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

Sistema di tag binarie per la creazione di relazioni in un Wiki semantico

Relatore: Prof. Marco Colombetti
Correlatore: Ing. David Laniado

Tesi di Laurea di:
Graziano Graziosi, matricola 669390

Anno Accademico 2007-2008

Indice

Capitolo 1

Introduzione

Sono passati ben 17 anni da quando Tim Berners-Lee pubblicò sul newsgroup alt.hypertext una breve descrizione del suo progetto, il *World Wide Web*. Di lì a poco, il WWW sarebbe stato reso disponibile al pubblico; da allora, niente è stato più lo stesso, portando il Web ad essere uno dei pilastri nell'ambito dell'informazione e della comunicazione.

Col passare degli anni il Web è cambiato, si è evoluto, arrivando a influenzare diversi ambienti; basti pensare alla sua attuale importanza nell'ambito dell'economia, delle relazioni sociali, della diffusione dell'informazione. L'ultimo capitolo della storia del Web è rappresentato dal boom del **Web 2.0** . Una possibile definizione di Web 2.0 potrebbe essere quella di:

> a knowledge-oriented environment where human interactions generate contents that are published, managed and used through network applications in a service-oriented architecture.
>
> [de Judicibus, 2008]

Rispetto al "Web 1.0" e alla sua staticità, si passa a una concezione di Web più dinamica e che annovera tra le sue caratteristiche fondamentali quella di comprendere nel suo processo di evoluzione il contributo diretto degli utenti. E' proprio la concezione di partecipazione attiva nel processo di creazione e gestione dell'informazione una delle novità più significative: mentre in passato per creare grandi database ci si affidava a lavoratori pagati, o a volontari, con le applicazioni web 2.0 sono proprio gli utenti a creare i contenuti [Bricklin, 2000]: in un certo senso, possiamo dire che le applicazioni associate col Web 2.0 sfruttino l'intelligenza collettiva [O'Reilly, 2004]. Si

passa così dai siti personali ai blogs, dai sistemi di gestione dei contenuti ai *wikis*, dalle directories alle folksonomies.

Tuttavia, in molti pensano che nonostante i grandi cambiamenti apportati dalla rivoluzione del Web 2.0, quest'ultimo non rappresenti ancora l'evoluzione del World Wide Web, poichè, nonostante apporti diverse novità, non porta ancora a una nuova concezione di Web, che rimane quella di sempre, ossia "connettere persone"[Laningham, 2006]. Il prossimo passo da compiere, secondo il creatore stesso del WWW, sembra essere il *Web Semantico*: passare da una concezione di Web in cui le informazioni contenute sono comprensibili da umani ma restano semplice testo per le macchine, a un'altra in cui le macchine riescano a "capire" i dati, le loro descrizioni e le relazioni che intercorrono tra di essi, riuscendo a dedurre nuove informazioni basandosi su quelle che già possiede. In questo lavoro ci poniamo in un'ottica di congiunzione tra queste due realtà: sfruttare la potenza dei sistemi partecipativi propri del Web 2.0 per creare informazione che sia non solo *human readable* ma anche *machine readable*. Per questo ci occuperemo di un campo attualmente in via di sviluppo, quello dei *wiki semantici*, ossia dei wiki aventi come base un'ontologia semantica. Il principale problema in questo frangente è quello di coniugare l'ambiente mutevole e instabile dei wiki (dove l'informazione si evolve e cambia costantemente) con la rigidità della logica alla base delle ontologie. Tenteremo di coniugare questi due aspetti nell'ambito non tanto di definire gli oggetti quanto in quello di descrivere delle *relazioni* tra di essi.

1.0.1 Struttura della tesi

Nel **Capitolo 2** ci occuperemo inizialmente di dare una definizione di Web Semantico e di descrivere il modo in cui le ontologie vengono implementate, dando una panoramica sulle tecnologie esistenti (RDF, RDF-S, OWL) e soffermandoci in particolare sulle definizioni di relazioni tra risorse. In seguito, spiegheremo sinteticamente cosa sono i Wiki e quali sono le loro peculiarità, passando subito dopo a parlare della loro variante semantica. Infine, definiremo i concetti di tag, di tagging collaborativo e di tag binaria, che formeranno la substruttura per la creazione di relazioni nel wiki e che saranno fondamentali per capire come raggiungere gli obiettivi che ci poniamo.

Nel **Capitolo 3** saranno analizzati gli obiettivi da raggiungere e le problematiche che incontreremo durante la progettazione. Quindi, verrà spiegato

il modo in cui si intende realizzare il sistema, passando dalla gestione e l'inserimento delle tag binarie da parte degli utenti alla creazione effettiva di relazioni nell'ontologia.

Nel **Capitolo 4** verrà descritto il modo in cui ciò che è stato ideato in fase di progettazione viene effettivamente implementato: verranno descritti brevemente i tools utilizzati per la creazione del wiki, l'indexing delle tag binarie, la gestione dell'ontologia; l'effettiva implementazione delle varie parti del sistema; le interfacce utente create per interagire con il wiki e con l'ontologia; le scelte implementative e gli algoritmi per la generazione dei suggerimenti, il tutto fornendo esempi che permettano allo stesso tempo di vedere e di capire come funziona il sistema.

Infine, nel **Capitolo 6** , verranno tirate le somme e ipotizzati i possibili sviluppi di questo software.

Capitolo 2

Stato dell'arte

2.1 Il Web Semantico

> *I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web - the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize.*
>
> *Tim Berners Lee*
> *[Berners-Lee and Fischetti, 1999]*

Il concetto di Web Semantico fu sviluppato dal creatore del World Wide Web, Tim Berners-Lee: rientrava nella sua visione del Web come di un mezzo di scambio di dati, informazioni e conoscenza. La base del Semantic Web è la capacità di scrivere dati non solo in linguaggio naturale, ma anche in formato che sia "comprensibile" e processabile da una macchina, permettendo di gestire e di dedurre informazioni, e di integrare dati provenienti da fonti diverse in maniera efficiente. Per arrivare a questo risultato, è fondamentale le risorse, e le relazioni intercorrenti tra di esse. Le prime possono essere sia risorse web (come pagine web, immagini, documenti) che risorse fisiche non accessibili in rete (persone, luoghi, oggetti ecc.) o concetti astratti. Tutte queste risorse saranno identificate univocamente in rete da un URI (*Uniform Resource Identifier*).

2.1.1 Definire risorse e relazioni: RDF

Per raggiungere quest'obiettivo, nel Semantic Web i dati vengono scritti in **RDF**, un linguaggio formale basato su XML che permette di descrivere metadati da associare alle risorse in modo che essi possano essere "capiti" dalla macchina. L'assunto di RDF è semplice: le informazioni possono essere scritte tramite asserzioni composte dalla tripla soggetto-predicato-oggetto [Klyne and Carroll, 2004].

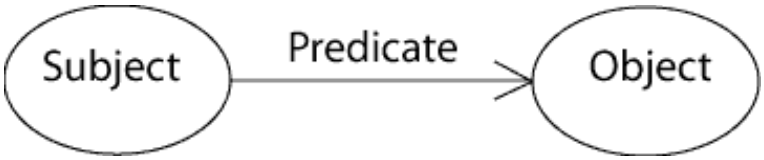

Figura 2.1: La struttura di una tripla RDF.

Un esempio di tripla potrebbe essere quella della Tabella 2.1

	Contenuto
Soggetto	Frank Zappa
Predicato	is member of
Oggetto	Mothers of Invention

Tabella 2.1: Esempio di tripla RDF

in cui ogni risorsa sarà identificata univocamente da un URI, come ad esempio:
Frank Zappa: *http://musicbrainz.org/show/artist/?artistid=2101*
is member of: *http://en.wiktionary.org/wiki/member*
Mothers of Invention: *http://musicbrainz.org/show/artist/?artistid=342367*

L'asserzione di una tripla RDF sta a significare che vi è una certa relazione, indicata dal predicato, tra due risorse indicate da soggetto e oggetto della tripla. L'oggetto della tripla può anche essere un dato primitivo, come numeri, stringhe, date, e così via, che non devono essere necessariamente identificati da un URI; ciò non può accadere per quanto riguarda soggetto e predicato.

Da quanto esposto è intuibile come RDF sia uno strumento potente per definire attributi e relazioni tra risorse. Tuttavia pone diverse limitazioni, in

quanto non permette, ad esempio, di descrivere le risorse, o le relazioni che le legano. Come si vede in Figura 2.2 RDF non è che la base; andremo ad analizzare alcuni degli "strati" superiori.

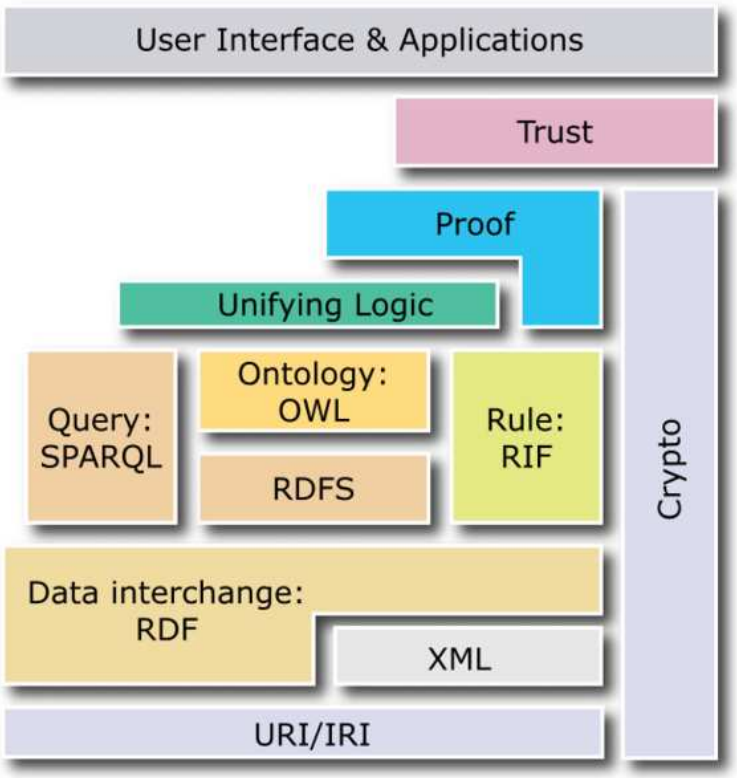

Figura 2.2: Semantic Web Layer Cake

2.1.2 Definire classi e relazioni: RDF-Schema

RDF-Schema è un'*estensione semantica* di RDF, ossia aggiunge a RDF dei costrutti semantici, in modo da estendere le sue potenzialità. L'obiettivo di RDF-Schema è quello di fornire meccanismi per descrivere gruppi di risorse e le relazioni che intercorrono tra tali risorse [Brickley and Guha, 2004].

Classi

RDF-Schema permette di dividere le risorse in gruppi chiamati *classi* e categorizzarle tramite l'attributo **rdf:type** . Le classi sono a loro volta risorse

identificate da un URI. I membri di un determinato gruppo identificato da una classe sono chiamati *istanze* o *individui* di quella classe.

Un costrutto fondamentale fornito da RDF-Schema è quello che permette di definire le gerarchie: se una classe C è sottoclasse di un'altra classe C', allora, definendo con C(a) l'espressione "a è un'istanza della classe C", si ha

$$\forall x(C(x) \Rightarrow C'(x))$$

ossia che tutti gli individui appartenenti alla classe C appartengono anche alla classe C'. In RDF-Schema tale relazione tra classi si definisce tramite il costrutto **rdfs:subClassOf** .

Proprietà

Una proprietà (*property*) è una *relazione* che lega soggetto e oggetto, assimilabile al predicato che avevamo citato in precedenza. In RDF-Schema viene introdotto il concetto di sottoproprietà: se una property P è sottoproprietà di un'altra relazione P', allora, definendo con P(a,b) l'espressione "a in relazione P con b", si ha

$$\forall x \forall y(P(x,y) \Rightarrow P'(x,y))$$

ossia se una relazione P che coinvolge due individui x e y è sottoproprietà di P', allora x e y saranno in relazione P' tra di loro.

