Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

L'APPLICABILITÀ DEI CONTESTI NEL WEB SEMANTICO

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
ANDREA PAGLIARANI

Sessione Straordinaria
Anno Accademico 2019/20

Scegliamo il nostro mondo successivo in base a ciò che apprendiamo in questo. Se non impari nulla, il mondo di poi sarà identico a quello di prima, e avrai anche là le stesse limitazioni che hai qui, gli stessi handicap.

Richard Bach

Indice

Introduzione

L'idea di base di questo lavoro, si basa sullo studio di possibili soluzioni al problema dell'esprimere concetti complessi attraverso le strutture del *Semantic Web*.
Le normali triple, strutture classiche di espressione delle asserzioni nel Web Semantico, non permettono di gestire aspetti complessi di un'asserzione, come per esempio il contesto di validità di quest'ultima, oppure l'autore o la data di creazione.
Non è infatti sufficiente costruire concetti complessi, basandosi sull'uso di costrutti valutati da una macchina allo stesso livello: servono strutture più complesse, in grado di realizzare una gerarchia semantica che incapsuli concetti anche estesi, all'interno di un contesto utilizzabile e direttamente riconducibile all'artefatto stesso.
Adattare il *Semantic Web* a una struttura in grado di esprimere concetti complessi come quelli di un contesto, o meglio di una provenance dell'artefatto, potrebbe portare a una duttilità dei contenuti, sino a farli diventare un costrutto modellabile a piacere dall'utente che li esamina.
Immaginiamo un ambiente in cui sia possibile scegliere quale contenuto, relativamente a un certo argomento, si adatti meglio alle nostre esigenze: questo permetterebbe non solo di avere accesso ad informazioni articolate e collegate, ma anche ricche di una storicizzazione, un versionamento e della capacità di adattarsi alle esigenze del momento.
Questo contesto diviene più che mai rilevante in questi anni, quando il Web è divenuto così ricco di opinioni contrastanti e prive di qualsiasi filtro o contestualizzazione: gli effetti di questa caratteristica sono disparati e possono essere più o meno apprezzabili.

Inoltre, pensando a come sia cambiato nel tempo l'accesso alle informazioni, è probabile che sia latente l'esigenza di disporre di uno strumento che vada oltre il semplice ipertesto, e permetta di evolvere il contenuto fino a un punto di adattabilità completo.
Quello che cercheremo di fare, sarà dunque verificare sul piano pratico, la possibilità di arricchire degli artefatti RDF, sfruttando i moderni standard e le tecnologie attuali, di informazioni utilizzabili da un elaboratore, per manipolare il contesto in maniera *intelligente*.
Per comprendere appieno il concetto alla base di queste affermazioni, bisogna considerare le evoluzioni che il Web ed il *Semantic Web* hanno affrontato nel corso della loro giovane vita.
Per chi, come me, ha vissuto a cavallo tra gli anni '90 e 2000, il Web ha rappresentato certamente una delle rivoluzioni più importanti ed eclatanti che abbiamo avuto la possibilità di tàngere personalmente.
Peter Hancock definisce la tecnologia come "la reazione naturale dell'uomo alla distanza tra la percezione e l'azione" [1]: l'impatto che le tecnologie di Internet e del WWW hanno avuto per queste due dimensioni è stato senza dubbio rilevante, portando a un cambiamento radicale alle dinamiche sociali, all'approccio risolutivo dei problemi, oltre che alla nostra percezione e conoscenza della realtà.
Per fare alcuni esempi basti pensare a come si sia diffuso il mondo degli acquisti on-line, tanto da cannibalizzare spesso i canali distributivi tradizionali, ai social e al loro impatto globale, culturale e politico ed alle infinite possibilità di accesso immediato ad un archivio di informazioni unificato, le cui dimensioni non sarebbero state nemmeno immaginabili in precedenza.
Se volessimo pensare sempre in termini di distanza tra percezione e azione, penso che la nascita del WWW possa certamente ricondursi ad una necessità dell'uomo di comunicazione e condivisione collettiva della conoscenza: dapprima questa esigenza è sorta nel ristretto ambito della ricerca scientifica (all'interno del CERN di Ginevra), evolvendo

poi in un sofisticato sistema diffuso di servizi.

La capacità di facilitare l'accesso alle informazioni rappresenta quindi, quasi certamente, uno degli aspetti maggiormente proficui e accattivanti di Internet ed anche uno degli impulsi alla base di questa dissertazione.

Accanto all'evoluzione dei servizi offerti sul Web, si può constatare una concomitante evoluzione delle tecnologie disponibili: le pagine statiche vengono soppiantate da contenuti dinamici, i cascade style sheet forniscono uno strumento potente per la formattazione delle pagine, JavaScript evolve e consolida la propria posizione di "linguaggio del Web" sino a conquistare un posto di rilievo, anche tra i linguaggi server side, sull'onda della filosofia *JavaScript Everywhere* e gli ipertesti evolvono cercando di legare i contenuti non solo tra loro, ma anche al loro contesto semantico grazie alle ontologie ed al *Semantic Web*.

L'idea di base su cui si sviluppa questo progetto di tesi, è l'applicabilità dei contesti e la gestione delle congetture nei grafi rdf all'interno di casi d'uso reali: questi appaiono particolarmente utili nella valutazione della semantica dei contenuti, per identificare e successivamente "collassare alla realtà" triple raccolte in ambito di incertezza e di variabilità nel tempo e idealmente per una valutazione dinamica di contenuti web con il fine di confrontare la validità ed attualità delle asserzioni.

È facile immaginare come, all'interno di un contenuto web, si possano trovare informazioni non corrette o comunque non aggiornate al variare del contesto: immaginiamo per esempio un testo scientifico che necessita di aggiornamenti, in base al progresso della ricerca, dove sarebbe estremamente utile modificare il contenuto.

Questo può essere fatto, come suggerito da Barabucci, Tomasi e Vitali, sfruttando il costrutto delle quadruple, previste nel nuovo standard RDF 1.1, arricchendo le asserzioni di informazioni sul contesto a cui fanno riferimento ed in particolare, associando la congettura a una particolare fonte e ad un riferimento temporale, valutarne veridicità ed attualità.

Il primo passo, sarà dunque quello di creare un semplice contesto privo di particolari complessità, per la gestione, l'inserimento e l'archiviazione di ontologie: la costruzione avverrà tramite una semplice app web, utilizzabile dall'utente per la sottomissione di costrutti semantici tramite formato Turtle (convertito in TriG), N-Triples (convertito in N-Quads) o query SPARQL.

Le congetture così inserite, verranno automaticamente associate dal sistema all'utente che le ha inserite e ad una data: per raggiungere questo scopo utilizzeremo le estensioni fornite da RDF1.1 per il supporto di multi grafi (quadruple).

In ultima istanza, per mostrare il potenziale del *Semantic Web*, l'applicazione si occuperà di interpretare le triple inserite, restituendo un costrutto lessicale comprensibile ad un essere umano.

Potenziali estensioni dell'applicazione possono essere ricercate nella creazione di un'interfaccia per associare ad ogni ontologia un determinato punteggio di affidabilità, che agevolerà poi gli utilizzatori nel collassare alla realtà le triple presenti, sulla base delle preferenze dell'utente.

Capitolo 1

Il Web Semantico

1.1 Il modello RDF

In questo capitolo parleremo del *Semantic Web*, di come è nato, dell'idea di base da cui si è evoluto e dello stato dell'arte delle tecnologie e della diffusione di esse nell'odierno Web.

Il *Semantic Web* nasce da un'idea di Tim Berners-Lee, che assieme a Robert Cailliau fu uno degli inventori del World-Wide Web, pubblicata nel maggio del 2001 in un articolo ("The Semantic Web"), scritto assieme a James Hendler e Ora Lassila, in cui teorizzava un'evoluzione dei contenuti del WWW verso un loro design non solo *human readable* ma anche in grado di permetterne la manipolazione da parte delle macchine.

Il concetto di base, secondo quanto descritto da Lee, era che un *agent* fosse in grado "navigando di pagina in pagina di compiere prontamente azioni sofisticate per gli utenti [...], facendo diventare il Web Semantico non una separazione dell'attuale WWW, quanto una sua estensione, in cui alle informazioni viene dato un significato specifico, portando a migliori possibilità di collaborazioni tra uomo e macchina"[2].

Per raggiungere suddetto scopo, divengono quindi necessarie strutture logiche in grado di permettere a una macchina di interpretare il significato di un'asserzione e dei suoi

elementi, deducendoli dal contesto.

