggi hanno figli, genitori e coniuge. Se esso ha tali parenti e questi non sono ancora inseriti, vengono creati gli individui corrispondenti ad essi. Quindi vengono create le proprietà a seconda dei dati disponibili (come ad esempio i soprannomi o la razza).

Questo è il ciclo usato per inserire i personaggi di cui si hanno i template:

```
for (int i=0; i<size; i++) {
      String name = this.characters.get(i).getName();
      this.ontmanager.insertIdividual(nsOntology + name,
            nsOntology + ‘‘Character’’);
      charactersns.add(nsOntology +
           this.characters.get(i).getName());
}
```

Dove **size** rappresenta il numero di personaggi e **charactersns** è un ArrayList di stringhe in cui si memorizzano i namespace. La variabile name è utilizzata per memorizzare il nome del personaggio, ottenuto dalla funzione **getName**. **Characters** è un ArrayList che contiene una sequenza di classi di tipo **InfoCharacter**, nsOntology è una costante in cui è memorizzato il namespace dell'ontologia (D:/Documenti/Poli/LotrOntology.owl#). Inoltre dopo l'inserimento di questi individui, si memorizzano i genitori, i figli e il coniuge in tre ArrayList, uno per tipo di informazione. Queste liste vengono usate per inserire i personaggi che non hanno template ma che sono imparentati con i personaggi già inseriti.

Per memorizzare questi altri individui si è fatto uso del seguente codice, in cui la variabile **found** serve per indicare se il personaggio che è genitore, figlio o coniuge di qualcuno è già stato inserito, se viene trovato viene posta a *true* e non viene ulteriormente inserito.

```
for (int j = 0; j<children.size(); j++){
  boolean found = false;
```

```
        for (int k = 0; k<this.characters.size(); k++){
             if(this.characters.get(k).getName()==
                        children.get(j)){
                  found = true;
                  break;
             }
        }       if (found){
             continue;
        }
        this.ontmanager.insertIdividual(nsOntology
                  + children.get(j),
                  nsOntology + ‘‘Character’’ );
        charactersns.add(nsOntology + children.get(j));
        characteradded++;
   }

   for (int j = 0; j<parentage.size(); j++) {
        boolean found = false;
        for (int k = 0; k<this.characters.size(); k++) {
             if (this.characters.get(k).getName().equals(
                       parentage.get(j))) {
                  found = true;
                  break;
             }
        }
        if (found){
             continue;
        }
        this.ontmanager.insertIdividual(nsOntology
                  + parentage.get(j),
                  nsOntology + ‘‘Character’’ );
        characteradded++;
        charactersns.add(nsOntology + parentage.get(j));
   }
   {
        boolean found =false;
        for (int k = 0; k<names.length; k++){
             if (names[k]==spouse || spouse.length()<2){
                  found = true;
                  break;
             }
       }
       if (!found){
             this.ontmanager.insertIdividual(
                       nsOntology + spouse,
                       nsOntology + ‘‘Character’’ );
             characteradded++;
             charactersns.add(nsOntology + spouse);
       }
   }
```

La variabile **parentage** è un ArrayList di stringhe che rappresentano i nomi dei genitori, stesso discorso per la variabile **children** per i figli, mentre

spouse è il nome del coniuge. La variabile **characteradded** è un intero, e serve a conteggiare il numero di personaggi ulteriormente inseriti. Nei vari cicli prima si controlla se il personaggio che è parente di un personaggio inserito è a sua volta già stato aggiunto, in caso negativo viene creato l'individuo corrispondente.

È stato scritto il seguente codice, che è all'interno di un ciclo, in modo che inserisca tutte le proprietà per tutti i personaggi di cui si conoscono i dati, cioé i personaggi che hanno pagine con template su i due siti utilizzati.

```
for (int j = 0; j<parentage.size(); j++){
      this.ontmanager.setProperty(nsOntology + name,
                nsOntology + ‘‘isChildOf’’,
                nsOntology + parentage.get(j));
}
for (int j = 0; j<children.size(); j++) {
      this.ontmanager.setProperty(nsOntology + name,
                nsOntology + ‘‘isParentOf’’,
                nsOntology + children.get(j));
}
for (int j = 0; j<othernames.length(); j++){
      if (othernames.charAt(j)!=‘,’){
           oname = oname + othernames.charAt(j);
      }
      if (othernames.charAt(j)==‘,’) {
           this.ontmanager.setDataProperty(nsOntology + name,
                     nsOntology + ‘‘hasOtherName’’,
                     oname);
           oname = ‘‘’’;
      }
      if( j==othernames.length()-1) {
           this.ontmanager.setDataProperty(nsOntology + name,
                     nsOntology + ‘‘hasOtherName’’,
                     oname);
           break;
      }
}
this.ontmanager.setProperty(nsOntology + name,
           nsOntology + ‘‘hisRaceIs’’,
           nsOntology + race.toLowerCase());
this.ontmanager.setProperty(nsOntology + name,
           nsOntology + ‘‘hisGenderIs’’,
           nsOntology + gender.toLowerCase());
this.ontmanager.setProperty(nsOntology + name,
           nsOntology + ‘‘hisSpouseIs’’,
           nsOntology + spouse);
}
totcharacter = size + characteradded;
for (int i = 1; i<totcharacter; i++){
      String ns1 = charactersns.get(i);
      for (int j = 0; j<i; j++){
```

```
            String ns2 = charactersns.get(j);
            ontmanager.setDifferent(ns1,ns2);
        }
    }
```

Nello spezzone di codice sopra riportato (che è all'interno di un ciclo for che serve a scorrere tutti i personaggi inseriti) nel primo ciclo for si inserisce la proprietà **isChildOf** di ogni personaggio, nel secondo ciclo si aggiunge la proprietà **isParentOf**, nel terzo ciclo si aggiunge **hasOtherName**. Infine si inseriscono le proprietà **hisRaceIs**, **hisGenderIs** e **hisSpouseIs**. La variabile **ontmanager** è un'istanza della classe **OntologyManager**, il metodo **setProperty** aggiunge le proprietà di tipo object mentre **setDataProperty** aggiunge quelle di tipo data. Nell'ultimo ciclo si impone che tutti i personaggi siano differenti tra loro.