Altri concetti fondamentali sono quelli relativi a dominio (*domain*) e codominio (*range*). Il **dominio** è l'attributo di una property che definisce i valori per cui è definita. Se abbiamo un'asserzione del tipo

```
P rdfs:domain C
```

vogliamo intendere che tutti i soggetti delle triple il cui predicato è P hanno classe C, ossia che

$$\forall x \forall y(P(x,y) \Rightarrow C(x))$$

Il **codominio** invece agisce sull'oggetto dell'asserzione: se

```
P rdfs:range C
```

allora tutti gli oggetti delle triple il cui predicato è P hanno classe C:

$$\forall x \forall y(P(x,y) \Rightarrow C(y))$$

Grazie a RDF-Schema siamo in grado quindi di definire classi e proprietà di risorse RDF e di costruire gerarchie. Il prossimo passo sarà quello di avere uno strumento, avente come base RDF e RDF-Schema, che ci permetta di definire *ontologie*.

2.1.3 Creazione di ontologie: OWL

Il linguaggio **OWL** (Web Ontology Language) estende semanticamente quanto visto finora aggiungendo nuovi costrutti per la descrizione di classi e proprietà [van Harmelen Deborah L. McGuinness, 2004]. OWL si divide in 3 sottolinguaggi:

- *OWL Lite* , fornisce i mezzi per costruire gerarchie con semplici vincoli;
- *OWL DL* (la versione che useremo in questo lavoro) fornisce tutti i costrutti disponibili in OWL da usare però con alcune restrizioni. La loro potenza espressiva è pari a quella delle *Description logics*, un sottoinsieme *decidibile* (ossia: tutte le computazioni terminano in un tempo finito) della FOL, la logica dei predicati del primo ordine, che è solo semidecidibile.
- *OWL Full*, che ha un'espressività maggiore della FOL ma non è decidibile.

In quanto non esplicitamente necessari al lavoro che stiamo svolgendo, trascureremo i costrutti introdotti da OWL DL per la definizioni di classi; ci concentreremo invece sugli attributi che è possibile assegnare alle proprietà. Oltre al costrutto SubPropertyOf, di cui si è già parlato, vi sono diverse nuove possibilità nel definire le *property*:

- **owl:EquivalentProperty**, indica che due proprietà hanno la stessa estensione; esse rappresentano lo stesso insieme di individui, ma potrebbero corrispondere a concetti diversi, quindi sono proprietà *equivalenti* e non *la stessa* proprietà. Date due proprietà P e P', se

  ```
  P' owl:equivalentClass P
  ```

  si ha che:

  $$\forall x \forall y (P(x,y) \Leftrightarrow P'(x,y))$$

- **owl:inverseOf**, permette di definire una relazione inversa tra due proprietà. Ad esempio, la proprietà inversa di *publisher of* sarà *published by*. Date quindi due proprietà P e P', se

  ```
  P' owl:inverseOf P
  ```

il dominio di P' sarà pari al codominio di P e il codominio di P' sarà pari al dominio di P.

- **owl:FunctionalProperty**, indica che la proprietà in questione è funzionale. Data una proprietà funzionale P si ha che:

$$\forall x \forall y \forall z (P(x,y) \wedge P(x,z) \Rightarrow y = z)$$

ossia ad ogni elemento del dominio corrisponde *al più* un elemento del codominio.

- **owl:InverseFunctionalProperty**, indica che la proprietà in questione è inversamente funzionale. Data una proprietà inversamente funzionale P si ha che:

$$\forall x \forall y \forall z (P(x,z) \wedge P(y,z) \Rightarrow x = y)$$

ossia ad ogni elemento del codominio corrisponde *al più* un elemento del dominio.

- **owl:SymmetricProperty**, indica che la relazione gode di proprietà simmetrica. Data una proprietà P, questa è simmetrica se:

$$\forall x \forall y (P(x,y) \Rightarrow P(y,x))$$

ossia, se x è in relazione P con y, anche y dev'essere in relazione P con x.

- **owl:SymmetricProperty**, indica che la relazione gode di proprietà transitiva. Data una proprietà P, questa è transitiva se:

$$\forall x \forall y \forall z (P(x,y) \wedge P(y,z) \Rightarrow P(x,z))$$

Grazie a tutti questi costrutti, siamo in grado di scrivere delle ontologie, ossia delle strutture dati relative a un dato dominio definite come l'insieme delle risorse, delle relazioni intercorrenti fra di esse, dei vincoli e delle regole che le riguardano: una descrizione coerente di una porzione di realtà. Grazie alla decidibilità delle ontologie scritte in OWL-DL, inoltre, la macchina può ricavare, tramite ragionamento di tipo *deduttivo*, nuove informazioni a partire da quelle già contenute nell'ontologia, grazie a software chiamati *reasoners*.

Abbiamo ora tutti gli elementi che ci servono. Il prossimo passo sarà definire *cos'è* un Wiki semantico; prima di tutto, però, bisogna sapere cos'è un *Wiki*.

2.2 Dai Wiki ai Wiki semantici

2.2.1 I Wiki: una breve panoramica

Cos'è un Wiki?

Un *Wiki* è un software che permette ai suoi utenti di creare, modificare e collegare (tramite hyperlink) pagine web in modo semplice e veloce. La sua peculiarità, e il suo punto di forza, è che i suoi utenti possono gestire le pagine e i rispettivi contenuti in modo *collaborativo*; proprio per questo motivo, i wiki sono una delle colonne portanti della rivoluzione del Web 2.0. Ad oggi, il più grande, conosciuto e usato wiki è *Wikipedia* [1], un'enciclopedia online redatta in oltre 250 lingue in modo collaborativo da volontari, e uno tra i 10 siti più visitati del mondo.

La prima definizione di wiki fu data dal suo creatore, Ward Cunningham che definì un wiki come *the simplest online database that could possibly work* [Leuf and Cunningham, 2001], sottolineando il fatto che un wiki potrebbe essere considerato come un grande database 'collettivo' che permette di creare, visualizzare e modificare informazioni. Cunningham creò il primo wiki, *WikiWikiWeb*, nel 1995 per migliorare le modalità di scambio di informazioni tra programmatori. Il nome wiki deriva dalla parola hawaiiana per 'veloce': Cunningham aveva sentito per la prima volta questo termine all'aeroporto di Honolulu, quando un impiegato gli aveva suggerito di prendere il 'Wiki Wiki' bus che collegava i vari terminali. Scelse questo nome come sostituto allitterativo di veloce, per evitare di chiamare la sua creazione *quick-web*.

Caratteristiche

La gestione di un wiki avviene fondalmente grazie alla possibilità di modificare con facilità le sue pagine, tramite un qualsiasi web browser. A seconda delle politiche adottate nei vari wiki, la modifica e creazione di pagine wiki può essere effettuata solo previa autenticazione, mentre in altri casi il permesso è accordato anche a utenti anonimi.

Le pagine del wiki sono scritte in un *linguaggio di markup* spesso chiamato *wikitext* o *wiki markup*. Generalmente si tratta di un'alternativa "semplificata" al linguaggio HTML, ritenuto troppo complesso per permettere modifiche veloci ai contenuti delle pagine. La sintassi del wikitext non è univoca, ma

[1] www.wikipedia.org

Figura 2.3: La home page di Wikipedia.

può variare da wiki a wiki; ognuno di essi comunque fornisce i comandi fondamentali per la formattazione del testo, permettendo di scrivere in corsivo, grassetto, di creare elenchi puntati o numerati, di inserire immagini, tabelle e soprattutto di collegare le varie pagine wiki tramite *hyperlink*. Si tratta un punto focale del funzionamento del wiki: la possibilità di linkare le pagine wiki fra di loro permette di navigare agevolmente e in maniera non lineare tra di loro e di creare la struttura del wiki "dal basso". Inoltre, è possibile definire link a pagine non esistenti: in questo caso, si viene indirizzati a una pagina di modifica che permette di creare i contenuti della pagina wiki di destinazione.

I vari cambiamenti apportati al wiki non sono mai definitivi: in genere ogni singola versione della pagina modificata viene immagazzinata, in modo da poter ripristinare i contenuti originali in caso di necessità.

Da quanto visto sinora è immediato notare quanto i wiki siano un mezzo potente per la diffusione e la creazione di informazione. Tuttavia, i link tra le pagine wiki non sono tipizzati, e in effetti i dati sono *human readable* ma non *machine readable*: e se fosse possibile per la macchina "capire" le relazioni che intercorrono tra le pagine, e le proprietà delle stesse, tramite l'inserimento di

informazioni aggiuntive processabili? E' da questa idea che nasce il concetto di *Wiki Semantico*.

2.2.2 Wiki semantici

Un *Wiki semantico* è un software che si propone di coniugare il rigore formale e i pregi del Semantic Web (dati processabili dalla macchina, sistemi di reasoning, creazione di queries complesse) con la potenza e la semplicità d'uso dei wiki. Sostanzialmente si tratta di un wiki, supportato da un modello di rappresentazione della conoscenza, che permette di associare alle sue pagine delle informazioni aggiuntive riguardanti sia le pagine stesse che le relazioni tra pagine.

Un esempio

Immaginando un wiki che abbia come dominio il mondo della musica, avremo ad esempio la pagina di Frank Zappa: in un wiki normale questa conterrebbe informazioni come semplice testo. In un wiki semantico, ad esempio, la pagina potrebbe definire in linguaggio formale che Frank Zappa è un musicista, potrebbe definire delle relazioni con le pagine relative agli album che ha scritto o prodotto, di che gruppi fa parte, e così via; esportando questi dati in RDF/OWL, sarebbe possibile ricavare in modo automatico nuove informazioni, come ad esempio le persone con cui ha suonato, le vendite totali dei suoi cd, che genere di musica preferisca suonare, artisti simili, e così via.

Caratteristiche

Il modello alla base dei wiki semantici è scritto in linguaggio formale, in modo da poter essere processato dalla macchina; in genere si tratta di RDF/OWL, o di applicativi software che estraggono la semantica dai dati contenuti in database relazionali. A seconda delle scelte progettuali la notazione formale può essere inclusa nel testo delle pagine stesse (come accade in Semantic MediaWiki), come metadata della pagina wiki, o può essere dedotta da informazioni come ad esempio il nome della pagina. L'ontologia alla base del wiki può essere fissa (come ad esempio nei wiki che si basano su *Cyc*[2]) limitando drasticamente i contributi degli utenti, oppure, come nel caso delle

[2] www.cyc.com

living ontologies, essa può essere più o meno modificata dagli utenti, sempre seguendo un insieme di regole che evitino il vandalismo e la perdita di coerenza dell'ontologia. Questo è un punto fondamentale da valutare nella creazioni di nuove relazioni in un'ontologia, e, come si vedrà più approfonditamente nel capitolo 3, proprio per giungere ad un compromesso tra rigore formale e partecipazione attiva degli utenti nella modifica dell'ontologia, si è giunti all'idea di permettere l'introduzione di relazioni tramite l'uso di *tag binarie*.

2.3 Un modo alternativo (e informale) di creare relazioni: le tag binarie

2.3.1 Tag: definizione

Sostanzialmente, una tag non è altro che un'etichetta, una *keyword* che viene applicata a un segmento d'informazione (che può essere di qualsiasi tipo, da una foto a un documento a una pagina wiki). Ovviamente le tag non pretendono di descrivere l'oggetto a cui sono allegate in maniera rigorosa o esaustiva: si rivelano però particolarmente utili nell'ambito della categorizzazione e della ricerca basata su parole chiave. Tornando all'esempio precedente, la pagina wiki su Frank Zappa potrebbe avere come tag le parole *musician*, *guitarist*, *rock*, *avantgarde*, e così via.