Pensando alla complessità della lingua e della sua semantica, che per natura rimane pienamente intelligibile esclusivamente ad un'entità senziente non artificiale, diviene facile comprendere come la struttura di cui sopra, debba godere di caratteristiche che la rendano inequivocabile agli occhi di un'intelligenza artificiale e quindi rendere ogni asserzione priva di qualsiasi interpretabilità.

Al di fuori del mondo "aritificiale", esistono diversi strumenti che aiutano gli umani nell'interpretazione del linguaggio e da cui il mondo del *Semantic Web* ha tratto liberamente ispirazione e si è evoluto: tra questi, quelli che probabilmente hanno una maggior rilevanza, in questo contesto, sono le tassonomie.

Prima di parlare di tassonomie, è importante introdurre due concetti fondamentali, ovvero i metadati e i vocabolari controllati.

I metadati (ovvero i dati dei dati) hanno rivestito e rivestono tutt'ora un ruolo di rilievo nell'indicizzazione del Web: prima ancora dell'avvento del *Semantic Web*, sono stati un elemento prodromico di quest'ultimo, e Berners-Lee li definiva come "informazioni *machine understandable* su una risorsa web o altre cose"[3] (difatti lo stesso documento cita le evoluzioni successive, definendo i metadati come una introduzione alla medesima idea); essi introducono ed intersecano informazioni relative a un documento, che un agente (inteso come un applicativo che compie una qualche azione) può usare per avere un'interpretazione più agevole dei contenuti.

Da notare che, nel caso dei metadati, questi ultimi vengono rappresentati come una serie di asserzioni indipendenti tra loro: questa rappresenta certamente una importante differenza con RDF, di cui parleremo successivamente, che presenta invece la possibilità di costruire gerarchie e collegamenti con altri elementi, in maniera simile alle tassonomie.

Parlando di vocabolario controllato, invece, introduciamo il concetto di un insieme condiviso di terminologie non ambigue che deve necessariamente essere completo rispetto al dominio dei valori possibili.

Alla luce di quanto sopra, possiamo facilmente evidenziare quelle che sono state le problematiche che hanno portato a concettualizzare il Web Semantico: i metadati sono, appunto, dei dati che permettono di arricchire di informazioni un oggetto e soffrono dei problemi di ambiguità semantica precedentemente descritti e del limite legato alla loro indipendenza; se consideriamo, inoltre, che i vocabolari controllati soffrono dei limiti e delle rigidità dovute alle loro stesse caratteristiche, diventa estremamente difficile costruire un modello scalabile ed estensibile con questi due strumenti.
Tassonomie e più in particolare i tesauri, giungono quindi ad estendere orizzontalmente le relazioni gerarchiche tra vocaboli, e portano una soluzione parziale alla loro rigidità; le tassonomie, strumento tutt'altro che nuovo (considerando che risale al XVIII secolo), sono state storicamente utilizzate per classificare gli esseri viventi: più in generale, creano una gerarchia tra i termini di un vocabolario controllato e sono quindi fortemente correlate, assieme ai tesauri (che sono di fatto tassonomie usate per creare relazioni orizzontali tra termini), con il modello relazionale e gerarchico utilizzato per i grafi RDF.
Evoluzione ultima di tutto ciò, sono le ontologie, ovvero uno strumento estremamente evoluto, che permette di creare gerarchie relazionali, non solo all'interno di un vocabolario controllato, ma costruendo una composizione di classi e relazioni in grado di estendersi all'infinito.
Ovviamente, come spesso accade nel mondo della digitalizzazione delle informazioni, per strutturare il modello di conoscenza proposto dal *Semantic Web* si è resa necessaria l'adozione di un framework condiviso, che la storia ha individuato in RDF (Resource Description Framework): RDF è un "modello standard per lo scambio dei dati nel Web [...] che ne facilita l'evoluzione degli schemi nel tempo"[4], rendendolo di fatto perfetto (o quasi) per un contesto così vasto e in continua evoluzione come quello del Web.
RDF estende il concetto degli ipertesti e della loro struttura per permettere l'associazione di un IRI con altri elementi (altri IRI) con un paradigma molto semplice composto da triple del tipo soggetto-predicato-oggetto, garantendo quindi la possibilità di fare as-

serzioni all'interno di questo modello.
Nonostante il potenziale espressivo di un costrutto simile, rimane una difficoltà evidente nella possibilità di esprimere concetti complessi: immaginando di voler costruire un'asserzione, che abbia una contestualizzazione (e.g. Cicerone è stato console nel 63 a.C.), l'uso di un artefatto limitato alla struttura soggetto-predicato-oggetto è assolutamente insufficiente.
Le moderne linearizzazioni di RDF, mettono però a disposizione alcune strutture interessanti che potrebbero fare al caso nostro e che vedremo nel prossimo capitolo.

1.2 Le linearizzazioni di RDF

Questa sezione è dedicata all'analisi delle principali linearizzazioni di RDF e della loro storia: di fatto ci sono stati diversi tentativi di strutturare le triple all'interno di una sintassi più o meno intelligibile e ricca di zuccheri sintattici.
Tra questi, quella che può essere probabilmente vista come la più classica, principalmente perché legata ad uno standard di markup dalle origini storiche (in relazione alla storia del Web), ovvero RDF/XML.
XML è infatti un metalinguaggio ispirato a SGML, linguaggio originariamente utilizzato per la formattazione di documenti governativi e industriali in una struttura *machine readable*, che fu elemento chiave per lo sviluppo del primo WWW da parte di Berners-Lee.
RDF/XML basa quindi la sua struttura su un modello che racchiude gli IRI delle triple all'interno di QName, annidando altre triple all'interno di questi e permettendo la definizione di una struttura ordinata, anche se evidentemente verbosa e rigida.
Tutti i QNames hanno un nome legato a un vocabolario controllato, che può essere espresso tramite un prefisso breve o essere dichiarato all'interno dell'asserzione stessa.
I letterali RDF possono essere esclusivamente nodi oggetto, diventando quindi stringhe

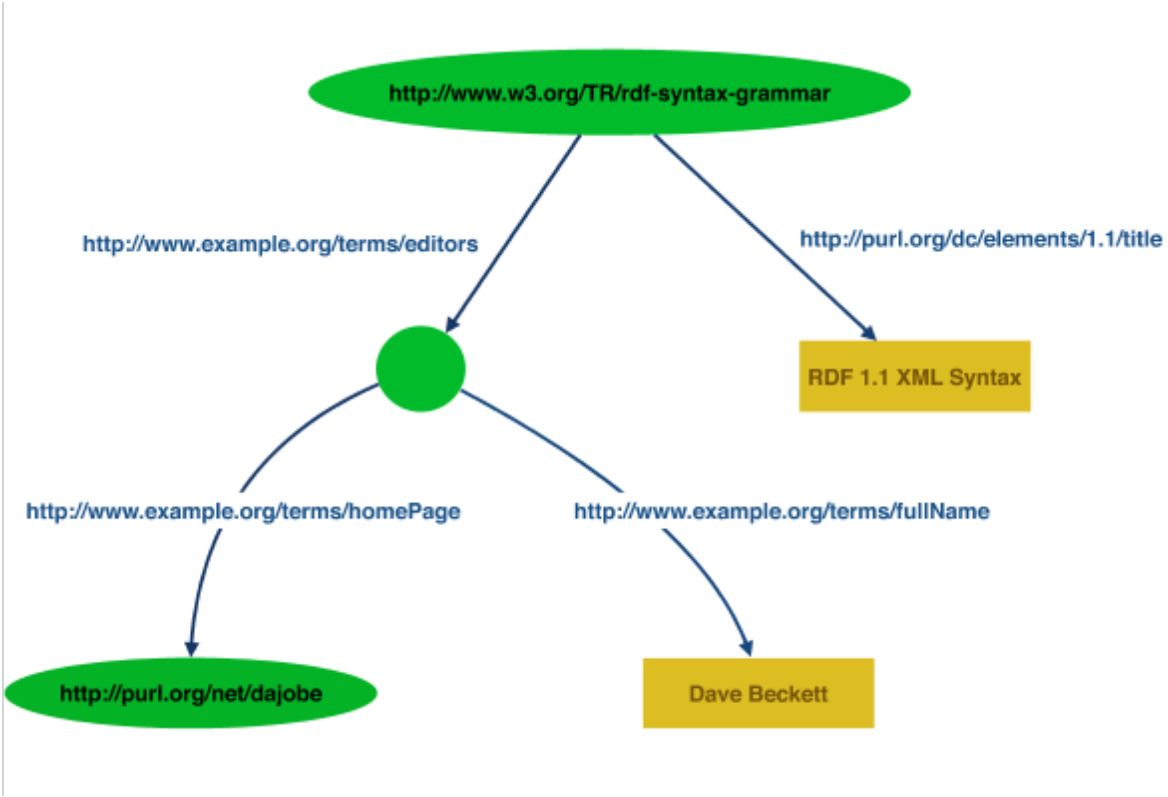

Figura 1.1: Grafo di esempio per il modello RDF/XML

racchiuse in tag XML o valori di attributi XML.
I grafi, possono essere rappresentati in RDF/XML, considerandoli come una raccolta di percorsi (figura 1.1) divenendo elementi annidati nella struttura XML.