La classe **QueryManager** crea le query usando le funzioni messe a disposizione da Jena. Questa classe mette a disposizione il metodo **makeQuery** che riceve come parametro una stringa in cui è presente la query, la esegue e restituisce il risultato salvandolo su un file. Tale metodo è implementato come segue:

```
public void makeQuery(String queryString) throws
                  FileQueryNotFoundException,
                  QueryFailureException {
     PrintStream qres;
     try {
          qres = new PrintStream(
                  new FileOutputStream( ".//queryresult.txt"));
          Query query = QueryFactory.create(queryString);
          QueryExecution qe =
            QueryExecutionFactory.create(query,
                      lotrmodel);
          ResultSet results = qe.execSelect();
          ResultSetFormatter.out(qres,results,query);
          qres.close();
          qe.close();
     } catch (FileNotFoundException e) {
          throw new FileQueryNotFoundException();
     } catch ( QueryParseException e1 ) {
          throw new QueryFailureException (e1.getLine(),
          e1.getColumn(),e1.getMessage());
     }
}
```

L' eccezione **QueryFailureException** è stata creata estendendo la normale classe **Exception**, in modo da segnalare il fatto che la query SPARQL è stata scritta male. È stata creata in modo da gestire l'eccezione **QueryParseException** senza interrompere l'esecuzione dell'applicazione. Quest'ultima viene intercettata, e nel blocco catch corrispondente viene generata la nuova eccezione, che viene rilevata nella classe **MainClass**. Se quest'ultima

classe la rileva avvisa attraverso l'interfaccia utente dell'errore di scrittura della query.

In **queryString** è memorizzata la query che si deve eseguire, il cui risultato viene memorizzato nel file queryresult.txt tramite la classe **PrintSream** messa a disposizione dalle librerie Java.

Per le regole è stata creata la classe **RuleClass**, che mette a disposizione il metodo **setNewPrefix** che permette di impostare prefissi e i relativi namespace diversi da quello già impostato. Quest'ultimo è "lotr" che corrisponde al namespace dell'ontologia.

Il metodo **setRules** crea il reasoner, legge la regola passata come argomento, e la fa applicare dal reasoner all'ontologia. Alla prima esecuzione tale metodo carica l'asserted model, una volta applicate le regole crea l'inferred model, salvato nel file LotrOntologyinferred.owl. Dalla seconda esecuzione in poi il metodo **setRule** carica questo file, in modo da conservare le regole applicate precedentemente.

Qui è riportato il codice scritto per il metodo **setRule**.

```
public void setRules(String rule)
           throws FileNotFoundException{

    FileOutputStream fout;
 fout = new FileOutputStream(ontol_file);

 try{

      OntModel modello = ModelFactory.createOntologyModel();
      File model = new File(ontol_file_inf);

      if( model.length()>0){

           FileInputStream fin =
                   new FileInputStream(ontol_file_inf);
            modello.read(fin, nsOntology);

      } else {

           modello = this.lotrmodel;

      }

      Reasoner reasoner =
           new GenericRuleReasoner(Rule.parseRules(rule));
      reasoner.setDerivationLogging(true);
      this.lotrmodel.write(fout);
      inf = ModelFactory.createInfModel(reasoner,
                modello);

      try {

           FileOutputStream foutinf =
```

```
                    new FileOutputStream(ontol_file_inf);
                inf.write(foutinf);

            } catch (FileNotFoundException e) {

                e.printStackTrace();

            }

        }catch (ParserException e){

            this.lotrmodel.write(fout);
            e.printStackTrace();

        }
    }
```

Nelle prime righe si controlla sul il file in cui si è salvato il modello inferito se l'ontologia è già stata creata. Se il file ha dimensione zero vuol dire che tale base di dati non è ancora stata costruita, quindi si carica l'ontologia asserita, se invece tale file ha una dimensione maggiore di zero si utilizza l'inferred model. Quindi si procede a creare il reasoner, a inserire la regola ed ad applicarla all'ontologia. Dopo di che si salva la base di conoscenza ottenuta. Per fare ciò si sfruttano le classi **ModelFactory** e **Reasoner** messe a disposizione da Jena.

Tutto ciò è utilizzabile dalla classe **MainClass** in cui è implementata una semplice interfaccia utente via console che dà la possibilità di scegliere cosa fare: scaricare i dati da internet o caricarli dai file, se costruire l'ontologia, fare query, costruire ed applicare regole per fare inferenza e infine salvare i dati su file.

Infine la classe **Statistic** è usata per memorizzare alcune statistiche sui dati relativi ai vari personaggi, usata all'inizio del progetto per decidere quali informazioni usare per costruire l'ontologia.

Qui di seguito sono riportate le statistiche dei campi, cioè fra i personaggi cercati, quanti hanno una determinata informazione.

Nome campo	percentuale sul totale
othernames	64,86%
race	53%
spouse	22.97%
children	2.7%
parentage	10%

Nei siti c'erano anche informazioni sulle date di nascita e di morte, non inserite, nonostante la gran parte dei personaggi avessero tale informazione, perché spesso imprecise (a volte dicevano semplicemente l'era in cui erano

nati o morti) e non esprimibili tramite un datatype esistente. Il calendario di Tolkien, infatti, si basa sul concetto di "Era" che non esiste nel nostro.

Altri campi sono stati scartati in quanto presenti in meno del venti percento dei personaggi. È stata fatta un'eccezione per i campi children e parentage perché è interessante riuscire a capire tramite query le relazioni di parentela, che sono poi ampliate grazie all'utilizzo delle regole; ad esempio con le regole è stata creata la proprietà **isBrotherOf**, che serve ad unire due personaggi se sono fratelli.