La possibilità offerta dalle tag di classificare l'informazione in modo veloce e intuitivo è uno dei motivi che le ha portate a essere uno dei capisaldi del Web2.0, in particolar modo nell'ambito delle *folksonomy* e del *collaborative tagging*.

2.3.2 Collaborative Tagging

Il *collaborative tagging* (conosciuto anche come folksonomy) consiste nella categorizzazione collaborativa dell'informazione tramite l'uso di tag. Mentre in precedenza la categorizzazione delle risorse veniva effettuata da esperti utilizzando un vocabolario controllato, con l'avvento delle folksonomy il ruolo attivo di creazione e gestione delle tag passa ai creatori e ai consumatori delle risorse taggate: così come per i wiki, le folksonomt vengono create dal basso, a partire dalla collaborazione tra utenti e in maniera "democratica", in modo da creare una struttura facile da consultare che rispecchi le idee e il vocabolario

dei fruitori stessi, senza mediazione da parte di esperti. Esempi ormai celebri di servizi web che fanno uso di collaborative tagging sono, ad esempio:

- **del.icio.us** , un sito di social bookmarking che permette di organizzare i propri bookmark taggando qualsiasi risorsa web sia provvista di un URL;
- **Flickr** , il più famoso e conosciuto sito per la condivisione di fotografie, che permette di fornire informazioni sulle immagini condivise tramite tag, in modo da rendere semplice la ricerca;
- **Last.fm** , una social community musicale che permette di taggare artisti, album e canzoni.
- **Technorati** , un motore di ricerca per weblog.

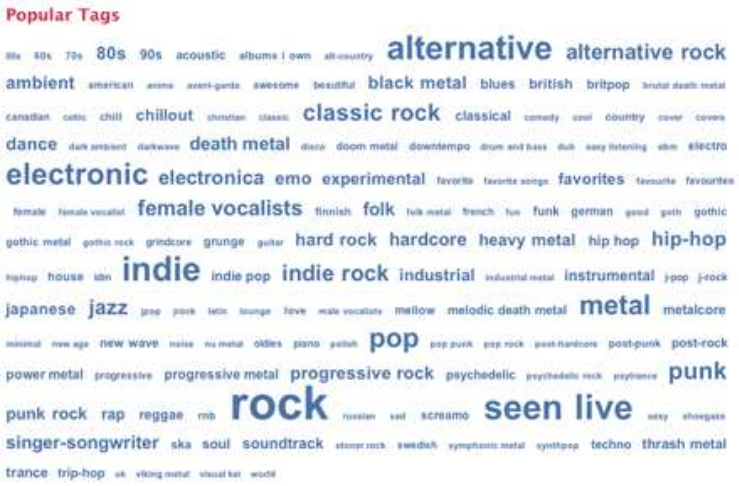

Figura 2.4: Una tagcloud di Last.fm

Una delle innovazioni più importanti a livello visivo introdotta dai sistemi di collaborative tagging (per esattezza da Flickr) è l'uso delle *tagcloud*: si tratta di rappresentazioni visive (nello specifico **liste pesate**) di insiemi di tag, generalmente ordinate per ordine alfabetico, che attribuiscono font più grandi alle tag più importanti o usate. In questo modo, l'utente può capire in modo intuitivo quali siano le tag più significative usate nel sistema ed accedere facilmente alle risorse ad esse collegate.

2.3.3 Tag per definire relazioni: le tag binarie

Finora abbiamo parlato di tag solo come mezzo per descrivere determinate risorse: e se invece esse venissero usate per definire *relazioni* tra risorse? Quando parliamo di tag binarie intendiamo proprio questo: delle tag che non identificano un'unica risorsa, ma servono a definire la relazione tra *due* risorse. Tecnicamente il principio è sempre lo stesso, ma in questo caso le risorse invece che immagini o pagine web sono *hyperlink* che collegano due risorse web. Questi link, all'apparenza normali collegamenti ipertestuali, presentano informazioni aggiuntive che descrivono la *natura* del link stesso e la relazione che c'è tra le risorse collegate.

Un primo utilizzo: link tipizzati

Questi particolari tipi di link furono inizialmente usati durante gli anni 80 in applicazioni ipertestuali come KMS, NoteCard e Ibis. In seguito, nella specifica degli hyperlink del linguaggio HTML furono inseriti due attributi, **rel** e **rev** , pensati per la definizione di informazioni aggiuntive sulla natura dei link. Il W3C ha definito dei valori standard per questi attributi che permettono di "tipizzare" i link e di definire delle relazioni elementari tra pagine web. Tali valori sono:

- alternate
- stylesheet
- start
- next
- prev
- contents
- index
- glossary
- copyright
- chapter

- section
- subsection
- appendix
- help
- bookmark

Come si può notare l'elenco è estremamente restrittivo e si occupa di definire delle relazioni di tipo "editoriale" più che concettuale, principalmente allo scopo di facilitare la navigazione all'interno di un sito web.

I microformat

I *microformat* sono un'estensione del linguaggio di markup HTML che permette espressioni semantiche usando gli attributi HTML class, rel, rev, in modo da poter estrarre informazioni utili dalle pagine web. Un esempio potrebbe essere la definizione delle generalità di un individuo tramite il microformat hCard:

```
<div class="vcard">
     <div class="fn">Joe Doe</div>
     <div class="org">The Example Company</div>
     <div class="tel">604-555-1234</div>
     <a class="url" href="http://example.com/">
     http://example.com/</a>
 </div>
```

Come si può notare il microformat fa uso di valori tipizzati dell'attributo 'class' per definire le varie componenti, come ad esempio fn (first name), org (organization), tel (telephone number). In questo modo non solo l'utente è in grado di leggere i dati, ma anche la macchina può recuperarli direttamente dalla pagina.

In genere i microformat che si occupano di definire relazioni tra pagine fanno uso dell'attributo **rel** proprio degli hyperlink: in particolare esistono alcuni microformat, come XFN, che si occupano di definire link "taggati".

Un esempio: XFN

XFN (*XHTML Friends Network*) è un microformat utilizzato per descrivere relazioni umane tramite l'uso di link. Viene usato spesso nei *blogroll* (ossia nella lista di link ai blog "preferiti" dall'autore del blog) per descrivere le relazioni intercorrenti tra l'autore e i proprietari dei blog linkati. Per definire queste relazioni basta inserire delle keyword sotto l'attributo rel del link. Ad esempio il link

```
<a href="http://jimmy.examples.com/" rel="colleague">
Jimmy Example</a>
```

indica che il proprietario del blog

`http://jimmy.examples.com/`

è collega del proprietario del blog in cui è situato il link.

Come si è visto finora, l'utilizzo di tag binarie non è molto diffuso. Tuttavia, nel nostro caso sono la scelta ideale per la definizione informale delle relazioni tra pagine di un wiki, in attesa che queste relazioni vengano esportate e formalizzate in RDF/OWL.

Capitolo 3

Obiettivi e scelte progettuali

3.1 Obiettivi e motivazioni

Nell'ambito di un wiki semantico la strategia da seguire per permettere all'ontologia di "evolversi" (o di aggiungere nuove classi o relazioni) non è fissa, ma deve venir valutata a seconda degli obiettivi che ci si prefigge per il wiki. E' proprio il fatto che la semantica abbia come substrato software un ambiente collaborativo in continua evoluzione come il wiki che crea maggior problemi: l'unione dello sforzo creativo di menti diverse per provenienza, cultura, opinioni può dar luogo a grandi risultati ma può anche creare molti problemi, specialmente in un ambiente come quello delle ontologie, caratterizzato da un grande rigore formale. L'obiettivo che ci poniamo con questo lavoro è quello di trovare un ideale compromesso tra la partecipazione attiva degli utenti del wiki alla creazione ed evoluzione dell'ontologia, e il mantenimento della coerenza fondamentale al funzionamento dell'ontologia stessa. Ci si propone di fornire agli utenti la possibilità di determinare attivamente come il wiki semantico si evolverà, mettendo però dei paletti (il meno restrittivi possibile) in modo da evitare il caos, e allo stesso tempo di fornire a chi tali paletti li gestisce (nel nostro caso, gli amministratori del wiki) di avere un tool integrato con il wiki che permetta loro di capire *cosa* aggiungere all'ontologia e *come* aggiungerla, basandosi proprio sui suggerimenti degli utenti.

Abbiamo deciso di focalizzare la nostra attenzione su un particolare e fondamentale aspetto dello sviluppo di un wiki semantico, ossia quello riguardante le relazioni tra individui dell'ontologia. Ci occuperemo quindi di sviluppare delle funzionalità aggiuntive che permettano sia di definire relazioni tra pagine wiki basate su proprietà già esistenti nell'ontologia, sia di *creare*

nuove property (sfruttando tutte le potenzialità fornite da RDF/OWL) nell'ontologia stessa, il tutto in maniera coerente con quanto sia già contenuto nella base di conoscenza. Trascureremo invece la parte riguardante la definizione di nuove classi e la modifica delle classi esistenti, e ci limiteremo a fornire (integrata all'editor del wiki) la possibilità di tipizzare le pagine wiki (funzionalità indispensabile per la definizione di relazioni tra le pagine).

3.2 La principale problematica: il campo d'azione degli utenti

3.2.1 Caso 1: gli utenti definiscono semplici relazioni RDF

Come già visto nel Capitolo 2, vi sono diversi gradi di interazione con la struttura semantica che vengono permessi agli utenti. Sul fronte più restrittivo troviamo i wiki che fanno uso di basi di conoscenza non modificabili dall'utente: in genere si tratta di ontologie già ben definite e vaste, come Cyc, e il contributo dell'utente è limitato a creare collegamenti concettuali tra le varie pagine, senza però creare *nuove* relazioni da aggiugere all'ontologia; si capisce come ciò limiti di molto l'evoluzione del wiki semantico.

Un altro caso può essere quello di **Platypus Wiki**[1], in cui ogni pagina è considerata una risorsa RDF e ogni hyperlink "etichettato" presente nel testo della pagina è tradotto in una tripla RDF [Tazzoli et al., 2004]: anche in questo caso l'utente può solamente agire sui collegamenti tra le pagine ma la struttura dell'ontologia rimane invariata. Lo stesso principio esteso in modo da comprendere anche RDF-Schema viene utilizzato in **Semantic MediaWiki** : l'utente può tipizzare le pagine (usando le *categories* già proprie di MediaWiki) e creare relazioni tra di esse (direttamente nelle pagine wiki tramite una semplice sintassi simile a quella per creare i normali links)[Krötzsch et al., 2006]. Sarebbe possibile applicare questi principi anche nel nostro caso?

[1] http://platypuswiki.sourceforge.net/

3.2.2 Caso 2: gli utenti definiscono relazioni e proprietà

Per capire se il metodo del Caso 1 rispecchi le nostre esigenze, dobbiamo prima di tutto valutare il contesto in cui ci muoviamo. Nel caso precedente l'utente era in grado di creare relazioni RDF; tuttavia nel nostro wiki ci muoviamo in un contesto più ampio ma allo stesso tempo più restrittivo, ossia OWL DL. Immaginiamo di lasciare totale libertà di movimento agli utenti: nel caso di un normale wiki ciò non ne minerebbe la struttura interna. Anche nei due casi precedenti comunque i danni sarebbero limitati, essendo la struttura un grafo RDF.

Nel nostro caso invece, proprio per il fatto di muoverci in ambiente OWL DL, un'immissione di dati errata, con conseguente modifica dell'ontologia, sia nella creazioni di relazioni che di proprietà apporterebbe gravi danni alla struttura di base del wiki semantico: la coerenza dell'ontologia potrebbe essere compromessa. Rispetto a Platypus Wiki o a Semantic MediaWiki noi introduciamo la possibilità di "tipizzare" le relazioni tra individui, ossia permettiamo di creare proprietà a cui poi quelle relazioni dovranno attenersi: introduciamo dei vincoli che l'utente dovrà necessariamente rispettare. Tuttavia per vari motivi questi vincoli potrebbero essere ignorati: l'utente potrebbe essere inesperto o incompetente e non conoscere adeguatamente la struttura di OWL, portando alla creazione di classi o proprietà concettualmente sbagliate o in contrasto con l'ontologia; potrebbe semplicemente non conoscere a fondo i concetti di cui si occupa; o addirittura inserire informazioni errate volontariamente, come avviene ad esempio col fenomeno del vandalismo in Wikipedia.