La possibilità di creare strutture annidate, diviene uno strumento molto potente per una rappresentazione gerarchica, a discapito di una struttura scarsamente intelligibile e abbastanza rigida.

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
```

```
    <rdf:Description>
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

[23]

Un secondo modello sintattico, decisamente interessante per alcuni *syntactic sugar* che ne semplificano l'utilizzo, è Turtle: un linguaggio per sua stessa definizione conciso (*The Terse RDF Triple Language*) che permette di svincolarsi dalle rigide logiche di XML e di esprimere concetti complessi in maniera più concisa.
Rispetto a RDF/XML, gli elementi più interessanti, sono probabilmente la possibilità di dichiarare i prefissi (senza dover quindi specificare di volta in volta l'IRI completo), la possibilità di separare i predicati afferenti uno stesso soggetto con un punto e virgola, così come gli oggetti con una virgola.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

[20]

N-Triples è invece un *subset* di Turtle: permette di fatto la costruzione di triple su un'unica linea, perdendo però la possibilità di costruire artefatti complessi in maniera semplice come con il suo *superset* e rendendolo quindi, probabilmente, meno interessante (se non per la semplificazione rispetto ad RDF/XML).

```
<http://one.example/subject1> <http://one.example/predicate1>
   <http://one.example/object1> . # comments here
# or on a line by themselves
_:subject1 <http://an.example/predicate1> "object1" .
_:subject2 <http://an.example/predicate2> "object2" .
```

[21]

Sia Turtle che N-Triples godono di estensioni in grado di rappresentare grafi, ovvero le strutture maggiormente complesse introdotte con RDF1.1, che per loro natura apportano una rappresentazione gerarchica delle asserzioni: TriG (rappresentata poco sotto) ed N-Quads.

```
# This document encodes one graph.
@prefix ex: <http://www.example.org/vocabulary#> .
@prefix : <http://www.example.org/exampleDocument#> .

:G1 { :Monica a ex:Person ;
            ex:name "Monica Murphy" ;
            ex:homepage <http://www.monicamurphy.org> ;
            ex:email <mailto:monica@monicamurphy.org> ;
            ex:hasSkill ex:Management ,
                        ex:Programming . }
```

[22]

Citiamo in conclusione un'altra sintassi molto importante, che è di fatto un'estensione di JSON a RDF: JSON-LD; questa rappresenta sicuramente un importante strumento di scambio dati nel Web moderno, grazie alla compatibilità con la struttura di REST API e database non strutturati come MongoDB.

1.3 SPARQL

In questa sezione, andremo ad analizzare ulteriormente gli strumenti esistenti, legati al *Semantic Web*, ed in particolare la sua tecnologia principe: SPARQL.
SPARQL è un linguaggio di interrogazione per grafi RDF, che può essere utilizzato su diverse basi dati che storicizzano nativamente grafi RDF (o li gestiscono tramite middleware).
Nel tempo sono state sviluppate numerose soluzioni che implementano il linguaggio SPARQL, che vanno a integrarsi nativamente con diversi linguaggi: quella scelta per lo sviluppo di *Conjector*, ovvero l'applicazione alla base di questa dissertazione, è stata *Quadstore*, che supporta nativamente SPARQL e che andremo ad analizzare successivamente.
La sintassi di SPARQL è molto semplice e riconducibile ad altre sintassi per l'interrogazione di basi dati (in particolare SQL):

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
     <http://purl.org/dc/elements/1.1/title> ?title .
```

```
}
```

[18]

SPARQL rappresenta una soluzione molto duttile per elaborare gli artefatti di una base dati e creare proiezioni, rappresentando quindi un *agent* in grado di collassare alla realtà gli artefatti, sulla base dei comandi dell'utente.
Per fare un esempio, il costrutto MINUS, risulterebbe particolarmente utile nel caso in cui volessi escludere dalla mia interrogazione specifici contesti.

```
PREFIX : <http://example/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?s
WHERE {
   ?s ?p ?o .
   MINUS {
      ?s foaf:givenName "Bob" .
   }
}
```

[18]

Rimane tuttavia il problema di come rappresentare correttamente strutture complesse, come quelle descritte in precedente, con il framework RDF.

1.4 Vocabolari RDF importanti

In questa sezione, analizzeremo i più importanti vocabolari RDF: come detto in precedenza, il Web Semantico, evolve appoggiandosi a una serie di vocabolari controllati,

che permettono di esprimere concetti in maniera inequivocabile anche per un *agent*. Il primo è sicuramente RDF, ovvero il vocabolario di base di RDF, che contiene alcune delle ontologie maggiormente usate nell'espressione di triple RDF.
RDF include al suo interno ontologie per descrivere classi e proprietà, tra cui, a titolo di esempio, possiamo elencare:

- **rdf:type** - utilizzata come predicato per collegare una risorsa a una particolare classe;
- **rdf:value** - una proprietà usata per assegnare valori a un soggetto;
- **rdf:subject** - usata come predicato per assegnare un soggetto a un IRI;
- **rdf:predicate** - usata come predicato per assegnare un predicato a un IRI;
- **rdf:object** - usata come predicato per assegnare un oggetto a un IRI.

Un altro importante vocabolario è sicuramente OWL, che viene utilizzato per la creazione di ontologie.
Un vocabolario, invece, dalla storia interessante è Dublin Core: considerato il nucleo delle meta-informazioni interessanti per descrivere qualunque risorsa digitale.
DC nasce da una iniziativa nata durante una conferenza avvenuta nell'omonima cittadina dell'Ohio, a cui parteciparono bibliotecari, archivisti, editori, ricercatori e sviluppatori di software, oltre ad alcuni membri dai gruppi di lavoro dell'IETF nel 1995.
Alcuni esempi delle ontologie proposte da Dublin Core sono:

- **dc:created** - usato per assegnare una data di creazione ad una risorsa;
- **dc:creator** - usato per assegnare un creatore ad una risorsa;
- **dc:subject** - usato per descrivere il soggetto trattato da una risorsa;
- **dc:title** - usato per assegnare un titolo a una risorsa.

Un altro importante progetto, nell'ambito dei vocabolari del *Semantic Web* è FOAF (friend of a friend).
FOAF è stato ideato come "progetto dedicato a collegare persone e informazioni utilizzando il Web [...] integrando tre tipi di rete: reti sociali di collaborazione umana, amicizia e associazione" [14].
Considerando la sua natura è evidente come sia rilevante ed abbia un importante potenziale oggi, che il tessuto *Social* si è così tanto sviluppato.
FOAF suddivide i termini del suo vocabolario, tra quelli che hanno senso esclusivamente nel contesto del Web e quelli che hanno un'applicabilità universale, quando si parla di collegamenti tra persone e informazioni.
Alcuni esempi, tra i più importanti, delle ontologie FOAF sono:

- **foaf:knows** - che rappresenta un legame di conoscenza tra soggetto e oggetto;
- **foaf:Person** - ovvero la classe che rappresenta le persone;
- **foaf:name** - che associa a un IRI un nome.

1.5 Rappresentare strutture complesse in RDF

1.5.1 Reificazioni

Quanto descritto sino ad ora fa intuire il potenziale a cui un agente potrebbe attingere, per valutare in maniera ancora più intelligente i contenuti semantici del Web.
Per comprendere meglio ciò, possiamo fare un esercizio mentale ed immaginare le difficoltà di discernimento e selezione delle informazioni sul web: spesso ci vengono proposti articoli datati, oppure forniti da fonti che vorremmo in qualche modo filtrare.
Possiamo concordare tutti sul fatto che avere la possibilità di richiedere a un agente di estrapolare informazioni maggiormente focalizzate sulle nostre esigenze, sarebbe più che auspicabile, così come estrarre informazioni da una base dati, filtrandole non solo per

soggetto, ma anche focalizzandosi sui contesti più adeguati.
Il Web Semantico mette quindi a disposizione tutti gli strumenti necessari per ampliare il potenziale dei metadati, garantendo strutture adeguate a gestirne l'evoluzione nel tempo e la confrontabilità in ambito di incertezza delle informazioni.
Tutto ciò, trova però alcune limitazioni nella rappresentazione di strutture complesse: in particolare, sorgono evidenti difficoltà nel momento in cui si volesse arricchire un artefatto di reificazioni, ovvero meta-affermazioni su di esso.
Il processo di reificazione, prevede che ad una risorsa venga assegnato un IRI, per poter trattare quest'ultima come un oggetto ed esprimere dunque asserzioni ulteriori su di esso.
Le sintassi principali di RDF, soffrono del grande limite di non poter esprimere reificazioni in maniera gerarchica.
Se volessi esprimere un concetto come quello visto in precedenza (Cicerone è stato console nel 63 a.C.), potrei farlo esclusivamente su un livello paritetico dal punto di vista di un interprete RDF ed il risultato sarebbe qualcosa di simile a questo:

```
:Cicerone a foaf:Person .
:console a schema:Role .