Con l'aiuto delle regole è stato possibile inoltre aggiungere relazioni **isParentOf**: dopo tale operazione la percentuale dei personaggi che possiedono informazioni sui figli è salita al dieci percento. Inoltre sono state ampliate le informazioni sulle razze: applicando la regola secondo cui i personaggi devono avere la stessa razza dei figli o dei genitori, la percentuale dei personaggi di cui si conosce la razza sale al sessanta percento.

In Figura 3.4 è riportato il grafico che rappresenta la parte dell'applicazione che recupera i dati, mentre nella Figura 3.5 è mostrata la parte dell'applicazione che si occupa del popolamento dell'ontologia, di fare le query e di creare e applicare le regole. Grazie ad esse esse si può avere un'idea di come le varie classi interagiscano tra loro, con l'ontologia e internet.

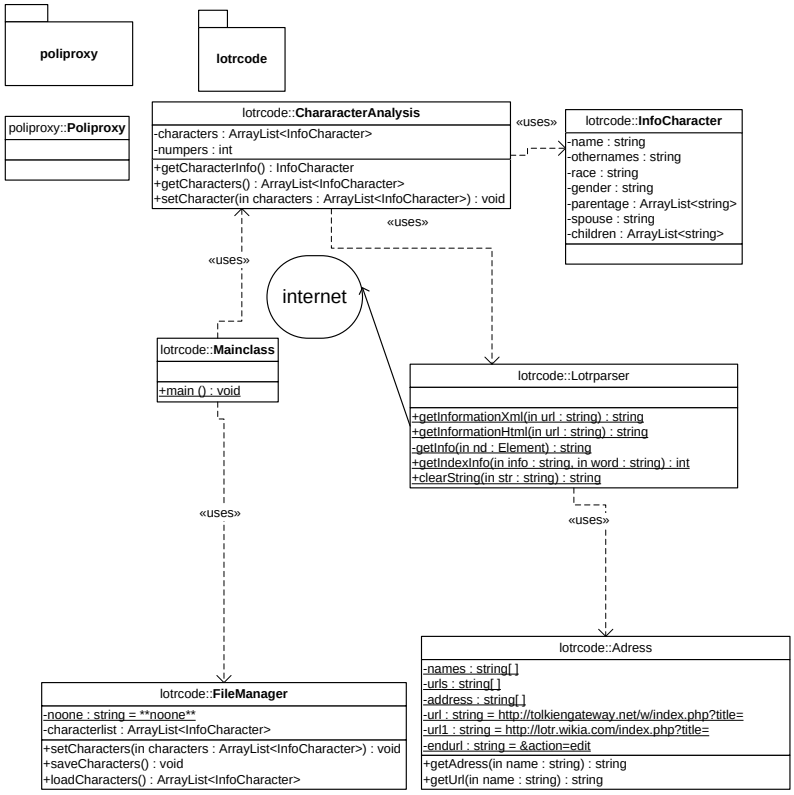

Figura 3.4: Recupero Dati

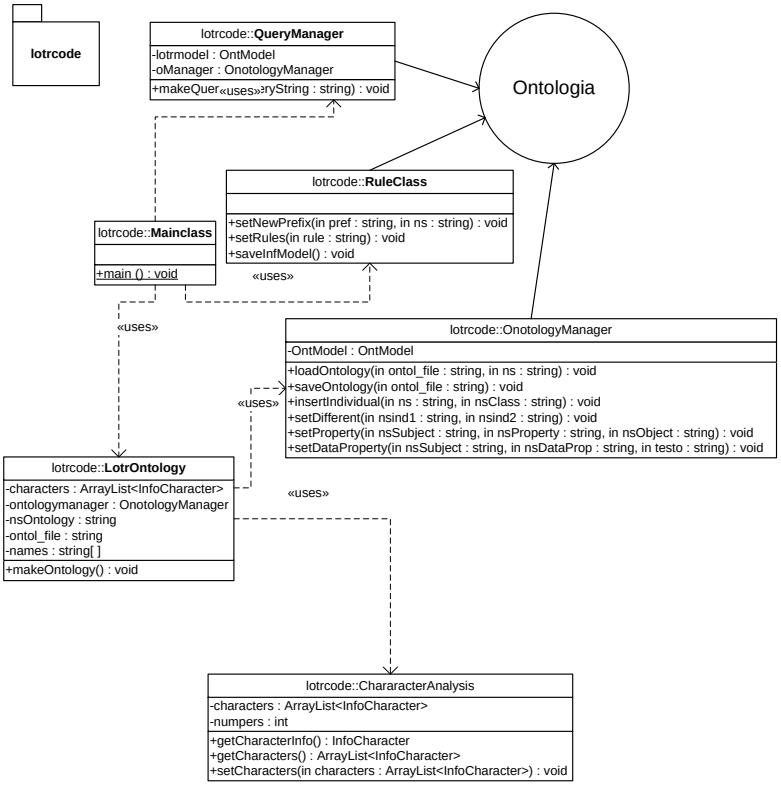

Figura 3.5: Gestione dell'ontologia

3.2.2 L'ontologia e tipo di regole

L'ontologia è composta da tre classi: **Character**, **Race**, **Gender**. La prima è usata per inserire gli individui che rappresentano, appunto, i personaggi. La seconda le razze, ed è già popolata in fase di costruzione dell'ontologia, mentre i personaggi vengono inseriti successivamente. L'ultima contiene due individui che rappresentano il genere, maschile o femminile.