La coerenza dell'ontologia potrebbe essere compromessa in vadi modi: ad esempio poniamoci nel contesto della creazione di nuove relazioni. Per crearne una dovremmo attenerci ai vincoli imposti dalla corrispondente proprietà: ad esempio, dovremmo scegliere un soggetto e un oggetto della relazione che rispettino i vincoli di dominio e codominio. Se ignoriamo questi vincoli, i danni potrebbero essere gravi: d esempio, se un utente inserisse nella pagina del nostro wiki relativa a Frank Zappa l'asserzione "Frank Zappa suona Captain Beefheart" non solo l'affermazione sarebbe sbagliata, ma, nel caso di una procedura di reasoning, potrebbe portare il reasoner a stabilire che "Captain Beefheart è uno strumento musicale", con tutto ciò che ne consegue! Un altro esempio potrebbe essere il rispetto del vincolo di funzionalità di una proprietà. pensiamo al caso in cui (ammesso che una persona possa avere

una sola nazionalità) un utente asserisca che "Frank Zappa è americano", un'altro "Frank Zappa è inglese": si creerebbe una contraddizione all'interno dell'ontologia.

Se invece trattiamo le proprietà vere e proprie, i danni potrebbero essere ancora maggiori: ad esempio se un utente creasse una proprietà con attributi (ad esempio di dominio e codominio) sbagliati, e le nuove relazioni avrebbero un modello errato a cui attenersi. Se invece l'utente modificasse una proprietà già esistente in modo errato, potrebbe invalidare tutte le relazioni già presenti nell'ontologia.

Gli errori in questo caso sarebbe quindi molto più dannosi che in un normale wiki: bisognerebbe andare ad operare sul codice RDF/OWL per eliminare le inconsistenze, con conseguente perdita di efficienza. La soluzione ideale per chi scrive sarebbe quella di evitare situazioni del genere: si potrebbe *impedire* all'utente di andare a modificare direttamente l'ontologia, e lasciare che a farlo sia una figura competente, come ad esempio un amministratore del wiki.

3.2.3 Caso 3: l'utente suggerisce, l'amministratore formalizza

Arriviamo così alla soluzione proposta nel nuovo lavoro: l'utente si limiterà a "suggerire" le relazioni, mentre di formalizzarle e di creare nuove proprietà si occuperanno gli aministratori del wiki, o altre figure competenti. Inoltre, per non caricare eccessivamente di lavoro l'amministratore e per permettergli di conoscere con precisione la volontà dell'utenza, introduciamo una nuova entità: il wiki vero e proprio.

Quando si decide di "promuovere" una tag binaria creata da un utente a relazione, vengono raccolte informazioni su tutti i link in cui la stessa tag è stata usata. All'amministratore viene fornita un'interfaccia grafica integrata nel wiki, che gli permette di aggiungere direttamente la relazione all'ontologia (se esiste già una *property* con quel nome e i tipi di range e domain sono compatibili) o di crearne una ex novo; nel secondo caso, questa viene generata tramite una procedura guidata, e i dati raccolti dal software vengono processati in modo da generare *suggerimenti* per l'amministratore, riguardo vari fattori come domain e range da assegnare alla relazione, e le caratteristiche che essa debba avere. Una volta creata tale property, è possibile aggiungerla all'ontologia, insieme al tag binario da cui è stata generata. In questo modo

vi è sì un fattore di controllo da parte dell'amministratore, che decide quali link formalizzare ed esportare nell'ontologia; tuttavia, può esportare solo relazioni che non compromettano l'ontologia, e nel creare nuove proprietà è portato a considerare i suggerimenti di *tutta* la comunità, in quanto le informazioni da cui nascono i suggerimenti vengono recuperate da tutti i link in cui è stata usata quella particolare tag.

3.3 La creazione delle tag binarie

3.3.1 Inserimento

Come già detto, si è scelto di far inserire le tag binarie sotto forma di hyperlink direttamente nel testo delle pagine wiki: in questo modo, non vi è bisogno di interfacce aggiuntive per la creazione di tag binarie, e anche la modifica e l'eliminazione sono facilmente attuabili semplicemente editando la pagina. Altro punto a favore, è che, nel caso di modifiche improprie ai link taggati, si possa tornare alle versioni originali semplicemente *ripristinando* la versione precedente della pagina, proprio come si trattasse di semplice testo.

3.3.2 Ricerca

C'è però un problema: come recuperare i dati di cui abbiamo bisogno, se questi appaiono come testo nelle pagine wiki? Una soluzione potrebbe essere quella di scandire tutte le pagine del wiki quando vi è una richiesta di informazioni; tuttavia, si tratterebbe di una soluzione poco elegante e particolarmente dispendiosa, specialmente nei wiki di grandi dimensioni.

Si è optato per una soluzione alternativa: le pagine vengono scansionate solo in fase di startup da un apposito parser (che chiameremo *LinkParser*) che recupera tutti i link taggati presenti nelle pagine del wiki; in seguito questi vengono immagazzinati in un indice, su cui poi effettuare le ricerche. Man mano che il wiki viene modificato, l'indice viene modificato di conseguenza, aggiungendo e cancellando tag binarie a seconda che i link corrispondenti nelle pagine siano aggiunti o cancellati. Questa soluzione presenta diversi vantaggi: la fase di ricerca ora è molto più veloce; inoltre, in caso di corruzione dell'indice, i dati non vengono persi, perchè al riavvio del software questi vengono estratti dalle pagine stesse e l'indice viene ricostruito. D'altro canto, la fase di startup rimane dispendiosa in termini di risorse, specialmen-

te nel caso il wiki sia di dimensioni notevoli; si tratta comunque di un effetto collaterale trascurabile.

Per quanto riguarda la ricerca di tag binarie, si è scelto di permetterla sempre attraverso una particolare sintassi da inserire nel wikitext. In questo modo non si interferisce con la funzione di ricerca vera e propria e si possono inserire liste di relazioni nelle pagine, effettuando ricerche secondo vari parametri.

3.3.3 Tagcloud

Altra opzione riguardo le tag binarie è quella relativa alla creazione di tagcloud. Si è deciso di creare tagcloud che comunque forniscano informazioni sia rispetto all'uso globale di tag nel wiki, che rispetto all'uso per quanto riguarda una particolare pagina. Queste tagcloud potrebbero rivelarsi molto utili all'amministratore, ad esempio per capire quali siano le tag più usate e in un certo senso quali siano le più "urgenti" da formalizzare nell'ontologia.

3.4 Creazione di nuove relazioni RDF/OWL

Per quanto riguarda la creazione di nuove relazioni, si intende fornire all'amministratore un percorso guidato che gli permetta di effettuare scelte secondo il volere comune dell'utenza evitando comunque di compromettere la coerenza dell'ontologia.

3.4.1 Assegnazione dei tipi

Un requisito fondamentale per la creazione di nuove relazioni è il fatto che le pagine abbiano un tipo. Si è scelto di trattare le pagine come se corrispondessero tutte a *istanze* di determinate classi dell'ontologia (a scelta dell'amministratore). All'avvio del motore wiki, viene recuperata la lista di tutte le pagine del wiki; se ve n'è qualcuna che non figura nell'ontologia, essa viene associata a una nuova istanza di **owl:Thing** (la classe di default per tutti gli individui dell'ontologia). Se invece la pagina è già presente nell'ontologia, essa non viene modificata. In questo modo si è sicuri che l'ontologia copra *tutte* le pagine del wiki. Nel caso una pagina abbia lo stesso nome di una classe presente nell'ontologia, viene creata un'istanza dal nome *class*

+ *nome_classe* in modo da sottolineare il fatto che il contenuto della pagina non riguarda un particolare individuo ma un *gruppo* di individui. In un eventuale futuro sviluppo, su pagine con nomi di questo tipo si potrebbero compiere operazioni differenti, come la definizione e la modifica delle classi corrispondenti nell'ontologia. Per le pagine che invece rappresentano proprietà (ad esempio create a partire dai predicati delle tag binarie) l'istanza corrispondente nell'ontologia avrà un nome del tipo *property + nome_classe*. Queste distinzioni occorrono per evitare che vengano definite nell'ontologia relazioni tra classi o tra proprietà invece che tra istanze; se fosse possibile, usciremmo da OWL DL, perdendo la decidibilità. Quindi, sarà impedita la creazione di relazioni aventi come soggetto o oggetto pagine che rappresentino classi o proprietà dell'ontologia: ci limiteremo a definire relazioni tra istanze.

Un'altra approssimazione che bisogna adottare è quella riguardante la modifica del tipo di una pagina che possiede già relazioni formalizzate: in questo caso il nuovo tipo potrebbe non essere compatibile ad esempio con dominio e range delle relazioni già presenti nell'ontologia per quell'individuo: il rimedio (piuttosto drastico a dire il vero) che verrà utilizzato è quello di *cancellare* l'istanza relativa alla pagina in questione e tutte le relazioni che lo riguardano dall'ontologia, e crearne uno nuovo. Le relazioni potranno essere aggiunte nuovamente a partire dalle tag binarie che rimarrano invariate, sempre supponendo la compatibilità del nuovo tipo con quanto esisteva in precedenza. L'ambiente ideale per la scelta del tipo di una pagina potrebbe essere la pagina di edit.

3.4.2 Trasformare le tag binarie in nuove relazioni

Passiamo ora a uno dei punti focali dell'intero lavoro: la "promozione" di tag binarie a relazioni formalizzate nell'ontologia.

Prima di tutto l'amministratore deve *scegliere* il link da formalizzare. Bisogna creare quindi un menu con un elenco delle tag binarie descritte per pagina di provenienza, tag e pagina di destinazione; per evitare liste troppo lunghe o confusionarie, si è scelto di creare una tab apposita (chiamata *Manage TLinks*) per ogni pagina wiki in cui sono contenute tutte le tag binarie relative a quella pagina (quindi tutte quelle di cui la pagina è oggetto o soggetto). All'amministratore sarà possibile selezionare una di queste tag, e, sempre dalla stessa pagina, formalizzarla. Ovviamente, sarà possibile un'e-

sportazione in RDF/OWL "al volo" solo se esiste una proprietà nell'ontologia col nome uguale a quello della tag, e se dominio e range di quella relazione sono compatibili con i tipi delle pagine di origine e destinazione. Se tutti i requisiti sono soddisfatti, basterà un semplice click per ottenere il risultato desiderato. In caso contrario, invece, sarà necessario *creare una nuova proprietà* nell'ontologia che si adatti alla situazione in esame.

Creare una nuova OWL Property

Una nuova proprietà dell'ontologia potrà essere definita sia ex-novo, sia avendo come "modello" una particolare tag binaria selezionata dall'utente.

Nel primo caso ciò sarà possibile direttamente dalla pagina del predicato che vogliamo formalizzare: ad esempio, se avessimo delle tag binarie del tipo `x --> citizenOf --> y` e volessimo creare una nuova proprietà `citizenOf` senza avere come base una particolare tag binaria, dovremmo andare sulla pagina wiki *citizenOf*. La tab *Manage TLinks* (descritta in precedenza) di questa pagina mostrerà tutte le tag aventi quel particolare predicato; tuttavia non sarà possibile selezionarle: la proprietà quindi non avrà alcuna tag come "modello" su cui basarsi. Ovviamente il processo di creazione sarà possibile in questo primo caso solo se non vi è già una property nell'ontologia chiamata *citizenOf*.