:_s rdf:Type rdf:Statement ;
rdf:subject :Cicerone ;
rdf:predicate schema:hasOccupation ;
rdf:object :console .

:_c rdf:type ctx:Context ;
ctx:forStatement :_s ;
ctx:validFor "-0063"^^xsd:gYear .
```

La struttura mostrata sopra, presenta un rilevante problema dal punto di vista di un interprete: l'artefatto ed il contesto si trovano allo stesso livello; conseguenza di questo è che non esiste alcun modo per creare un'associazione gerarchica, in grado di limitare la validità dell'asserzione al contesto.
Questo limite è legato all'impossibilità di esprimere relazioni ternarie con le classiche strutture RDF.

1.5.2 Relazioni n-arie

Per garantire una strutturazione delle meta-informazioni gerarchica, bisogna passare dalla rappresentazione degli artefatti tramite triple a una struttura maggiormente complessa.
Questa struttura dovrebbe essere in grado di creare un collegamento tra l'artefatto ed il contesto, per fare in modo che l'*agent* sia in grado di anteporre quest'ultimo al resto del contenuto e valutare se *collassarlo alla realtà*, ovvero rappresentarlo come vero, contestuale, attendibile.
Ad oggi esiste una soluzione potenzialmente sfruttabile per creare una struttura simile: i grafi.

1.5.3 Grafi

RDF è un "modello dati per la rappresentazione dei contenuti web in un formato machine readable"[16], con la sua versione 1.1 ha introdotto un'importante novità: la possibilità di esprimere asserzioni complesse attraverso i grafi.
La W3C Recommendation del 25 febbraio 2014 definisce RDF 1.1 una sintassi astratta, come base del linguaggio e delle specifiche per i grafi annidati (non presenti in RDF 1.0), lasciando la struttura base invariata, ovvero composta de triple nella forma di soggetto-

predicato-oggetto, con tre tipi di nodi: IRI, letterali e nodi anonimi.
Appunto dall'uso dei grafi, prendono spunto Barabucci, Tommasi e Vitali che, nel loro lavoro "Supporting complexity and conjectures in cultural heritage descriptions", suggeriscono di sfruttare questa estensione per associare un contesto alle triple: essi infatti sottolineano come, fisiologicamente, RDF non sia orientato a preservare differenti punti di vista o diverse asserzioni espresse o riferite ad archi temporali differenti, deficitando quindi di una capacità di storicizzazione e versionamento delle informazioni.
Grazie a queste nuove strutture è ora possibile collegare in maniera gerarchica un artefatto con le sue meta-informazioni, creando a tutti gli effetti una relazione n-aria utilizzabile da un interprete moderno.
Nonostante tutto, esiste ancora un problema rilevante: per rappresentare in maniera sofisticata una relazione n-aria gerarchica, nel caso in cui le meta-informazioni siano molteplici, è necessario utilizzare grafi annidati; ad oggi, non esiste una completa standardizzazione di questi e i moderni interpreti RDF non supportano questa architettura.

1.5.4 Provenance

Esiste un'ulteriore rilevante tematica, legata ai contesti, ovvero la *provenance* delle asserzioni: nel mondo delle congetture, che rappresenta l'opportunità forse principale che garantirebbe l'utilizzo dei contesti, la provenienza di un'asserzione assume un ruolo rilevante.
Se volessimo conservare una serie di artefatti, contenenti opinioni anche contrastanti su un soggetto, diventerebbe fondamentale stabilirne la provenienza, in modo da renderla evidente e usufruibile per un *agent*.
Tra i vocabolari RDF, esistono soluzioni che propongono ontologie per esprimere la provenienza di un'asserzione: in particolare PROV-O "Fornisce una serie di classi, proprietà e restrizioni che possono essere utilizzate per rappresentare e scambiare informazioni sulla

provenienza generate in diversi sistemi e in diversi contesti" [24]. A titolo esemplificativo, alcune delle ontologie proposte sono:

- **prov:wasGeneratedBy** - usata per collegare il soggetto con l'attività che l'ha generato.
- **prov:wasAttributedTo** - usata per ascrivere un'entità a un *agent*.
- **prov:wasRevisionOf** - usata per descrivere un'entità derivata da un'altra.

La presenza di queste ontologie, risolve solo parzialmente il problema, perché ancora una volta non permette ad un *agent* di determinare l'idiosincrasia dell'artefatto in maniera gerarchica.

Capitolo 2

Congetture e contesti

2.1 Il problema

Sino a questo punto abbiamo evidenziato con chiarezza le difficoltà del *Semantic Web* nel rappresentare strutture complesse in maniera adeguata.
Non possiamo, semplicemente linearizzando le asserzioni attraverso triple soggetto-predicato-oggetto, esprimere in maniera adeguata concetti rilevanti per l'utilizzo dei grafi RDF.
L'impossibilità di gestire in maniera efficace la contestualizzazione di un'asserzione, pone dei limiti all'utilizzo che possiamo fare del *Semantic Web* e, tanto più lavoriamo in ambiti di incertezza ed evoluzione, tanto più questa cosa diventa rilevante.
Pensiamo all'ambito della ricerca, o degli studi in generale: cosa succederebbe se volessimo rappresentare un simposio in maniera evoluta, ovvero in formato digitale, sfruttando gli strumenti messi a disposizione del *Semantic Web* come tali?
Abbiamo bisogno di un sistema molto più evoluto, che ci permetta di associare opioni a contesti e provenienza, per poter discriminare e al tempo stesso preservare questa ricchezza informativa.
Il problema che si cerca di superare, è quello di avere informazioni sull'artefatto e sul contesto poste allo stesso livello da un agente, che non sarebbe quindi in grado di *col-*

lassare alla realtà le asserzioni: la soluzione diviene dunque quella di utilizzare dei grafi annidati, che permettano di anteporre una "guardia", legata al contesto, alle informazioni espresse in RDF; questo permetterebbe a un agente di discriminare, sulla base di una selezione, le informazioni corrette per l'elaborazione della richiesta.
Nelle prossime sezioni andremo ad analizzare la proposta di Barabucci, Tomasi e Vitali, ovvero l'uso di congetture e contesti per esprimere concetti complessi nel Web Semantico.

2.2 Le congetture

L'idea alla base della congettura è quella di "esprimere formalmente affermazioni accademiche, in attesa della valutazione del contesto" [5]. Infatti le congetture non acquisiscono la qualità di asserzione in senso forte, sino a quando il loro contesto non viene valutato e di conseguenza *collassato alla realtà*: parliamo di un'operazione attraverso il quale si trasforma il contenuto protetto da una *guardia* in un equivalente comune grafo RDF.
La sintassi proposta da Barabucci, Tomasi e Vitali a questo scopo è la seguente:

```
Cj (G) {{
S1 P1 O1 .
S2 P2 O2 .
}}
```

[5]
Il grafo rappresentato qui sopra, presenta una guardia (G) anteposta alla congettura, che grantisce all'*agent* la possibilità di decidere se accettarne il contenuto o meno, dopo aver valutato quest'ultima.
Le congetture presentate, possono essere ricondotte ad un contesto, detto anche "grafo

dal contesto indebolito", sostituendo ai predicati dei corrispondenti predicati con guardia, che vengono eventualmente collassati alla realtà dall'interprete.