Ci sono inoltre sei proprietà, cinque di tipo object e una di tipo dati. Esse sono:

- **hisGenderIs**
- **hisRaceIs**
- **hisSpouseIs**
- **isChildOf**
- **isParentOf**
- **hasOtherName**

La proprietà **hisGenderIs** va da **Character** a **Gender** e collega ogni personaggio al suo genere, **hisRaceIs** va da **Character** a **Race** e collega ogni personaggio alla sua razza, **hasOtherName** è di tipo dati, e collega **Character** con una stringa che rappresenta un altro nome che il personaggio può avere. Infatti nel mondo creato da Tolkien non è raro che un personaggio abbia anche altri nomi. Le altre tre hanno la classe **Character** sia come dominio che come codominio. In particolare **hisSpouseIs** mette in relazione due personaggi che si sono sposati, **isParentOf** collega un personaggio con i suoi figli e **isChildOf** collega un personaggio con i suoi genitori. Ovviamente la proprietà **hisSpouseIs** è simmetrica, mentre la proprietà **isChildOf** è l'inversa di **isParentOf**.

In Figura 3.6 è mostrato un semplice grafico che rappresenta l'ontologia.

Dopo aver fatto alcune query (si veda la Sezione 4.2), si è proceduto ad applicare delle regole. Con esse è stata fatta inferenza, cioè si sono ricavate tramite processi deduttivi basate sulla logica SHOIN(D_n) nuove informazioni allargando la base di conoscenza. Le regole sono state create e applicate utilizzando le funzioni e il reasoner messi a disposizione da Jena.

La prima è una regola che permette di trovare i fratelli. Essa fa in modo che il reasoner cerchi un personaggio che abbia più di una relazione con altri personaggi di tipo **isParentOf**. Questi ultimi vengono messi in relazione tra loro con la nuova proprietà **isBrotherOf**. Un'altra regola inserita serve per estendere le informazioni sulla razza a personaggi che non hanno tale informazione. In particolare si è imposto che se un personaggio è padre di un personaggio di una certa razza, anche il genitore deve essere della stessa razza. Nella Figura 3.7 è rappresentata la nuova ontologia. Inoltre è

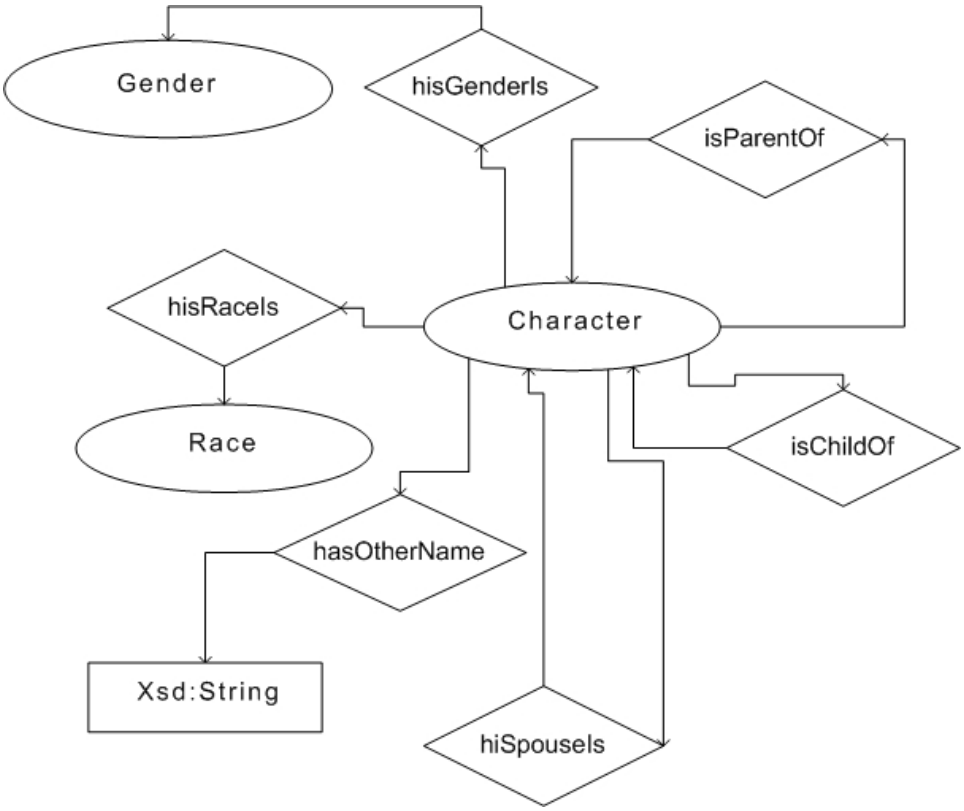

Figura 3.6: L'ontologia

stata creata una regola per rendere **isParentOf** effettivamente inversa della proprietà **isChildOf**.