Nell'altro caso invece la procedura è come quella descritta nel caso si volesse semplicemente formalizzare una semplice tag binaria: basta andare sulla pagina che ci interessa (che non rappresenterà stavolta un predicato di tag binaria ma un'istanza di classe), e selezionare sul tab *Manage TLinks* una tag dalla lista; questa sarà il modello su cui ci baseremo per creare la proprietà.

Per quanto riguarda la creazione della proprietà, potrebbero esservi vari modi di procedere. Nel caso avessimo una tag come modello, il più veloce e semplicistico sarebbe quello di creare una nuova proprietà col nome della tag, definire come dominio il tipo della pagina di origine e come codominio il tipo della pagina di destinazione. Tuttavia, il processo benchè fortemente automatizzato porterebbe spesso ad errori. Ad esempio, poniamo il caso di una tag binaria definita come in tabella 3.1 : Se utilizzassimo il metodo appena esposto, avremmo una property chiamata *citizenOf* con dominio in *Musician* e codominio in *Country*. E' comprensibile come tale proprietà sia stata formulata in modo sbagliato: il dominio, ad esempio, dovrebbe essere

Pagina d'origine	Frank Zappa
Tag	citizenOf
Pagina di destinazione	USA

Tabella 3.1: Esempio di binary tag

ragionevolmente di classe *Person*. Si presentano inoltre altri problemi: in primis, nel caso non avessimo una tag come base non avremmo elementi da cui partire. C'è da dire inoltre che all'amministratore dovrebbe essere lasciata l'ultima parola su quanto debba essere inserito nell'ontologia: non è detto che la macchina riesca a compiere scelte giuste, anche perchè a volte le informazioni date dagli utenti non sono attendibili. Inoltre, non ci è possibile utilizzare costrutti di OWL come gli attributi per relazioni funzionali, simmetriche, e così via. Per finire, mettiamo caso che la stessa tag citizenOf sia stata utilizzata in altre parti del wiki (sia nella stessa pagina che in altre): magari dalle altre tag sarebbe stato possibile ricavare altre informazioni che avrebbero permesso di risalire alla classe *Person* come dominio. In fondo considerare la sola binary tag selezionata come base per costruire una nuova proprietà significa ignorare l'opinione di tutti gli altri utenti che hanno utilizzato la stessa tag.

Un'altra soluzione, opposta alla prima, sarebbe lasciar prendere all'amministratore tutte le decisioni a sua discrezione. Con questa soluzione sarebbe possibile creare nuove proprietà senza un modello predefinito, tuttavia l'amministratore stesso potrebbe compiere errori, scegliendo in modo da compromettere l'ontologia esistente. Inoltre, i suggerimenti di tutti gli altri utenti verrebbero ignorati.

Per questo lavoro abbiamo deciso di giungere ad un compromesso che tenda inoltre ad eliminare anche la maggior parte dei problemi di cui soffrono le altre soluzioni: è sempre l'amministratore a scegliere come creare la nuova proprietà, ma

- gli viene impedita la creazione di proprietà che creino inconsistenze nell'ontologia;
- tra i valori concessi, gli vengono *suggeriti* quelli più probabili, basandosi sulle informazioni raccolte riguardo tutti i link etichettati con quella particolare tag nel wiki.

Per attuare il primo punto, basta fornire nell'interfaccia di creazione di

proprietà solo ed esclusivamente i valori consentiti. Per quanto riguarda il secondo, bisognerà sviluppare degli algoritmi che dopo aver ricevuto dei dati in ingresso (tags binarie, dominio, codominio, ecc) interroghino l'indice, computino i risultati e forniscano in uscita i più probabili valori da selezionare.

La procedura di creazione

C'è da notare che per generare alcuni dei suggerimenti avremo bisogno di valori che devono essere selezionati dall'amministratore: ad esempio, per generare una lista di proprietà a cui la nuova property possa essere subordinata, bisognerà conoscere dominio e codominio della relazione da creare. Sarà quindi necessario dividere la procedura di creazione in 2 fasi, di seguito elencate. Chiameremo *Link Manager* ciò che si occuperà di interrogare l'indice e recuperare informazioni sulle tag binarie, *Ontology Manager* ciò che si occuperà di processare le informazioni, generare i suggerimenti, modificare l'ontologia. Vedremo che nel nostro caso si tratterà di classi Java. Come esempio, prenderemo come proprietà da creare `citizenOf`. Per il caso in cui vi è una tag binaria come modello, utilizzeremo la tag `Frank Zappa-->citizenOf-->United States of America`.

Fase 1: scegliere dominio e codominio

Inizialmente, l'Ontology Manager interroga il Link Manager e recupera *tutte* le tag binarie che hanno predicato simile a quello della tag selezionata (corrispondente al nome della pagina o al predicato della tag binaria di base, a seconda dei casi). Sui risultati ottenuti, l'OM calcola i più probabili valori per dominio e range: in genere, se un tipo prevale nettamente sugli altri viene scelto; altrimenti si comincia a verificare se vi sono sopraclassi comuni, quali sono le più usate, e così via fino a ottenere un unico valore sia per dominio che per range. Questa procedura verrà spiegata in dettaglio nel capitolo 4. Se l'amministratore ha scelto una particolare tag binaria come modello della nuova proprietà, vengono spediti all'Ontology Manager i dati relativi alla tag e vengono verificati i tipi delle pagine di origine e destinazione. A questo punto viene creata l'interfaccia per la selezione di dominio e codominio.

Prendiamo ad esempio il caso del dominio: in primis, vi sarà una lista di valori tra cui scegliere. Se non abbiamo alcuna tag come base, saranno presenti *tutte* le classi dell'ontologia; altrimenti, vi saranno solo quelle com-

patibili con il tipo della pagina di origine: ad esempio, se il tipo di Frank Zappa è *Musician*, saranno selezionabili come dominio Musician e tutte le sue sopraclassi (quindi, ad esempio, *Person*). Sarebbe sbagliato considerare anche le sottoclassi: se selezionassimo una sottoclasse di Musician, non potremmo poi usare la property creata per promuovere la relazione! Sempre nel caso in cui nel creare la proprietà fossimo partiti da una particolare tag, subito dopo verrà stampato a video il tipo della pagina di origine. Infine, in entrambi i casi viene stampato il suggerimento dell'Ontology Manager. C'è da dire che la classe suggerita potrebbe non essere presente in lista: in questo caso, il suggerimento viene reso noto, ma l'amministratore viene avvertito che la classe suggerita non è compatibile con il tipo della pagina di origine. L'amministratore così ha diversi elementi per scegliere il dominio: in primis può scegliere solo tra valori validi; conosce il tipo della pagina di origine, se ha una tag come modello; sa quale potrebbe essere il tipo più *probabile* in base a quanto suggerito dall'utenza.

La determinazione del codominio procede secondo gli stessi principi, eccetto per il fatto che la pagina tenuta in considerazione è quella di destinazione.

Infine, sempre nel caso avessimo una tag come base, vi è un'ultima opzione, è quella della ossia la scelta del nome della property: in genere il nome può restare quello del predicato della tag binaria, tuttavia in un caso si rivelerebbe necessario cambiarlo: se esiste già una property con quel nome ma non è compatibile con il link da promuovere, bisognerà crearne una nuova, che però non dovrà avere lo stesso nome della property già esistente. Si dà quindi la possibilità di cambiare nome.

Una volta scelti dominio, codominio e nome, questi valori vengono spediti all'Ontology Manager e si passa alla fase successiva.

Fase 2: scegliere gli attributi

Si passa ora alla selezione degli attributi della nuova proprietà. L'Ontology Manager riceve il dominio e codominio che sono stati scelti in precedenza; in base a questi, vengono generate le liste di selezione per 3 possibili attributi:

- **SubPropertyOf** : viene data una lista di possibili proprietà di cui la nuova property potrebbe essere sottoproprietà. I requisiti per essere in lista saranno che rispettivamente il dominio e il range della potenziale

superproperty dovranno essere sopraclassi del dominio e del range della nuova property.

- **EquivalentPropertyOf** : viene data una lista di possibili proprietà equivalenti a quella da creare. Dominio e codominio dovranno essere gli stessi della nuova proprietà.

- **InverseProperty** : viene data una lista di possibili proprietà inverse a quella da creare. In questo caso rispettivamente dominio e codominio della possibile proprietà inversa dovranno essere uguali a codominio e dominio della nuova proprietà.

Nel caso una delle liste sia vuota, il corrispondente attributo non sarà selezionabile.

A seguire, l'Ontology Manager interroga nuovamente il Link Manager per generare i suggerimenti riguardanti gli attributi rimanenti, ossia **FunctionalProperty**, **InverseFunctionalProperty**, **SymmetricProperty**. Vengono forniti in uscita dei valori percentuali che indicano il numero di link processati compatibili con l'attributo da "suggerire"; se questi superano una certa soglia l'aggiunta dell'attributo sarà consigliato all'amministratore. Bisogna a questo punto fare un piccolo distinguo tra gli attributi di funzionalità/funzionalità inversa e di simmetria.

I primi 2 indicano attributi della nuova property in un certo senso *restrittivi*: ad esempio, se la proprietà è funzionale, potremo avere un solo valore per ogni elemento del dominio, e quindi, nel caso questo già fosse fissato, ci sarebbe *impedito* di inserirne altri. La funzionalità è un attributo fortemente vincolante, quindi il valore percentuale di soglia dovrebbe essere molto alto: non scegliamo il 100% perchè teniamo conto comunque di un certo margine di errore (come ben sappiamo i suggerimenti degli utenti sono ben lontani dall'infallibilità), quindi poniamo il valore di soglia al *90%*. In questo modo, si è ragionevolmente certi che una proprietà sia funzionale, in quanto quasi tutte le pagine di origine coinvolte hanno una sola tag binaria corrispondente alla proprietà da creare. Per quanto riguarda invece l'attributo di simmetria, possiamo dire che questo in un certo senso "estenda" la proprietà senza essere vincolante: il fatto che una proprietà sia simmetrica non pone alcuna restrizione aggiuntiva sulle relazioni che possiamo creare usandola, e anzi contribuisce ad aggiungere ancora nuove relazioni. In questo caso quindi per la scelta del valore di soglia possiamo essere molto più flessibili: scegliamo

quindi un valore pari al 60% . Un valore così basso è anche giustificato dalla natura stessa della proprietà: molto spesso magari un utente si limita a definire una relazione solo in un verso, dando per scontato che non sia necessario definirla anche nell'altro.

Ultimo attributo che è possibile scegliere è quello di transitività. Una volta fatte le dovute scelte, l'Ontology Manager si occuperà di modificare l'ontologia creando una property con le caratteristiche selezionate dall'amministratore. Se vi era una tag binaria come modello, questa verrà inserita nell'ontologia. A questo punto, sarà mostrato un messaggio che indicherà la corretta esecuzione dei comandi, se la procedura è andata a buon fine.
L'intero processo riguardante la modifica dell'ontologia è riassunto in Figura 3.1. Quello considerato è il caso in cui si abbia una tag come base per la costruzione della proprietà; nell'altro caso le fasi iniziali riguardanti la scelta della tag binaria non saranno presenti.