```
Ctx {
S1 PG1 O1 .
S2 PG2 O2 .
PG1 isConjecturallyEquivalentTo P1 .
PG2 isConjecturallyEquivalentTo P2 .
}
```

[5]
Quello che vedremo nella sessione successiva è l'uso ipotetico del contesto così creato e le tecniche per collassarlo alla realtà.

2.3 I contesti

Nella sessione precedente abbiamo introdotto i contesti come una soluzione tecnica all'uso delle congetture, garantendo una compatibilità con gli attuali interpreti che, come già detto, non supportano i grafi annidati.
Come sottolineato nell'articolo, non è possibile usare strumenti RDF standard per leggere e manipolare i grafi congetturali: quello che viene dunque proposto è di aggiungere programmaticamente delle triple, che "trasformino i predicati con guardia da congetture (asserzioni deboli) in fatti (affermazioni forti)" [5].
Di seguito un esempio preso direttamente dall'articolo, che chiarisce quanto espresso sin qui:

```
:collapseOfT a conj:Collapse ; prov:wasAttributedTo :GBarabucci ;
conj:affects :T .
:T prov:wasAttributedTo :FTomasi
```

```
:T {
:GBPiranesi T:creator :AntichitàRomane .
T:creator conj:isConjecturallyEquivalent dc:creator .
:collapseOfT {
T:creator own:sameAs dc:creator .
}
}
:V prov:wasAttributedTo :FVitali
:V {
:JWayne V:creator :AntichitàRomane .
V:creator conj:isConjecturallyEquivalent dc:creator .
}
```

[5]

2.4 Alcuni esempi

A questo punto, se volessimo riscrivere la nostra asserzione su Cicerone collassata alla realtà, secondo quanto detto sino ad ora, il risultato sarebbe qualcosa di simile:

```
:collapseOfT a conj:Collapse ;
conj:affects :T .

:T ctx:validFor "-0063"^^xsd:gYear .