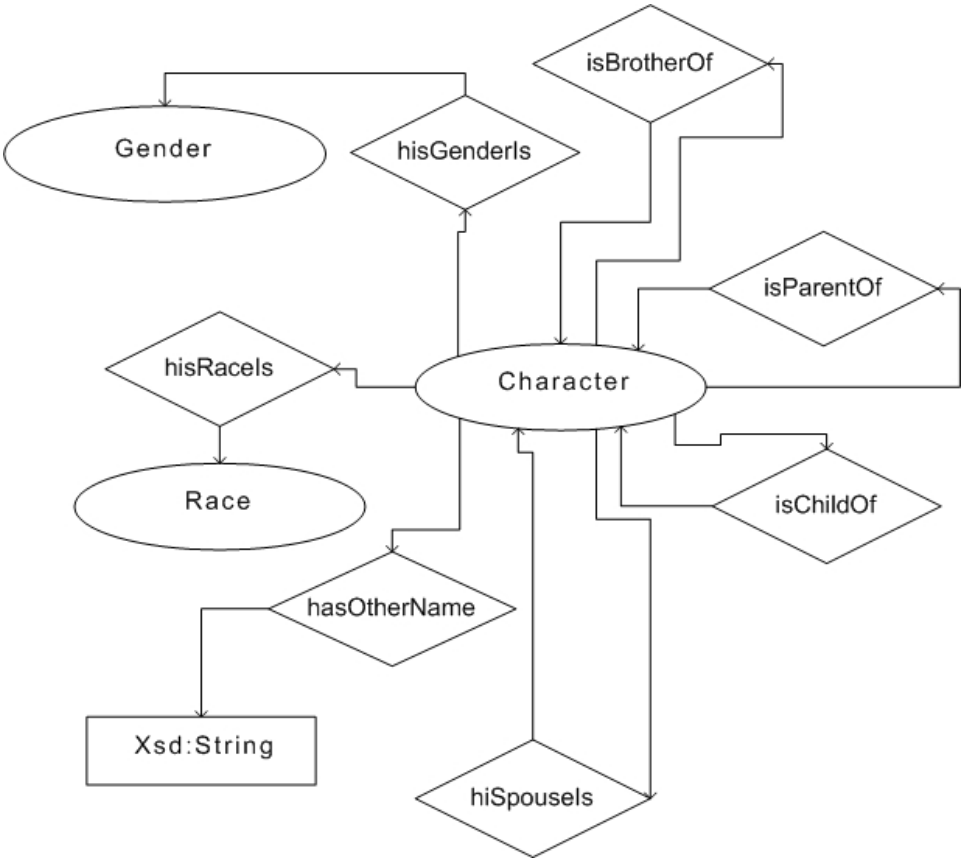

Figura 3.7: L'ontologia inferita

Capitolo 4

Esempi di utilizzo

«Coraggio, signor Frodo!», gridò. «Non posso portare io l'Anello, ma posso portare voi ed esso insieme. Alzatevi! Suvvia, signor Frodo, caro! Sam vi porterà in groppa. Ditegli dove deve andare, e lui vi andrà»

dal libro "Il Signore degli Anelli".

4.1 Esempi di esecuzione

Durante l'esecuzione dell'applicazione, prima di poter creare l'ontologia, bisogna recuperare i dati. Alla prima esecuzione è possibile recuperarli solo dai siti, in seguito (a patto di aver dato il comando di salvare i dati dopo il download dai siti) è possibile caricarli dai file. Le scelte su come recuperarle sono presentate attraverso una semplice interfaccia utente testuale. Il programma presenta un menù con i comandi che è possibile eseguire, tra i quali ci sono l'operazione di download dai siti e quella di caricamento da file.

4.1.1 Download dei dati dai siti

Quando si seleziona il comando di download per prima cosa viene chiesto di impostare il proxy: è possibile scegliere di lasciare quello di default, usare quello della rete del Politecnico di Milano o inserire un nuovo proxy. Quindi viene effettuata la connessione ai siti utilizzati, usando la classe **Address** per costruire gli indirizzi Web, e si recuperano i dati presenti nei wiki. Quando un personaggio non ha informazione tramite l'interfaccia utente si avvisa di questa mancanza e si procede al personaggio seguente. Una volta ottenuti, tutti i personaggi vengono inizialmente memorizzati ciascuno in un'istanza della classe **InfoCharacter** e tutte le istanze di questa sono inserite in un ArrayList. Tale lista viene passata alla classe **LotrOntology** che si occupa del popolamento dell'ontologia. Inoltre è possibile salvare i dati su file, in modo che in seguito si possano recuperare più facilmente e anche in mancanza di una connessione a internet.

4.1.2 Salvataggio e caricamento dei dati tramite file

Quando viene lanciato il salvataggio la classe **FileManager** crea i file, chiamandoli personaggio_nome.txt (dove "nome" rappresenta il nome del personaggio) e ci scrive dentro le informazioni. Tali file hanno il seguente formato:

Nome del personaggio
Soprannomi
Razza
Genere
Coniuge
Genitori
Figli

In questo modo l'operazione di caricamento è molto semplice, l'applicazione legge una riga per volta e memorizza ordinatamente le informazioni. Se un personaggio non ha un campo, sulla riga corrispondente è presente la stringa "**noone**". Per i campi soprannomi, genitori e figli, nel caso siano presenti più di un dato per quella riga (più di un soprannome, i genitori solitamente sono due e un personaggio può avere più di un figlio) si separa un termine dall'altro con la virgola, e per indicare che non ci sono ulteriori stringhe al termine della riga c'è il carattere #. Durante il caricamento (come nel caso precedente), ogni personaggio viene memorizzato in un'istanza della classe **InfoCharacter**, e ogni istanza è messa in un ArrayList che viene poi passato alla classe **LotrOntology** che si occuperà del popolamento dell'ontologia.

4.2 Esempi di query SPARQL

Una volta creata l'ontologia sono state preparate alcune query utilizzando SPARQL e sono state eseguite dall'applicazione Java, tramite la libreria Jena. Le query sono inserite tramite la semplice interfaccia implementata, la quale invoca il metodo **makeQuery** della classe **QueryManager**. Questa classe costruisce la query e salva su file il risultato, che viene poi letto dalla classe **MainClass** che implementa l'interfaccia utente e stampa a schermo ciò che legge sul file.

Come prima query è stato preparato il seguente comando, essa è una query molto semplice, che associa ai personaggi la propria razza:

```
PREFIX fo:<D:/Documenti/Poli/LotrOntology.owl#>
SELECT ?x ?y
WHERE { ?x fo:hisRaceIs ?y .}
ORDER BY ?y
```

Il risultato è mostrato nella tabella che segue.