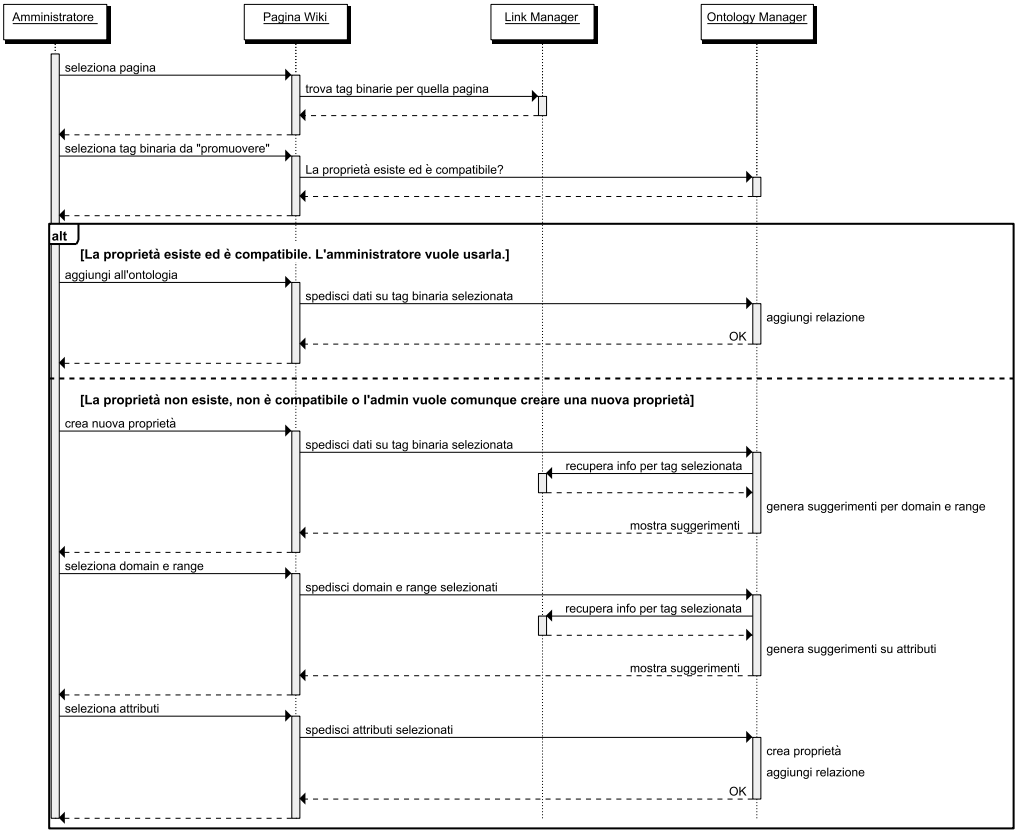

Figura 3.1: UML Sequence Diagram sulla creazione di una nuova relazione.

Capitolo 4

Implementazione e Risultati

Passiamo ora a parlare di come quanto esposto nel Capitolo 3 sia stato effettivamente implementato, e a mostrare un piccolo esempio d'uso.

4.1 Scelta del motore Wiki

Per la realizzazione del Wiki abbiamo scelto come software **JSPWiki** , un motore wiki basato sulla tecnologia standard J2EE (Java, Servlet e JSP). JSPWiki è stato sviluppato da Janne Jalkanen a partire dal 2001 e rilasciato sotto licenza LGPL; i server Sun Macrosystems lo includono tra le loro core applications. Tra le caratteristiche offerte da questo software vi sono:

- Un linguaggio di markup semplice e intuitivo, un'estensione di quello usato in *PHPWiki*;
- La possibilità di allegare files alle pagine;
- Il salvataggio delle vecchie versioni delle pagine, nonchè la possibilità di mostrare le differenze tra 2 versioni e di cancellarle.
- La possibilità di sviluppare *plugin* (utilizzabili sia nel testo delle pagine wiki che nella struttura delle pagine stesse) che permettano di estendere le potenzialità del wiki;
- la possibilità di modificare l'aspetto del wiki tramite l'uso di *template*. Ciò si rivelerà particolarmente utile per l'inserimento dell'interfaccia grafica per la creazione delle nuove proprietà.

Inoltre, il fatto che il software sia scritto con tecnologie Java ci permettera di comunicare agevolmente con l'ontologia tramite le librerie Jena, che vedremo in seguito. Faremo girare il wiki su un server *Apache Tomcat* (di cui utilizzeremo la versione 5.5). In Figura 4.1 un esempio di una pagina Wiki creata con JSPWiki.

Figura 4.1: Esempio di pagina wiki in JSPWiki.

4.2 L'ontologia

Oltre al motore wiki dovremo avere disponibile anche un'ontologia di supporto; allo scopo abbiamo creato una semplicissima ontologia "giocattolo" che presenta già dei tipi utilizzabili ed alcune proprietà già impostate. Per il wiki

d'esempio che stiamo creando, utilizzeremo un'ontologia a sfondo musicale. Abbiamo creato questa semplice ontologia con l'ausilio di *Protégé*, un editor di ontologie open source sviluppato alla Stanford University. La semplice ontologia creata, comprensiva di gerarchie di tipi ed alcune semplici proprietà preimpostate è quella mostrata in Figura 4.2, in un grafico disegnato grazie al software GrOWL:

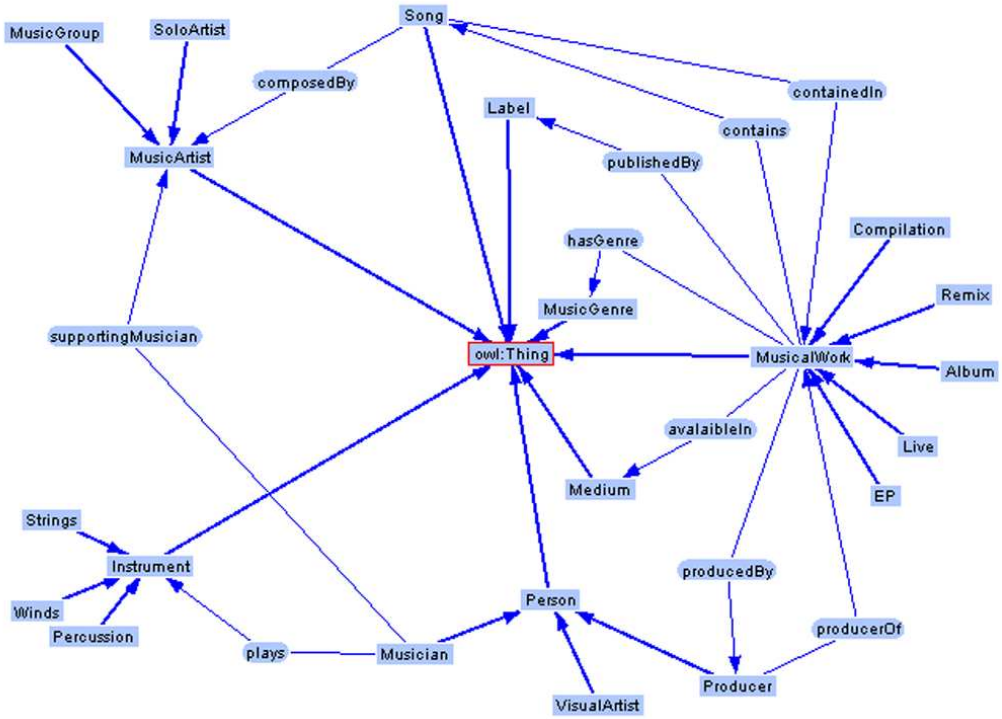

Figura 4.2: L'ontologia utilizzata nell'esempio.

Possiamo notare che, oltre alla solita owl:Thing, nell'ontologia vi sono *Persons*, che possono essere *Musicians* (i quali a loro volta possono essere *Guitarists*, *Drummers*, ecc.), *Producers*, *Visual Artists*; *Music Artist*, ossia *Musicians* e *Music Groups*; *Musical Works* (divisi nei vari tipi come emphAlbums, *EPs*, etc); *Songs*, *Labels*, e così via. Abbiamo definito anche alcune proprietà di partenza, come ad esempio *producedBy* (da Musical Work a Producer), *contains* (da Musical Work a Song) ed alcune altre. Questa semplice ontologia ci servirà da base per costruire il wiki e per mostrare le funzionalità aggiuntive che verranno implementate.

4.3 La gestione delle tag binarie

Passiamo ora a creare la parte del software relativa alla creazione, l'inserimento e la ricerca di tag binarie. Come già detto nei capitoli precedenti, le tag binarie potranno essere inserite direttamente nel testo della pagina.

4.3.1 Definizione

Prima di tutto bisogna definire quale sarà la struttura delle tag. Gli attributi fondamentali da assegnare per definire correttamente una tag binaria saranno i seguenti:

- **From**: la pagina di origine del link.
- **To**: la pagina di destinazione.
- **Tag**: l'etichetta da assegnare all'hyperlink.

Inoltre, ne aggiungiamo un'altra:

- **Occur**: il numero di volte che tale tag binaria appare nella pagina. La sua utilità sarà spiegata in seguito.

4.3.2 Inserimento

Ora che ne conosciamo la struttura, non resta che definire *come* inserire la tag binaria nella pagina. In questo ci è d'aiuto la possibilità offerta da JSPWiki di definire *plugin* inseribili direttamente nel testo della pagina: si tratta di semplici aggiunte al linguaggio di markup che ci permetteranno di inserire dei parametri. Ogni volta che la pagina viene visualizzata, la classe Java corrispondente al plugin inserito viene invocata ed esegue le proprie istruzioni, ritornando una stringa che viene inserita nella pagina wiki. Supponiamo quindi di voler creare un nuovo plugin chiamato **TaggedLinks** che permetta di inserire le tag binarie. I parametri da passare a rigor di logica dovrebbero essere la pagina di origine, la tag e la pagina di destinazione. Tuttavia, visto che in genere si stanno asserendo relazioni aventi come soggetto la pagina che si sta visualizzando, il parametro *from* si rivela ridondante: sarà quindi impostato di default col nome della pagina corrente. Visto che la tag binaria sarà visualizzata come hyperlink, si rende necessario anche un altro parametro, opzionale, che permetta di scegliere un testo alternativo. JSPWiki impone che il plugin debba essere nella forma

```
[{<nome plugin> <parametro1>='<valore>',
<parametro2>='<valore2>', ... <parametroN>='<valore>'}]
```

quindi la sintassi per inserire link taggati sarà

```
[{TaggedLinks to='<valore>', tags='<valore>', alt='<testo>'}]
```

col parametro `alt` opzionale. Il valore del parametro `to` dovrà avere *underscores* (_) al posto degli spazi: questo per fare in modo che la procedura di *parsing* (ossia il recupero delle tag binarie dalla pagina) sia eseguita correttamente. Potranno essere inserite più di una tag, separate da punti e virgola. Inoltre, la pagina di destinazione deve essere una pagina esistente nel wiki.

Ogni volta che quindi la pagina viene visualizzata, per ogni plugin di questo tipo inserito nel testo della pagina (quindi per ogni tag binaria inserita), viene caricata la corrispondente classe Java a cui sono passati i parametri definiti dall'utente. Resta da definire quali saranno le istruzioni che questa classe dovrà eseguire. Sicuramente dovrà stampare a video nella pagina wiki il link alla pagina di destinazione: questo sarà il valore della stringa ritornata. Il risultato è ciò che si vede in Figura 4.3.

Figura 4.3: Link corrispondente a una tag binaria.

Come visibile in figura, quando il puntatore viene trattenuto sul link vengono visualizzate le informazioni sulla tag binaria.

Oltre alla visualizzazione, bisogna gestire l'*indicizzazione* del link, come visto nel Capitolo 3. Di questo si occuperà un'altra classe Java, il Link Manager, che come detto precedentemente gestisce l'interrogazione e la modifica dell'indice. Ad ogni chiamata del plugin andrebbe quindi invocato tale Link Manager, ordinandogli di modificare l'indice; tuttavia, questa scelta si rivelerebbe errata. Per capire perchè, bisogna prima definire come è strutturato l'indice, e come il Link Manager opera su di esso.

L'indice delle tag binarie

La prima scelta possibile per immagazzinare i dati sarebbe costruire un database relazionale che contenga tutte le tag binarie. Tuttavia, vista l'estrema

semplicità della struttura, che non è altro che un insieme di record dello stesso tipo aventi ognuno quattro campi (quelli appena visti, ossia from, to, tag, occur) , si è scelto di creare un semplice indice con **Apache Lucene**. Apache Lucene è una API per il reperimento di informazioni inizialmente implementata in Java, che fornisce funzionalità di indicizzazione e ricerca *full text*. I dati gestiti da Lucene sono rappresentati come documenti (*document*) dotati di campi (*field*) testuali; la totalità dei documenti va a formare l'indice, su cui è poi possibile effettuare ricerche. Nel nostro caso Lucene può rivelarsi particolarmente utile: oltre a essere in Java, esso è usato per l'indicizzazione delle pagine da JSPWiki stesso. Potremmo quindi pensare di costruire l'indice in questo modo: ogni tag binaria sarà un *document* dell'indice, costituito dai 4 *field* from, to, tag, occur.