:T {
:Cicerone T:hasOccupation :console .
:collapseOfT {
```

```
T:hasOccupation own:sameAs schema:hasOccupation .
}
}
```

L'approccio adottato per questo lavoro, però, è ancora diverso rispetto a quanto proposto: lavorando con oggetti JavaScript che rappresentano grafi RDF ed un sistema di parsing, interpretazione e serializzazione che lavorano con la stessa struttura, è stato utile sfruttare le caratteristiche di questo modello.
Infatti, quanto proposto da Barabucci, Tomasi e Vitali, non si adatta facilmente agli strumenti ed alle librerie esistenti per la manipolazione dei grafi RDF in JavaScript ed esistono approcci tecnici alternativi, ma ugualmente efficaci.
Come vedremo nel capitolo su Conjector, l'applicazione alla base di questo lavoro, utilizzeremo la skolemizzazione dei grafi, per attribuire il contesto in maniera gerarchica, lasciando poi all'applicazione l'elaborazione delle asserzioni discriminandone la provenienza.

Capitolo 3

Conjector

3.1 Scopo di Conjector

Questo capitolo fornirà una panoramica approfondita sulla struttura di Conjector che, come già accennato è l'applicazione alla base di questa dissertazione e che si pone come obiettivo, quello di fornire un ambiente per la gestione di triple RDF associando ad esse un contesto.
Lo scopo di Conjector, infatti, è quello di sperimentare l'applicabilità dei contesti in un caso d'uso reale, adattando quanto teorizzato alle funzionalità di linguaggi di programmazione (in questo caso JavaScript) e alle librerie disponibili per la manipolazione dei grafi RDF.
Conjector, come applicazione, è stata pensata per presentare le principali caratteristiche, anche in termini di interfaccia, di un'applicazione moderna che permettesse l'inserimento di triple RDF, principalmente in due delle sintassi principali (Turtle ed N-Triples), dando inoltre la possibilità di interrogare direttamente le asserzioni esistenti a sistema tramite query SPARQL.
La limitazione all'inserimento di triple, piuttosto che grafi (come da standard RDF 1.1) è stata guidata da ragioni squisitamente tecniche; infatti la gestione del contesto avviene,

come detto in precedenza, inserendo un suo riferimento nel grafo della quadrupla: gestire questo paradigma con i grafi avrebbe complicato lo sviluppo in maniera significativa e, per questa prima versione, si è preferito optare per un modello semplificato, lasciando spazio ad estensioni future.

L'applicazione mette a disposizione alcuni esempi di default per testare l'inserimento di triple, tramite le sintassi gestite, e per interrogare alcuni esempi presenti di default a sistema.

L'applicazione prevede un sistema di login, la cui funzionalità va oltre quella esclusivamente legata alla sicurezza: la sessione dell'utente viene infatti gestita per generare il contesto da associare alla tripla inserita dall'utente; questa funzionalità risulta particolarmente utile per non permettere un inserimento forzato, ma gestito esclusivamente dal sistema.

Oltre al nome dell'utente che inserisce le asserzioni a sistema, viene gestita anche la data di inserimento dell'asserzione, tuttavia, all'inserimento di una tripla riconducibile a una precedente, confrontando soggetto e predicato, questa viene sostituita dalla nuova con il nuovo contesto, così da permettere all'utente di aggiornare i suoi inserimenti.

Come detto in precedenza, il sistema si avvale di un modello basato sulla skolemizzazione dei nodi anonimi, per permettere una persistenza del legame tra IRI di questo tipo, anche al di fuori del contesto dell'asserzione: questo permette fondamentalmente di gestire contesto e asserzione su livelli separati, permettendo poi di collassare quest'ultima alla realtà, considerando le informazioni legate al IRI del grafo esterno.

Rimane interessante notare come le funzionalità descritte sino a qui, generino un ambiente idealmente in grado di analizzare e manipolare liberamente i risultati di interrogazioni sul sistema, basandosi sui contesti delle quadruple inserite.

3.2 Usare Conjector

Quanto segue è una breve panoramica dell'applicazione, vista dal punto di vista dell'utilizzatore, accompagnata da una breve raccolta di immagini rappresentanti le pagine principali, che può essere utilizzata anche come guida all'utilizzo.
Dal punto di vista dell'interfaccia utente, l'applicazione è stata pensata come una *single-page application* basata sul framework React combinato con la libreria di componenti Material-UI, con lo scopo di costruire una struttura moderna, dinamica e dal design basato sulle linee guida Material.
Di conseguenza, tutte le operazioni avvengono tramite caricamento dinamico dei contenuti e utilizzo del pattern AJAX per la comunicazione con l'applicazione server.
Accedendo all'applicazione per la prima volta, viene presentata, in maniera un po' scolastica, un form di registrazione: come già detto l'autenticazione dell'utente va oltre le ordinarie esigenze di sicurezza, perché viene sfruttata per vincolare le triple al contesto, generando questa dimensione in maniera trasparente e soprattutto avulsa dall'interazione dell'utente.
Eseguito l'accesso, viene caricato dinamicamente un *text box* (figura 3.1), per permettere l'inserimento di asserzioni da parte dell'utente che ha la possibilità di farle parserizzare e serializzare dal server: in caso di successo nell'interpretazione, viene restituito un vettore, contenente l'elenco dei soggetti riconosciuti.
Il sistema, a questo punto, comunica nuovamente con il server, per effettuare una ricerca basata sulle informazioni ricevute.

Nel menu sono inoltre disponibili alcuni esempi, basati sulle principali sintassi accettate dal programma, che una volta selezionate, vengono automaticamente inserite (figura 3.2).
Quello che risulta dalle operazioni precedenti (figura 3.3) è un elenco, estratto da sistema, delle triple collegate ai soggetti inseriti, interpretato in linguaggio naturale, grazie

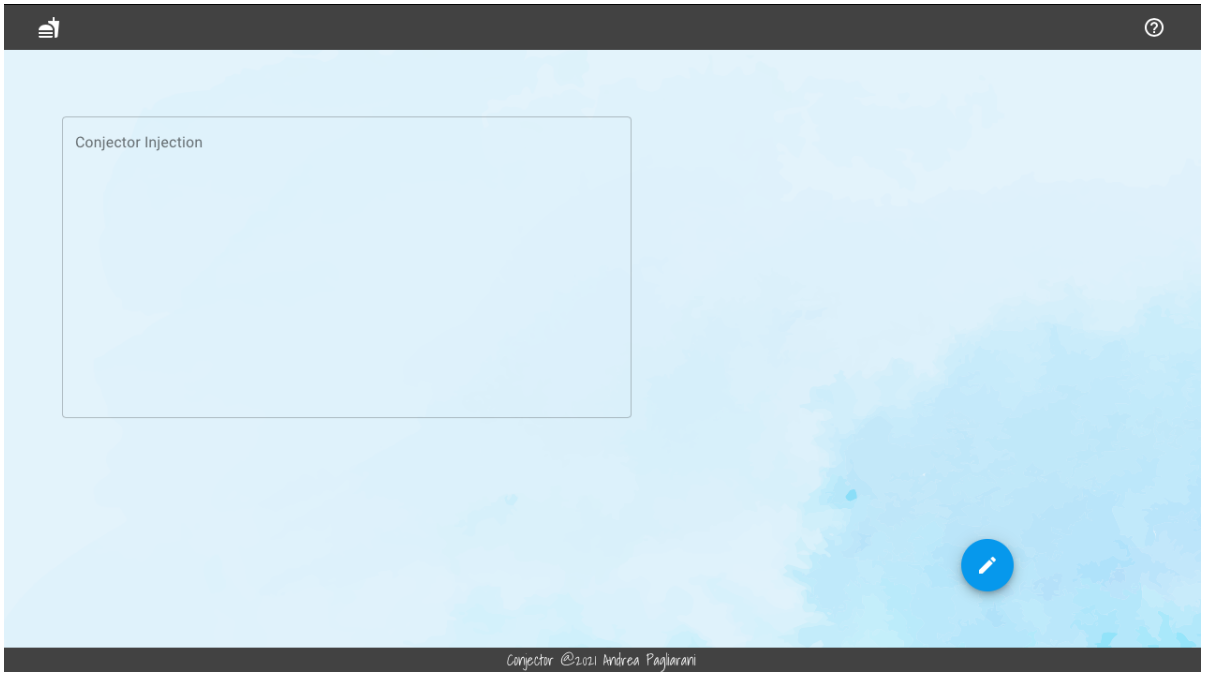

Figura 3.1: Text box per l'inserimento delle triple

Figura 3.2: Inserimento delle triple di esempio, espresse in Turtle, su Alexandre Dumas

Figura 3.3: Risultato del parsing delle triple espresse in Turtle e della ricerca per soggetto

alla struttura di template inserita sotto forma di ontologia con prefisso *ppp*.
I risultati vengono annidati sotto un elemento collassato della lista che riporta la data e l'autore dell'asserzione, per aggregarli in base al contesto: questo restituisce un'idea astratta del potenziale della soluzione nel discriminare i contesti.

La lista dei risultati, oltre a presentare un'interpretazione delle asserzioni, è collegata anche ad un vettore di grafi RDF, che viene restituito dalla API: cliccando su una di queste, l'applicazione sostituisce il contenuto della text box con la relativa tripla espressa in Turtle (figura 3.4).
L'utente ha così la possibilità di modificare quanto già da lui inserito a sistema o creare una nuova asserzione a partire da quella di qualcun altro.

Figura 3.4: Selezione di una delle triple tra i risultati di ricerca

Il sistema mette a disposizione anche esempi di asserzioni espresse in N-Triples e di interrogazioni tramite SPARQL, su dati presenti a sistema di default.

Il sistema di inserimento triple, mette a disposizione dell'utente anche informazioni in merito al corretto inserimento della tripla e supporta l'utente con informazioni in merito al funzionamento del sistema, richiamabili tramite il pulsante di *help* che è chiaramente individuabile nella *nav bar*.

3.3 Conjector e vocabolari importanti

Per maggiore fruibilità dell'applicazione, sono state inserite alcune delle ontologie più diffuse, con relativo template; di queste, il cui elenco esaustivo è visionabile direttamente nell'applicazione, è stato inoltre incluso il prefisso di default nelle asserzioni e sono: Dublin Core, FOAF, OWL, RDF ed RDFS.
Questo anche per dare la possibilità al sistema di avere una capacità interpretativa di base di quanto inserito e mostrare in un linguaggio intelligibile le triple inserite a sistema.
Dall'applicazione, sono visualizzabili le triple inserite a sistema in fase di inizializzazione, unici casi privi di contesto, che fanno riferimento ai principali prefissi RDF (figura 3.5).

3.4 Conjector, congetture e contesti

Nonostante Conjector non metta a disposizione un reale strumento per collassare alla realtà le triple, evidenzia come questo sia di fatto gestibile con le soluzioni adottate.
Il fatto di suddividere le asserzioni per autore e data, mostra come il sistema sia di fatto in grado di discriminare la provenienza, così come potrebbe fare lo stesso con altre in-

Figura 3.5: Triple inserite a sistema in fase di inizializzazione

formazioni che potrebbero arricchire il grafo in maniera illimitata.

Sviluppando alcune estensioni, Conjector sarebbe in grado di estrapolare informazioni, il cui contesto rispetti quanto richiesto dall'*agent*, ricalcando quindi, almeno parzialmente, quanto proposto da Barabucci, Tomasi e Vitali, anche se con una soluzione tecnica leggermente differente.

Capitolo 4

L'Architettura di Conjector

4.1 Introduzione alle tecnologie di Conjector

In questa sezione andremo a introdurre quelle che sono le tecnologie disponibili oggi per la manipolazione di grafi RDF, con particolare focus su quelle utilizzate nello sviluppo di Conjector e del loro contributo alla gestione dei contesti.
Pensando allo *stack tecnologico* correlato al *Semantic Web*, potrebbe essere utile creare una categorizzazione in 3 "strati", per l'intelligibilità di questa parte e per rendere evidente la separazione, di fatto necessaria anche a livello architetturale:

1. I framework e i modelli dati, che per quanto tecnologie non nel senso stretto, ne guidano comunque lo sviluppo;

2. Le tecnologie *proprie* del Semantic Web, ovvero tutte quelle che nascono esclusivamente per lo scopo di lavorare con RDF;

3. Le tecnologie *generiche*, ovvero quelle che non sono state create esclusivamente per questo contesto, ma mettono a disposizione o godono della presenza di framework in grado di supportare lo sviluppo orientato al Semantic Web.

Partendo dall'analisi del primo "strato", è sicuramente importante sottolineare come il "nuovo" framework RDF 1.1 abbia di fatto reso possibile la teorizzazione della gestione dei contesti, grazie all'introduzione dei grafi, come estensione delle triple.
Ad un livello tecnico, per ottenere questo risultato, si è poi introdotto un importante pattern, ovvero quello della skolemizzazione (*skolemisation*) dei nodi anonimi, come descritto nella W3C proposal `https://www.w3.org/2011/rdf-wg/wiki/Skolemisation`: questa risulta particolarmente utile, considerando che per i contesti verranno utilizzati dei nodi anonimi, per rendere questi ultimi indicizzabili anche al di fuori del grafo a cui appartengono.
Un altro importante elemento dei modelli dati sono i prefissi: sebbene l'uso di prefissi sia incluso nella struttura proposta da RDF, esiste una serie molto ampia di ontologie standard che possono essere incluse nelle triple RDF per esprimere concetti in maniera inequivocabile (considerando che rappresentano di fatto dei vocabolari controllati).
L'uso dei prefissi è parte integrante della costruzione dei grafi RDF e in questo lavoro si è scelto di includerne alcuni dei principali, precostituendo a priori un template per la loro interpretazione.
Sono stati inoltre introdotti due prefissi, con le relative ontologie, utili agli scopi sopra descritti:

- **ppp** per la raccolta dei template;
- **ctx** per la gestione dei contesti.

Considerando che, come vedremo successivamente, il progetto basa la sua architettura principalmente sull'utilizzo di JavaScript come linguaggio, è sicuramente importante introdurre anche una specifica comunitaria del *RDF JavaScript Libraries Community Group*, che definisce e standardizza il modello dati per la rappresentazione dei grafi RDF in classi JavaScript e può essere riassunto nel diagramma di classe dell'immagine sottostante (figura 4.1).

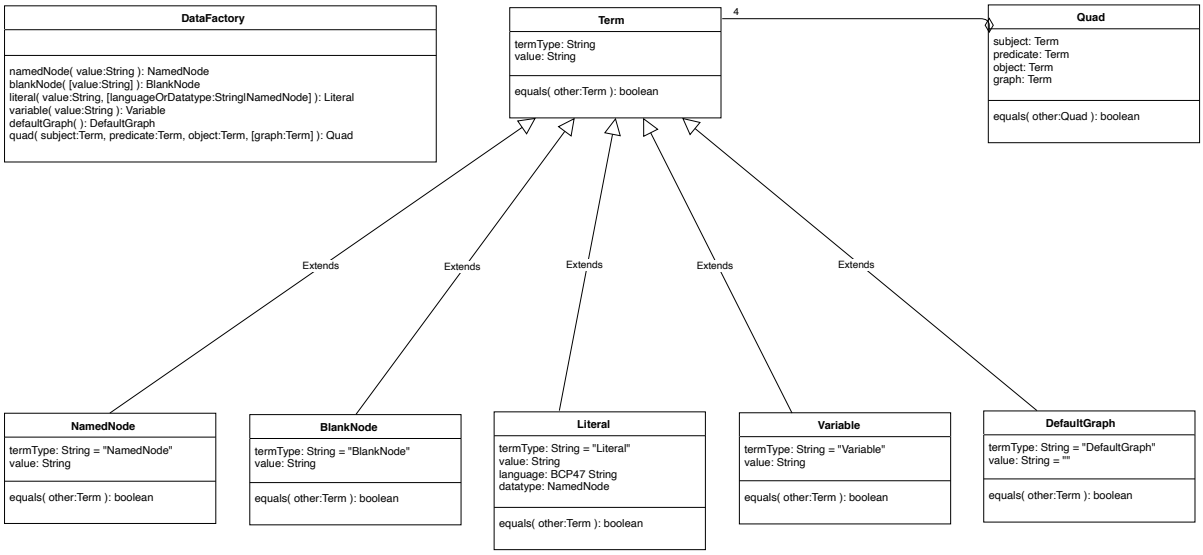

Figura 4.1: Diagramma di classe per il modello dati RDF/JS

Parlando infine dell'ultimo strato, ovvero quello delle tecnologie *generiche*, la tematica potrebbe ampliarsi oltre lo scopo di questa dissertazione, quindi il focus si manterrà esclusivamente sulle tecnologie coinvolte nello sviluppo dell'applicazione, che sono esclusivamente librerie JavaScript, ovvero Node.js.
Diviene doveroso fare una piccola premessa su come JavaScript abbia affermato, in particolare negli ultimi anni, la sua posizione di *linguaggio del Web* e come, pertanto le tecnologie scelte siano esclusivamente librerie sviluppate per questo linguaggio.
In particolare è stato utilizzato Quadstore, una libreria per la serializzazione di grafi RDF istanziati in classi che rispettano lo standard RDF/JS, per la gestione della persistenza: Quadstore si basa sulla versione per Node.js di LevelDB, una base dati NoSQL strutturata secondo il modello chiave - valore, per gestire la persistenza dei dati e mette a disposizione un motore basato su Comunica, meta query engine SPARQL, per permettere l'interrogazione della base dati tramite query SPARQL.
L'altra importante libreria utilizzata è stata graphy.js per il parsing delle triple Turtle/N-Trig e la costruzione di oggetti RDF/JS compliant: graphy.js mette a disposizione una serie di funzionalità per l'interpretazione di triple scritte nei principali formati sintattici per RDF e un'utile data factory per oggetti RDF/JS.

4.2 L'architettura in dettaglio

Descriveremo ora l'architettura di Conjector in dettaglio, andando ad analizzare le principali tecnologie che la compongono, soffermandoci sui dettagli più rilevanti della sua implementazione.
Come già accennato, l'applicazione è stata sviluppata principalmente basandosi su uno stack ispirato al modello *JavaScript Everywhere*.
La parte front-end è basata sul framework di Facebook React e su una sua libreria di

componenti grafici, basati sul design Material, Material-UI: questa scelta ha agevolato lo sviluppo di un'applicazione dinamica, basata dunque sul caricamento dinamico dei contenuti, seguendo l'interazione dell'utente.
Tralasciando dettagli non utili allo scopo della dissertazione, l'interfaccia grafica è stata sviluppata suddividendo logicamente le classi dei componenti, assieme ai loro metodi, dalle funzioni per la manipolazione del DOM e quelle per le operazioni AJAX; questo approccio ha permesso di mantenere lo sviluppo ordinato, e le componenti dell'applicazione separate sulla base della loro funzione.
Seppure la parte di front-end presenti altri dettagli interessanti, non risulta utile allo scopo della dissertazione l'analisi di questi ultimi, perché decisamente fuori contesto e poco significativi.
Lato back-end troviamo, per la parte applicativa, principalmente Node.js ed Express, abbinati ad altre librerie di supporto, di cui le più rilevanti sono:

- **Quadstore**, che, come già detto, è stata utilizzata per gestire la persistenza delle quadruple RDF;
- **graphy.js**, che è stato utilizzato come DataFactory e parser RDF;
- **Level** un wrapper di LevelDB per Node.js, che è stato utilizzato per la persistenza delle informazioni degli utenti.

Partendo dal livello più basso dell'applicazione, ovvero quello che si occupa della persistenza, vale la pena fare una breve digressione su LevelDB e su Quadstore, in quanto si tratta di strumenti decisamente interessanti, che permettono la gestione di una base dati NoSQL, senza dover istanziare nulla al di fuori dell'applicazione Node stessa.
LevelDB è un database NoSQL sviluppato in C++ da Jeffrey Dean e Sanjay Ghemawat per Google: basa il suo funzionamento sulla creazione di array di byte composti da una chiave e un valore collegati.
Quadstore è una libreria JavaScript, sviluppata per gestire la serializzazione di oggetti

RDF/JS, e lavora appunto, basandosi su un'interfaccia per basi dati chiave-valore.
Conjector mette a disposizione due classi, con i relativi metodi, per la gestione della persistenza di grafi RDF e dati degli utenti: la classe *GraphyConnector*, per i primi, e la classe *UsersConnector*, per gli ultimi; entrambe mettono a disposizione metodi per operazioni CRUD sui dati e sfruttano le librerie Quadstore e Level.
Una scelta implementativa rilevante, considerando che l'applicazione è stata sviluppata utilizzando principalmente un linguaggio, multi-paradigma, che ha come punto di forza la gestione asincrona dei processi, riguarda appunto la gestione dei flussi.
Molte funzioni sono state create sulla base di *promesse* e spesso, per una maggior eleganza del codice, si è sfruttato il costrutto *async - await*: uno zucchero sintattico che permette di gestire il flusso di operazioni, di una funzione asincrona, gestendo la *race condition* sulle variabili.
Un esempio significativo di questo costrutto, può essere trovato nelle funzioni usate per le operazioni sul quad store:

```
read: async function(match) {
            let { items } = await store.get(match)
                .catch((err) => {
                    console.log(err);
                });
            return items;
        }
```

Per la gestione dei grafi RDF, si è pensato di rappresentare questo oggetto di dominio con una classe ad hoc (la classe *Graphy*) che mette a disposizione una serie di funzioni utili alla manipolazione degli oggetti RDF/JS, di cui andremo ad analizzare alcuni di quelli più significativi.
Tra le funzioni principali per la gestione delle triple, quelle di interpretazione dei grafi

sono alla base del parsing delle asserzioni e della costruzione degli oggetti: questi ultimi vengono mantenuti in un array all'interno dell'oggetto stesso.
Di seguito è possibile vedere quella per l'interpretazione di Turtle, che sfrutta la libreria *graphy.js*: in questo caso conj è la variabile a cui, in fase di inizializzazione dell'oggetto, viene assegnata la stringa di testo contenente le asserzioni; da notare che vengono concatenati automaticamente i prefissi noti.

```
translateTurtle: function() {
            return new Promise((resolve, reject) => {
                ttl_read(ttl_prefixes + conj, {
                    data(y_quad) {
                        graphyObj.push(y_quad);
                    },
                    eof() {
                        resolve('done');
                    },
                    error() {
                        reject('BadSyntax');
                    }
                });
            });
        }
```

Un'altra importante funzione, in parte fulcro di questo lavoro, esposta dalla classe *Graphy*, è quella per la contestualizzazione delle asserzioni: come detto in precedenza, viene sfruttata la skolemizzazione dei nodi anonimi; nel nostro caso viene istanziato subito come IRI, in sostituzione del grafo di default, che nasce vuoto a causa della sua natura derivata da una tripla.

```
setGraph: function(context) {
            graphyObj.forEach((e) => {
                let graphUnique = uuidv4();
                e.graph = factory.namedNode(graphUnique);

                store.serialize(factory.quad(...[
                    factory.namedNode(graphUnique),
                    factory.namedNode('/ctx#assertedBy'),
                    factory.namedNode(`${ context }`)
                    ])
                );

                const toDate = new Date();