Risultati query

x	y
fo:Aragorn_II	fo:dunedain
fo:Halbarad	fo:dunedain
fo:Gimli	fo:dwarf
fo:Elrohir	fo:elf
fo:Celeborn	fo:elf
fo:Glorfindel_of_the_Golden_Flower	fo:elf
fo:Galadriel	fo:elf
fo:Haldir	fo:elf
fo:Elladan	fo:elf
fo:Cirdan	fo:elf
fo:Elrond	fo:elf
fo:Legolas	fo:elf
fo:Arwen	fo:half-elf
fo:Gollum	fo:hobbit
fo:Frodo_Baggins	fo:hobbit
fo:Peregrin_Took	fo:hobbit
fo:Meriadoc_Brandybuck	fo:hobbit
fo:Samwise_Gamgee	fo:hobbit
fo:Rose_Cotton	fo:hobbit
fo:Bilbo_Baggins	fo:hobbit
fo:Lobelia_Sackville-Baggins	fo:hobbit
fo:Fredegar_Bolger	fo:hobbit
fo:Lotho_Sackville-Baggins	fo:hobbit
fo:Gandalf	fo:istari
fo:Sauron	fo:maia
fo:Radagast	fo:maia
fo:Saruman	fo:maia
fo:Boromir	fo:man
fo:Grama_Wormtongue	fo:man
fo:Hama	fo:man
fo:Eowyn	fo:man
fo:Denethor_II	fo:man
fo:Erkenbrand	fo:man
fo:Eomer	fo:man
fo:Gamling	fo:man
fo:Theoden	fo:man
fo:Hirluin	fo:man
fo:Imrahil	fo:man
fo:Faramir	fo:man
fo:Witch-king_of_Angmar	fo:man

Questi sono quasi tutti i personaggi di cui si dispone nel template: di alcuni esso non riportava nessuna informazione sulla razza e quindi non vengono restituiti dalla query. Essa ha il vantaggio di aggregare i personaggi a seconda della razza.

Un'altra query molto semplice ricerca i genitori di genere maschile, cioé i padri. Questo è il comando SPARQL utilizzato per trovare tale individui:

```
PREFIX fo:<D:/Documenti/Poli/LotrOntology.owl#>
SELECT DISTINCT ?x
WHERE { ?x fo:isParentOf ?y .
        ?x fo:hisGenderIs fo:male.}
```

Il risultato è rappresentato nella seguente tabella:

Risultati query
x
fo:Denethor_II
fo:Elrond

Sono stati trovati solo due personaggi perché molti padri corrispondono a personaggi che non hanno dati. Essi sono stati inseriti, ma non avendo dati non sono state aggiunte le relazioni **isParentOf** che partissero da loro e non ci sono informazioni sul genere. Inoltre poiché che si sta lavorando con l'asserted model, si sta operando sul modello con le caratteristiche che sono state esplicitamente dette; non si è fatto reasoning, operazione che avrebbe aggiunto tale proprietà (se non già presente) quando il personaggio risulta punto di arrivo della proprietà **isChildOf**, in quanto essa è inversa della relazione **isParentOf**.

Per cercare di trovare tutti i genitori si è fatta la seguente query:

```
PREFIX fo:  <D:/Documenti/Poli/LotrOntology.owl#>
SELECT DISTINCT ?x
WHERE { { ?y fo:isChildOf ?x} UNION
        { ?x fo:isParentOf ?y} }
```

Si cercano i genitori usando la sua proprietà inversa, in modo da trovare anche gli individui che sono genitore di qualcuno ma poiché non hanno proprietà non risultano essere genitori di un qualche personaggio.

Il risultato è rappresentato nella seguente tabella:

Risultati query
x
fo:Elrond
fo:Drogo_Baggins
fo:Belladona_Tuc
fo:Primula_Brandibuck
fo:Finduilas_of_Dol_Amroth
fo:Bungo_Baggins
fo:Denethor_II

La prossima query ha l'obiettivo di trovare i personaggi che sono fratelli:

```
PREFIX fo:  <D:/Documenti/Poli/LotrOntology.owl#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?x ?y
WHERE { ?x fo:isChildOf ?z .
          ?y fo:isChildOf ?z .
          ?x owl:differentFrom ?y }
```

Il risultato è rappresentato nella seguente tabella:

Risultati query	
x	y
fo:Elladan	fo:Elrohir
fo:Elladan	fo:Arwen
fo:Faramir	fo:Boromir
fo:Elrohir	fo:Elladan
fo:Elrohir	fo:Arwen

Questa query appena mostrata è già un primo esempio di semplice inferenza, infatti nell'ontologia non sono presenti le relazioni che rappresentano il fatto di essere fratelli.

La successiva query ha l'obiettivo di trovare i soprannomi dei personaggi.

```
PREFIX fo:<D:/Documenti/Poli/LotrOntology.owl#>
SELECT ?x ?y
WHERE { ?x fo:hasOtherName ?y }
```

Il risultato è mostrato nella seguenta tabella:

Risultati query

x	y
fo:Aragorn_II	Strider
fo:Lobelia_Sackville-Baggins	Mistress_Lobelia
fo:Sauron	Mairon
fo:Witch-king_of_Angmar	Lord_of_the_Nazgul
fo:Gimli	Lord_of_the_Glittering_Caves
fo:Samwise_Gamgee	Sam
fo:Gimli	Elf-friend
fo:Radagast	Tender_of_Beasts
fo:Gandalf	Tharkûn
fo:Saruman	Sharkey
fo:Imrahil	Prince_of_Dol_Amroth
fo:Forlong	the_Old
fo:Peregrin_Took	Pippin
fo:Gimli	Lockbearer
fo:Saruman	Wise
fo:Beregond	Beregond_of_the_Guard
fo:Eeowyn	White_Lady_of_Rohan
fo:Saruman	Curunìr
fo:Rose_Cotton	Rosie
fo:Peregrin_Took	Thain_Peregrin_I
fo:Radagast	Bird_Friend
fo:Èomer	Èomer_Èadig
fo:Cìrdan	Nowë
fo:Erkenbrand	Master_of_the_Westfold
fo:Gandalf	The_White
fo:Gandalf	GandalfWand-elf
fo:Witch-king_of_Angmar	Chief_of_the_Nine
fo:Gandalf	Mithrandir
fo:Celeborn	Telerin
fo:Lotho_Sackville-Baggins	Pimple
fo:Erkenbrand	Lord_of_the_Westfold
fo:Meriadoc_Brandybuck	the_Magnificent
fo:Gandalf	The_Grey
fo:Saruman	White
fo:Meriadoc_Brandybuck	Merry
fo:Gamling	Gamling_the_Old
fo:Legolas	Greenlea
fo:Witch-king_of_Angmar	Black_Captain
fo:Erkenbrand	Lord_of_the_Deeping-coomb
fo:Saruman	Of_many_colours
fo:Hirluin	of_the_Green_Hills
fo:Gollum	Sméagol
fo:Gandalf	Olòrin
fo:Witch-king_of_Angmar	Lord_of_Minas_Morgul
fo:Aragorn_II	Elessar
fo:Sauron	Necromancer
fo:Eowyn	Lady_of_the_Shield-arm
fo:Lobelia_Sackville-Baggins	Lobelia_Bracegirdle
fo:Sauron	Annatar
fo:Radagast	Aiwendil