E' importante notare che Lucene ammette i duplicati: sono consentiti documenti con gli stessi valori. E' per questo che non possiamo permettere l'aggiunta delle tag all'indice direttamente dal plugin: quest'ultimo viene seguito ogni volta che la pagina viene *processata*, quindi non solo quando viene modificata, ma anche quando viene semplicemente visualizzata; continuerebbero quindi ad aggiungersi tag inutili man mano che il wiki viene utilizzato! Ci ritroveremmo con un indice enorme pieno di informazioni errate. Potremmo anche fare in modo che se una tag è già presente nell'indice, essa non venga aggiunta; tuttavia, questo porterebbe a eliminare i duplicati "legittimi" (cioè, tag binarie che sono state usate più volte nella stessa pagina), ed anche a una perdita dell'efficienza notevole, poichè per ogni tag binaria verrebbe invocato il Link Manager e effettuata una ricerca, più un'eventuale modifica. Altro annoso problema riguarda la *cancellazione* di una tag binaria dal testo: in questo caso il plugin non viene proprio invocato, impedendo di fatto la cancellazione della tag! La cosa migliore da fare quindi è *impedire* al plugin qualsiasi operazione di modifica dell'indice: sarà il Link Manager a occuparsene, nel modo che andiamo a illustrare.

Il Link Manager

Come già esposto, il *Link Manager* è la classe Java adibita all'interrogazione e la modifica dell'indice. Appena creato (all'avvio del Wiki) il Link Manager si occupa del completo *reindex* del wiki: recupera la lista delle pagine del wiki e, per ognuna di esse, esegue il *parsing* del testo estraendo tutte le tag binarie presenti, inserendole in seguito nell'indice. Se una tag binaria appare più di

una volta nella pagina, non viene creato un duplicato: semplicemente il valore del field *Occur* del tag in questione viene incrementato di 1. In questo modo l'indice è più compatto, senza perdita di informazione (visto che comunque tutte le istanze di una certa tag binaria saranno sempre presenti nella stessa pagina). In questo modo, avremo già l'indice completo e aggiornato allo startup del wiki.

Tuttavia, non abbiamo ancora risolto il problema di *come* inserire le tag nell'indice dopo l'uso del plugin *TaggedLinks*; la soluzione è la seguente: all'avvio il Link Manager si mette *in ascolto* (tramite una classe apposita chiamata WikiListener); quando una pagina viene modificata e salvata, esso si attiva e fa li reindex di quella pagina. In questo modo l'indice si aggiorna costantemente e *solo* quando c'è un'effettiva modifica del testo di una pagina, di fatto eliminando tutti i problemi che affliggevano l'uso del plugin per la modifica dell'indice. Oltre che per la modifica, il Link Manager si mette in ascolto anche per quanto riguarda un'eventuale *cancellazione* della pagina: se questa avviene, tutti i link con origine o destinazione in quella pagina verranno eliminati.

Ora che abbiamo definito come inserire ed eliminare le tag binarie da una pagina, illustreremo come operare per reperire informazioni dall'indice.

4.3.3 Ricerca

Per effettuare una ricerca nell'indice utilizzeremo lo stesso metodo usato per l'inserimento, ossia la permetteremo tramite un plugin. Una ricerca potrà essere effettuata tramite i seguenti parametri:

- La pagina di origine;
- La pagina di destinazione.
- Le tag. Potranno essere inserite più di una tag, separate le une dalle altre da punti e virgola; se vi è più di una tag inserita, verranno cercate e visualizzate le pagine che sono in relazione tra di loro per *tutte* le tag inserite.

La sintassi del plugin (che chiameremo `SearchLink`) sarà quindi

```
[{SearchLink from='<valore>' to='<valore>'
     tags='<valore1>;...;<valoreN>'}]
```

Tutti i valori sono opzionali, ma *almeno* uno di essi dovrà essere presente. Quando tale plugin viene invocato, esso si serve del LinkManager (che genera una *query* nella sintassi di Lucene) per interrogare l'indice, ed ottiene una lista di tag binarie corrispondenti ai dati inseriti. I risultati vengono in seguito visualizzati sotto forma di un elenco contenuto in un box a scomparsa. Ad esempio, se volessimo generare la discografia delle Mothers of Invention, potremmo inserire nel testo della pagina

```
[{SearchLink from='Mothers of Invention' tags='authorOf'}]
```

Il risultato sarebbe quello illustrato in Figura 4.4

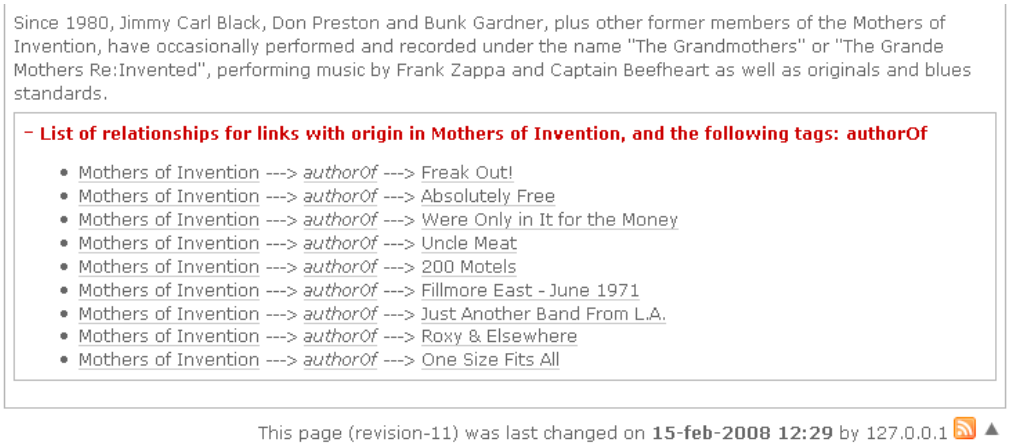

Figura 4.4: Risultati della ricerca di tag binarie.

4.3.4 TagCloud

Per quanto riguarda la creazione di tagclouds, utilizzeremo ancora un altro plugin, che chiameremo `LinkCloud`. In questo caso i parametri da passare al plugin saranno:

- L'eventuale pagina sui cui dati vogliamo costruire la tagcloud;
- Il massimo numero di tag da visualizzare.

La sintassi sarà quindi:

```
[{LinkCloud page='<valore>' tagnum='<valore>'
```

Entrambi i parametri sono opzionali. Se il primo parametro è nullo, verrà

costruita una tagcloud relativa all'intero wiki, altrimenti le tag visualizzate saranno relative alle sole tag binarie in cui è coinvolta, come soggetto o oggetto, quella pagina. Se il secondo parametro è nullo, il numero massimo di tag visualizzabili è fissato a 20.

Una volta invocato il plugin, il Link Manager effettua una ricerca e recupera le tag binarie relative a tutto il wiki o alla specifica pagina, a seconda dei parametri passati. In seguito estrae dalle tag binarie tutti i predicati (le etichette vere e proprie) e crea una lista ordinata per occorrenze, in numero decrescente. Seleziona i primi `tagnum` valori della lista, e a seconda del loro *rank* (le tag più usate avranno un rank maggiore) le stampa a video in dimensioni diverse, dentro un box. Ad esempio, creiamo una tagcloud globale: `[{LinkCloud}]` Il risultato è quello in Figura 4.5

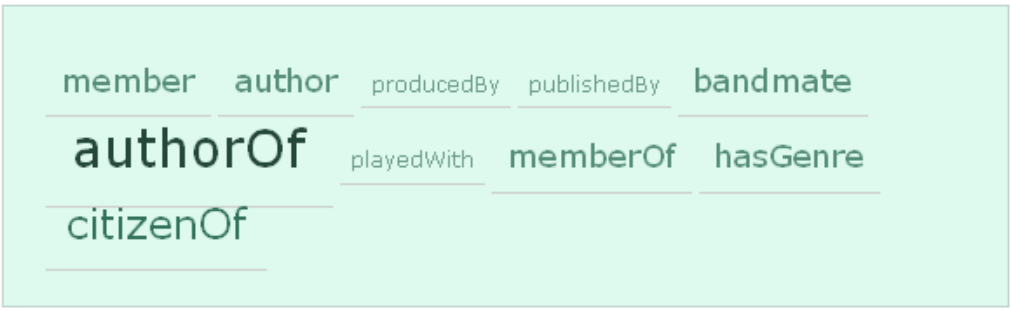

Figura 4.5: Tagcloud

4.3.5 Una piccola interfaccia

Al fine di facilitare l'inserimento dei plugin appena descritti, è stata realizzata una piccola interfaccia inserita nell'editor del wiki, illustrata in Figura 4.6.

4.4 Creazione di relazioni e formalizzazione di nuove proprietà

Passiamo ora a vedere come sono state implementate le funzioni di modifica dell'ontologia.

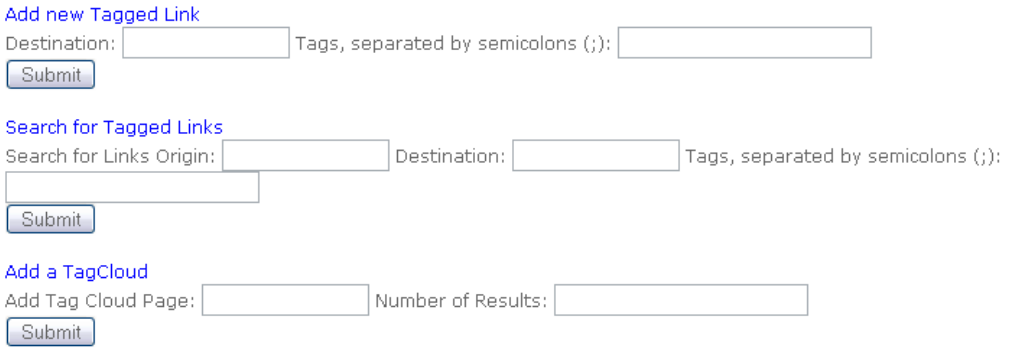

Figura 4.6: L'interfaccia per la gestione delle binary tags

4.4.1 Comunicare con l'ontologia: Jena

Come interfaccia tra le classi Java e l'ontologia in RDF/OWL abbiamo scelto di usare *Jena*, un framework open source per la creazione di applicazioni orientate al Semantic Web. Esso fornisce un'API che permette di estrarre informazioni e di scrivere grafi RDF, che sono rappresentati come "modelli" astratti in cui è possibile inserire anche dati da file, database, URL e quant'altro. L'API fornisce classi, metodi e letterali che semplificano notevolmente la gestione di un modello RDF; inoltre, Jena permette anche di creare modelli che supportano OWL, permette query tramite *SPARQL* e fornisce un reasoner interno (anche se è possibile usare reasoner esterni come *Pellet*).

4.4.2 L'Ontology Manager

Come già spiegato nel Capitolo 3, l'Ontology Manager si occuperà di gestire la comunicazione tra il wiki e l'ontologia che ne è alla base, utilizzando appunto le API fornite da Jena. Come per il Link Manager, è stato implementato con una classe Java che viene istanziata all'avvio del Wiki; alla sua creazione, legge il file dove è contenuta l'ontologia e crea un modello basato su di essa. Inoltre, recupera l'elenco delle pagine e verifica se tutte le pagine sono presenti nell'ontologia come individui. Se una pagina è già presente nell'ontologia, il Manager non fa nulla; altrimenti crea un nuovo individuo di tipo `owl:Thing` e lo inserisce nell'ontologia. In questo modo ogni pagina ha associato un oggetto nel modello, su cui apporteremo le modifiche. I tipi delle pagine già presenti nel modello non vengono modificati in alcun modo. In seguito l'Ontology Manager si mette in attesa di istruzioni. Si occuperà

della creazione di suggerimenti, della visualizzazione delle liste presenti nelle interfacce, e della modifica dell'ontologia vera e propria, ossia, nel nostro caso, l'assegnazione dei tipi e la creazione di relazioni e proprietà.