                store.serialize(factory.quad(...[
                    factory.namedNode(graphUnique),
                    factory.namedNode('/ctx#assertionDate'),
                    factory.dateTime(toDate)
                    ])
                );
            });
        }
```

Gli oggetti *Graphy*, per la traduzione delle triple in linguaggio naturale, sfruttano una classe di servizio per l'interpretazione degli oggetti RDF/JS: *Interpreter*.
Questa classe si occupa di ricercare tra le ontologie *ppp* un template relazionato al predicato delle asserzioni e, laddove possibile, restituisce una stringa di senso compiuto.

Per compiere questa operazione, nel caso in cui soggetto ed oggetto non siano dei letterali, ricerca un'etichetta (*rdf:Label*) adeguata nello store e costruisce una frase sostituendo questa negli opportuni placeholder ${subject} e ${object} del template.
Risalendo l'architettura dell'applicazione server troviamo la classe *GraphyService*, che può essere vista come un *Façade Controller* per tutte le operazioni sui grafi: essa si occupa di gestire le istanze di *Graphy*, di richiamarne i metodi sulla base delle richieste fatte al sistema e di interagire con il quad store, per tutte le operazioni *GET* che arrivano al sistema.
Al di sopra di queste troviamo, infine, i controller per la gestione delle *REST API* e il cosiddetto *boilerplate code* comune, o comunque simile, a tante applicazioni Node.js (quindi di scarso interesse per lo scopo di questa dissertazione).
Come ultima riflessione sull'architettura, pensando a un utilizzo in un contesto reale, l'applicazione presenta diversi punti di inefficienza su cui vale la pena riflettere per il miglioramento della stessa: ad oggi il parsing delle triple avviene *server side* e potrebbe certamente essere replicato lato client, per alleggerire il carico di lavoro del back-end, che potrebbe occuparsi solo delle operazioni *CRUD* e della costruzione delle frasi in linguaggio naturale.

Conclusioni

Questo lavoro ha rappresentato certamente una buona occasione per analizzare l'utilizzo dei contesti, applicandoli a un caso reale: la contestualizzazione delle asserzioni, rappresenta certamente una possibilità importante per la gestione delle opinioni nel web semantico.

A mio avviso questo apre la strada a tante possibilità, tra cui, come detto da Barbucci, Tomasi e Vitali, quella di preservare le opinioni all'interno di artefatti afferenti a un contesto scientifico o culturale; non solo, penso che potremmo facilmente immaginare un web in cui il contenuto delle pagine web viene arricchito da opinioni (e non solo informazioni) sui soggetti trattati, dando la possibilità al *agent* di collassare alla realtà le asserzioni sulla base del contesto.

A mio parere questo lavoro può essere visto, almeno concettualmente, come una possibile estensione agli odierni prodotti del Web Semantico: molti di questi gestiscono il versionamento delle informazioni (come per esempio fa MediaWiki), ma ad oggi non consentono di inserire all'interno di un artefatto, differenti opinioni.

Questo sarebbe particolarmente utile per garantire la possibilità non tanto di modificare il contenuto, mantenendo un'unica realtà, ma di gestire congetture espresse in ambito di incertezza, mantenendo tutte le informazioni all'interno dell'artefatto (e non in un archivio).

L'utilizzabilità dei contesti, non si limita al preservare opinioni o informazioni circostanziali, ma potrebbe essere impiegata anche per la valutazione dei contenuti.

Infatti, una delle estensioni possibili di questo lavoro, potrebbe essere quella di integrare un sistema di *rating* delle triple, che permetterebbe di collassare alla realtà asserzioni sulla base del gradimento. Il Web nasce certamente come un mondo libero (o quasi), ereditando da questo aggettivo pregi e difetti: l'impatto della disinformazione derivante da contenuti inattendibili o addirittura manipolati è stato oggetto di discussione in diversi ambienti; potrebbe essere interessante, con riferimento a questo, sperimentare un arricchimento dei contenuti che permetta di valutare l'attendibilità di asserzioni inerenti un soggetto.

Una pecca della soluzione proposta, è quasi certamente legata all'utilizzo del pattern di skolemizzazione dei e separazione del contesto dal grafo stesso.

Questa soluzione genera un evidente overhead e impedisce di eseguire ricerche basate già sul contesto desiderato, costringendo a un esame esteso a tutte le asserzioni esistenti sul soggetto.

Non è probabilmente evidente in un sistema ridotto e sviluppato con scopi dimostrativi, ma considerando una base dati più ampia, potrebbe diventare un problema gestire tante elaborazioni di questo tipo.

Inoltre genera anche un incremento significativo della memoria utilizzata, anche se lineare, perché ad ogni tripla, viene di fatto associato un grafo collegato ad un corrispondente grafo che ne rappresenta il contesto.

Nonostante queste considerazioni, rimane ugualmente una soluzione probabilmente interessante, che meriterebbe di essere sperimentata in un contesto operativo, per comprenderne il reale valore e le effettive capacità.

Ulteriore estensione al sistema di *rating*, potrebbe essere dunque quella di sviluppare una estensione per browser, che permetta di parserizzare il contenuto delle pagine, arricchendo i soggetti di informazioni consultabili e scelte dal *agent* sulla base di preferenze in termini di affidabilità e contesto.

Sarebbe infine certamente interessante, tentare di utilizzare algoritmi di *Natural Langua-*

ge Processing, per costruire triple partendo dal linguaggio naturale con cui popolare lo store: risulta certamente un limite dover conoscere e utilizzare un framework RDF per inserire asserzioni, limitando l'uso dell'applicazione a esperti.

Ringraziamenti

Vorrei ringraziare il Professor Vitali per avermi guidato in questo percorso e per le importanti e mai banali nozioni che ha trasmesso durante le sue lezioni.
Un ringraziamento speciale va a Vincenzo Rubano, che mi ha mostrato come leggere il Web in modo diverso e sarebbe stato impossibile da immaginare senza il suo aiuto.
Vorrei ringraziare i miei genitori, che come sempre mi hanno supportato in questo percorso, anche ricordandomi che "devo uscire quando ci sono le belle giornate perché il Sole fa bene alla salute".
Un grazie speciale va ad Elisa, la mia compagna, che più che supportato sarebbe giusto dire mi ha sopportato.
Vorrei ringraziare chi ha creduto in me, ma soprattutto chi non l'ha fatto, chi ha detto che quel che stavo facendo era inutile e chi mi ha detto che non avrei dovuto farlo, perché mi hanno spinto ad insistere ancora di più.

Bibliografia

[1] Stefano Triberti ed Eleonora Brivio 2016. User Experience - Psicologia degli oggetti, degli utenti e dei contesti d'uso. Maggioli Editore

[2] Tim Berners-Lee, James Hendler e Ora Lassila 2001. The Semantic Web. Scientific American

[3] Tim Berners-Lee 1997. Metadata Architecture. W3C DesignIssues. `https://www.w3.org/DesignIssues/Metadata.html`

[4] W3C 2014. Resource Description Framework (RDF). W3C Recommendation. `https://www.w3.org/2001/sw/wiki/RDF`

[5] Gioele Barabucci, Francesca Tomasi e Fabio Vitali. Supporting complexity and conjectures in cultural heritage descriptions.

[6] International Standard ISO-2788. Documentation - Guidelines for the development of monolingual thesauri. Second edition. 1986-11-15

[7] Chris Taylor. An Introduction to Metadata. `http://www.library.uq.edu.au/iad/ctmeta4.html`

[8] Serafina Spinelli. Introduzione all'indicizzazione. `http://mail.biocfarm.unibo.it/~spinelli/indicizzazione/`

[9] Serafina Spinelli. Introduzione ai thesauri. http://mail.biocfarm.unibo.it/~spinelli/indicizzazione/thesauri.htm

[10] MARC21. http://www.loc.gov/marc/bibliographic/ecbdhome.html/

[11] PREMIS Data Dictionary for Preservation Metadata. http://www.loc.gov/standards/premis/

[12] IFLA Study Group on the Functional Requirements for Bibliographic Records 1997. Functional requirements for bibliographic records : final report. Munich: K.G. Saur Verlag. https://www.ifla.org/files/assets/cataloguing/frbr/frbr_2008.pdf

[13] DMCI. Dublin Core Metadata Initiative. 05/11/2007. http://www.dublincore.org/

[14] FOAF - The Friend of a Friend project. http://www.foaf-project.org/

[15] SKOS - Simple Knowledge Organisation System. http://www.w3.org/2004/02/skos/

[16] W3C 2014. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation. https://www.w3.org/TR/rdf11-concepts/

[17] W3C 2008. SPARQL Query Language for RDF. W3C Recommendation. https://www.w3.org/TR/rdf-sparql-query/

[18] W3C 2013. SPARQL 1.1 Query Language. W3C Recommendation. https://www.w3.org/TR/sparql11-query/

[19] RDF JavaScript Libraries Community Group. RDF/JS: Data model specification. Draft Community Group Report. https://rdf.js.org/data-model-spec/

[20] W3C 2014. RDF 1.1 Turtle. W3C Recommendation. https://www.w3.org/TR/turtle/

[21] W3C 2014. RDF 1.1 N-Triples. W3C Recommendation. https://www.w3.org/TR/n-triples/

[22] W3C 2014. RDF 1.1 TriG. W3C Recommendation. https://www.w3.org/TR/trig/

[23] W3C 2014. RDF 1.1 XML Syntax. W3C Recommendation. https://www.w3.org/TR/rdf-syntax-grammar/

[24] W3C 2013. PROV-O: The PROV Ontology. W3C Recommendation. https://www.w3.org/TR/prov-o/#bib-PROV-DM

Elenco delle figure