fo:Fredegar_Bolger	Fatty
fo:Aragorn_II	Thorongil
fo:Forlong	the_Fat
fo:Gandalf	Incànus
fo:Radagast	Bird-tamer
fo:Eowyn	Dernhelm
fo:Peregrin_Took	Ernil_i_Pheriannath
fo:Gandalf	Gandalf_Greyhame
fo:Radagast	The_Brown
fo:Saruman	Curumo
fo:Hirluin	the_Fair
fo:Gandalf	The_White_Rider
fo:Meriadoc_Brandybuck	Kalimac_Brandagamba
fo:Théoden	Ednew
fo:Peregrin_Took	Razanur_Took
fo:Gandalf	Stormcrow

La seguente query ha il compito di cercare il genere di tutti gli elfi presenti nell'ontologia.

```
PREFIX fo:<D:/Documenti/Poli/LotrOntology.owl#>
SELECT ?x ?y
WHERE { ?x fo:hisRaceIs fo:elf .
          ?x fo:hisGenderIs ?y }
```

Essa ha prodotto il risultato visualizzato nella tabella che segue.

Risultati query

x	y
fo:Elrond	fo:male
fo:Cirdan	fo:male
fo:Haldir	fo:male
fo:Elladan	fo:male
fo:Galadriel	fo:female
fo:Elrohir	fo:male
fo:Glorfindel_of_the_Golden_Flower	fo:female
fo:Celeborn	fo:male
fo:Legolas	fo:male

4.3 Esempi di applicazione delle regole

Come già detto sono state quindi applicate delle regole. La prima ha permesso di rendere effettivamente inversa la proprietà **isParentOf**, imponendo che se un personaggio è figlio di un altro individuo, quest'ultimo deve anche essere il genitore del primo. La regola usata è la seguente:

```
?x lotr:isChildOf ?y -> ?y lotr:isParentOf ?x
```

Dove lotr è il prefisso corrispondente al namespace dell'ontologia.

Quindi è stata applicata la regola per ottenere i fratelli. La regola è la seguente:

```
?x lotr:isParentOf ?y  and  ?x lotr:isParentOf ?z
                    and
?y owl:differentFrom ?z  -> ?y lotr:isBrotherOf ?z
```

Sono state applicate anche le seguenti regole:

```
?x lotr:isParentOf ?y  and  ?y lotr:hisRaceIs ?z ->
            ?x lotr:hisRaceIs ?z
?x lotr:isParentOf ?y  and  ?x lotr:hisRaceIs ?z ->
            ?y lotr:hisRaceIs ?z
```

Questa regola fa in modo che i genitori abbiano la stessa razza dei figli.

Dopo l'applicazione delle regole si è rifatta la query per cercare i genitori, e si è potuta effettuare solo usando la relazione **isParentOf**.

I risultati della query sono stati:

Risultati query
x
fo:Elrond
fo:Drogo_Baggins
fo:Belladona_Tuc
fo:Primula_Brandibuck
fo:Finduilas_of_Dol_Amroth
fo:Bungo_Baggins
fo:Denethor_II

I risultati sono gli stessi della sezione precedente, ma la query risulta più semplice. Purtroppo non si è potuto restringere la ricerca ai soli padri, in quanto gli unici tra i personaggi sopraelencati che hanno informazioni sul genere (ed esso corrisponde al genere maschile) sono Denethor e Elrond.

Con l'applicazione delle regole anche la ricerca dei fratelli si semplifica, basta utilizzare la proprietà **isBrotherOf**:

```
PREFIX fo:<D:/Documenti/Poli/LotrOntology.owl#>
SELECT DISTINCT ?x ?y
WHERE { ?x fo:isBrotherOf ?y.  }
```

I risultati sono gli stessi della sezione precedente:

Risultati query	
x	y
fo:Elladan	fo:Elrohir
fo:Elladan	fo:Arwen
fo:Faramir	fo:Boromir
fo:Elrohir	fo:Elladan
fo:Elrohir	fo:Arwen

Si sono di nuovo ricercate le razze dei personaggi:

Risultati query

x	y
fo:Aragorn_II	fo:dunedain
fo:Halbarad	fo:dunedain
fo:Gimli	fo:dwarf
fo:Elrohir	fo:elf
fo:Celeborn	fo:elf
fo:Glorfindel_of_the_Golden_Flower	fo:elf
fo:Galadriel	fo:elf
fo:Haldir	fo:elf
fo:Elladan	fo:elf
fo:Cirdan	fo:elf
fo:Elrond	fo:elf
fo:Legolas	fo:elf
fo:Arwen	fo:half-elf
fo:Gollum	fo:hobbit
fo:Frodo_Baggins	fo:hobbit
fo:Peregrin_Took	fo:hobbit
fo:Meriadoc_Brandybuck	fo:hobbit
fo:Samwise_Gamgee	fo:hobbit
fo:Belladona_Tuc	fo:hobbit
fo:Bungo_Baggins	fo:hobbit
fo:Drogo_Baggins	fo:hobbit
fo:Primula_Brandibuck	fo:hobbit
fo:Rose_Cotton	fo:hobbit
fo:Bilbo_Baggins	fo:hobbit
fo:Lobelia_Sackville-Baggins	fo:hobbit
fo:Fredegar_Bolger	fo:hobbit
fo:Lotho_Sackville-Baggins	fo:hobbit
fo:Gandalf	fo:istari
fo:Sauron	fo:maia
fo:Radagast	fo:maia
fo:Saruman	fo:maia
fo:Boromir	fo:man
fo:Grama_Wormtongue	fo:man
fo:Hama	fo:man
fo:Eowyn	fo:man
fo:Denethor_II	fo:man
fo:Erkenbrand	fo:man
fo:Eomer	fo:man
fo:Finduilas_of_Dol_Amroth	fo:man
fo:Gamling	fo:man
fo:Theoden	fo:man
fo:Hirluin	fo:man
fo:Imrahil	fo:man
fo:Faramir	fo:man
fo:Witch-king_of_Angmar	fo:man

Rispetto a prima dell'applicazione dele regole, si sono trovati alcuni personaggi in più, aumentando le informazioni disponibili.

Capitolo 5

Conclusioni

«Ricordi le parole di Gandalf: persino Gollum potrebbe avere ancora qualcosa da fare? Se non fosse per lui, Sam, non avrei distrutto l'Anello. La missione sarebbe stata vana, proprio alla fine. Quindi perdoniamolo! La missione è compiuta, e tutto è passato.»

dal libro "Il Signore degli Anelli".

5.1 Conclusioni

Questo lavoro presenta una possibile soluzione ai limiti dei wiki, cioè presenza di dati non strutturati difficili da trovare ed elaborare automaticamente. L'applicazione creata è in grado di estrarre dati da queste pagine web, e di organizzarli in maniera strutturata in una ontologia implementata in OWL. Su di essa è stato quindi possibile effettuare delle ricerche con query scritte usando il linguaggio SPARQL, è stato così possibile trovare le informazioni desiderate in maniera semplice.

Con l'applicazione creata è possibile costruire e applicare regole in grado di inferire nuova conoscenza, cioè di ampliare l'ontologia già costruita, in questo modo si è riusciti a completare alcune informazioni sui vari personaggi, ad esempio (come mostrato negli esempi di query) quelle di parentela tra i personaggi.

Naturalmente il progetto ha presentato alcune complicazioni: i siti utilizzati hanno informazioni lacunose sui personaggi, e molti di essi sono sprovvisti di alcune (se non di tutte) informazioni, problema parzialmente risolto grazie all'applicazione delle regole. Inoltre sono state necessarie alcune modifiche manuali per sistemare alcuni particolari, relativi ai personaggi, ma la gran parte del lavoro è svolto in maniera automatica, risparmiando la fatica di dover recuperare i dati manualmente.

Un altra difficoltà riscontrata è stata causata dal fatto che nello stesso sito i campi per indicare una informazione sono chiamati all'interno del codice del sito in maniera leggermente differente. Ad esempio in un personaggio il nome di un campo è scritto tutto minuscolo, con i due punti subito attaccati alla

parola, mentre in una pagina di un'altro individuo lo stesso campo è scritto con la prima lettera maiuscola, al posto dei due punti c'è l'uguale e con uno spazio che precede tale carattere. Il problema è stato risolto realizzando il codice in maniera che ne tenga conto, ma questo ha reso il codice un po' pesante e meno leggibile.

Nonostante questo l'applicazione è in grado di semplificare l'operazione di estrazione dati, organizzandoli in un'ontologia in cui si possono fare facilmente ricerche avanzate; inoltre grazie alle regole è stato possibile colmare almeno in parte alcune lacune di informazioni, e ricavare nuovi dati.

5.2 Possibili sviluppi futuri

Una prima piccola estensione potrebbe essere l'aggiunta di altri siti basati su wiki da cui trarre più informazioni e personaggi, stando anche attenti che riportino informazioni sulle parentele, così da essere poi in grado di costruire i vari alberi genealogici dei personaggi del mondo creato da Tolkien.

Un altro possibile sviluppo potrebbe essere l'estensione di questo lavoro in modo da raccogliere dati su personaggi di altri libri o film fantasy, modificando la parte del codice relativo alla formazione degli indirizzi dei siti.

Si potrebbe anche estendere l'ontologia in modo che sia possibile inserire personaggi di libri qualsiasi, o modificarla in maniera che possa contenere informazioni di altro tipo, mettendo così a disposizione uno strumento per raccogliere dati e di organizzarli in maniera strutturata. Su tali informazioni si potrà facilmente fare ricerche avanzate e saranno utilizzabili anche da parte di elaboratori, che potranno lavorare facilmente su tale base di conoscenza.

Un utile aggiunta potrebbe essere la creazione di un interfaccia grafica, che metta a disposizione uno strumento per fare ricerca sull'ontologia che costruisca in automatico le stringhe SPARQL, in modo che chiunque possa ricercare le informazioni volute senza conoscere nello specifico le tecnologie usate.

Bibliografia

[1] Wikipedia. Storia del web. http://it.wikipedia.org/wiki/Storia_di_Internet, 2009.

[2] Professore Marco Colombetti. Dispensina di ingegneria della conoscenza. Pagina personale del professore sul sito del DEI: www.dei.polimi.it, 2008-2009.

[3] Wikipedia. Resource description framework. http://it.wikipedia.org/wiki/Resource_Description_Framework, 2009.

[4] Wikipedia. Xml. http://it.wikipedia.org/wiki/XML, 2009.

[5] Ing. Mario Arrigoni Neri. Dispensina xml. Disponibile sulla pagina personale del professor Colombetti sul sito:www.dei.polimi.it, 2008-2009.

[6] Ing. Mario Arrigoni Neri. Dispensina xsd. Disponibile sulla pagina personale del professor Colombetti sul sito:www.dei.polimi.it, 2008-2009.

[7] Ing. Mario Arrigoni Neri. Dispensina dtd. Disponibile sulla pagina personale del professor Colombetti sul sito:www.dei.polimi.it, 2008-2009.

[8] Wikipedia. wiki. http://it.wikipedia.org/wiki/Wiki, 2009.

[9] Wikipedia. Template: Infobox. http://it.wikipedia.org/wiki/Template:Infobox, 2009.