4.4.3 Visualizzare i dati relativi a una pagina presenti nell'ontologia

Ovviamente, oltre a poter creare nuove relazioni, l'amministratore deve essere in grado di visualizzare *cosa* è già stato definito nel modello riguardo una certa pagina; per far questo, nell'ambiente di modifica della pagina è stata inserita una tab (chiamata *Ontology Data*) che porta a una semplice schermata contenente l'elenco di asserzioni riguardo la pagina corrente: saranno visualizzati il tipo della pagina con relative superclassi, e le relazioni in cui la pagina è coinvolta. Un esempio è mostrato in Figura 4.7.

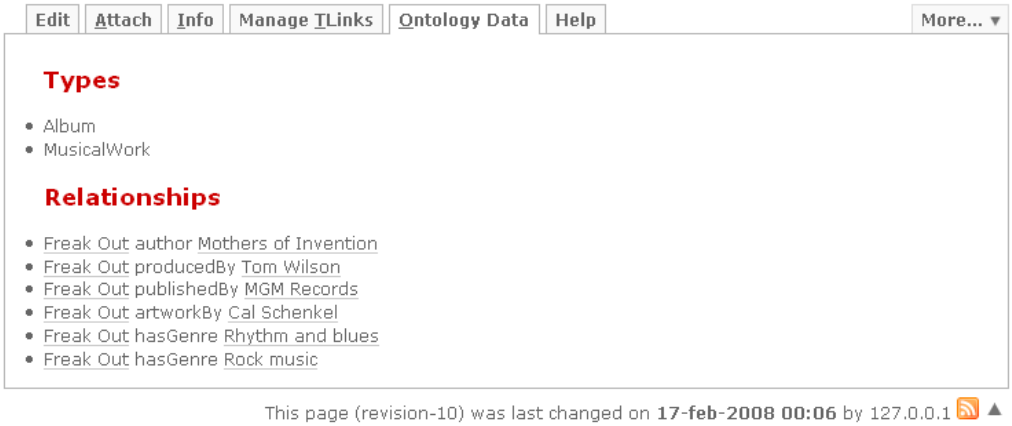

Figura 4.7: I dati nell'ontologia relativi alla pagina corrente.

4.4.4 Definire il tipo di una pagina

Per la definizione del tipo di una pagina come già proposto sarà inserita una piccola interfaccia nell'editor del wiki, illustrata in Figura 4.8.

Come già esposto nel Capitolo 3, ogni volta che il tipo di una pagina viene cambiato, per evitare incompatibilità con eventuali relazioni già presenti l'individuo relativo alla pagina viene *eliminato* e con esso tutte le asserzioni in cui compare; in seguito ne viene creato uno ex novo, avente il tipo selezionato.

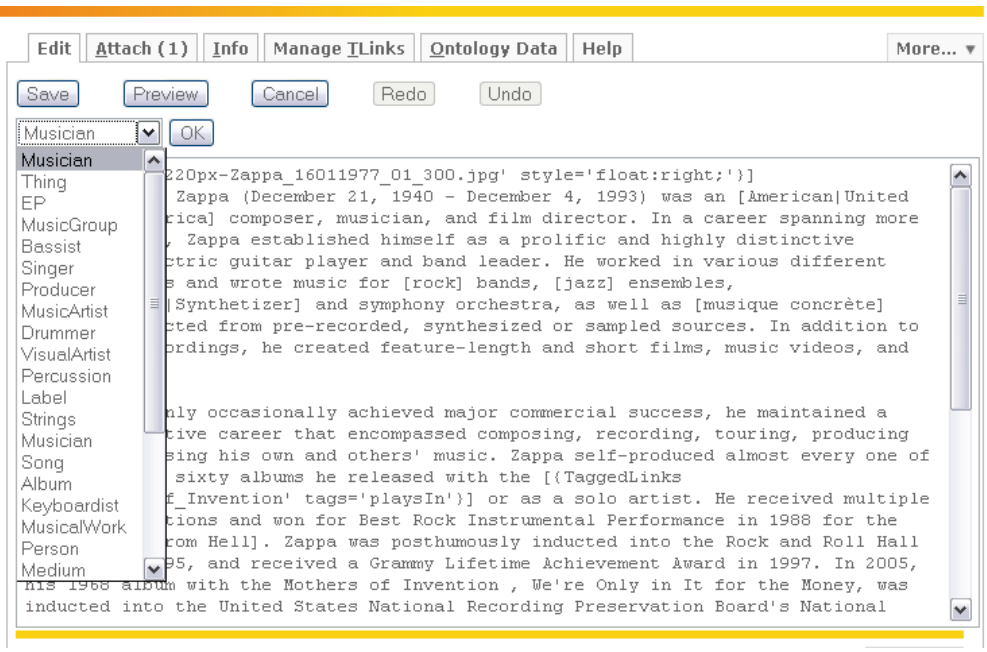

Figura 4.8: L'interfaccia per la scelta del tipo.

4.4.5 Inserire nuove relazioni

Passiamo ora all'inserimento di nuove relazioni. Per prima cosa ci occuperemo dei casi in cui il tipo di relazione è già esistente; di quelli in cui bisognerà creare una nuova proprietà ci occuperemo in seguito. L'interfaccia per la creazione di nuove relazioni sarà integrata nella struttura del wiki. Più precisamente, quando si entra nel contesto di modifica di una certa pagina, apparirà una nuova tab chiamata *Manage TLinks* che, se selezionata, porterà a una nuova schermata. In questa schermata saranno presenti tutte le tag binarie della pagina corrente. Per crearla, viene invocato il Link Manager che effettua una ricerca di tutte le tag aventi come pagina di origine o di destinazione la pagina in questione; i risultati saranno inseriti nella tabella, avente quindi come colonne i campi *From*, *To*, *Tag* e *Occurences* (ossia il numero di volte che la tag compare nella pagina: questo parametro ci sarà utile nella creazione dei suggerimenti riguardanti dominio e codominio di una nuova proprietà) e come righe le tag binarie. Inoltre, di fianco a ogni riga sarà posta una checkbox; in questo modo sarà possibile selezionare la tag da "promuovere" a relazione. Il risultato è visibile in Figura 4.9.

Per avviare il processo di "promozione" di una tag binaria basta cliccare

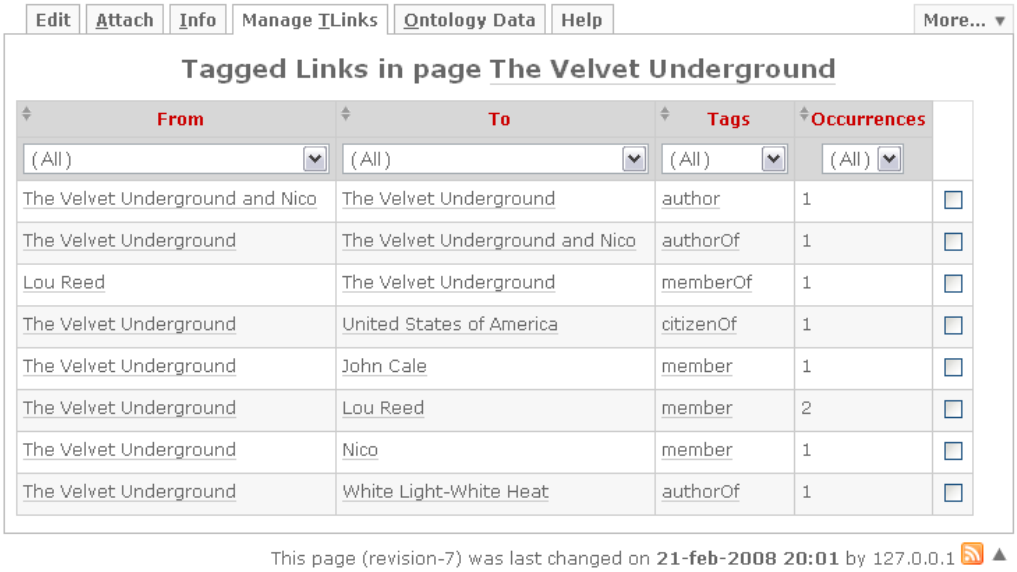

Figura 4.9: Schermata di gestione delle tag binarie.

sulla checkbox corrispondente alla nostra scelta. A questo punto i dati della tag binaria selezionata vengono spediti all'Ontology Manager, che compie le seguenti operazioni:

- Verifica che esista una proprietà con il nome pari al predicato della tag binaria;
- Se esiste, verifica che domain e range di tale proprietà siano compatibili con i tipi delle pagine di origine e destinazione;
- Se la proprietà da usare è funzionale verifica che vi sia nell'ontologia un'asserzione avente per soggetto la pagina di origine e per predicato la proprietà: in caso positivo, inserire un'altra relazione di questo tipo porterebbe a violare la funzionalità della property.

Se gli esiti dei tre punti sopra elencati sono positivi, si rende disponibile l'opzione per promuovere istantaneamente la tag binaria a relazione tramite un semplice click. Ad esempio, come tag da formalizzare scegliamo `Frank Zappa--> memberOf--> Mothers Of Invention`: sotto la schermata appariranno le checkbox in Figura 4.10:

Se scegliamo di usare la relazione in questione, verrà invocato l'Ontology Manager che aggiungerà la relazione all'ontologia, scrivendo il seguente codice nel file .owl dove è contenuta l'ontologia:

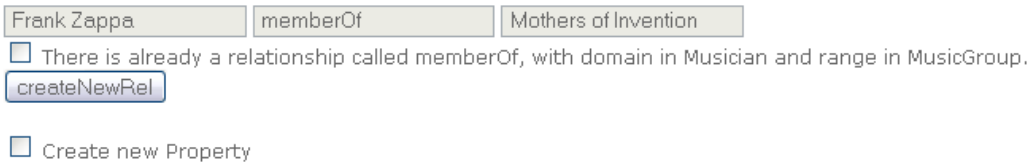

Figura 4.10: Inserimento di relazione compatibile con una property già esistente.

```
<Musician rdf:about="<namespace>#FrankZappa">
 <playsIn rdf:resource="<namespace>#MothersOfInvention"/>
</Musician>
```

Se provassimo invece a formalizzare la relazione (errata)
`United States of America--> memberOf--> Mothers Of Invention`
il risultato sarebbe quello in Figura 4.11. In questo caso non è possibile

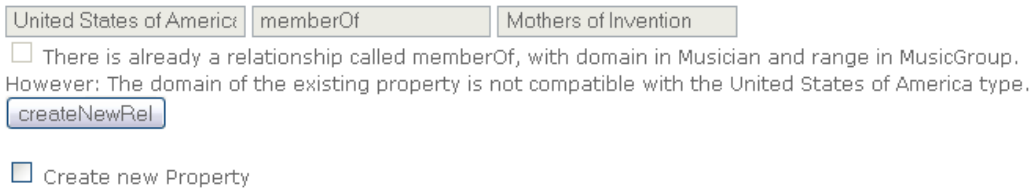

Figura 4.11: La relazione non è compatibile con la property già esistente.

inserire la relazione (per incompatibilità di dominio) e quindi la checkbox viene disattivata.

4.4.6 Creare nuove proprietà

Per quanto riguarda la creazione di nuove proprietà, se ci troviamo su una pagina che rappresenta il predicato di una tag binaria le checkbox non saranno presenti; la proprietà sarà creata senza una tag binaria come modello. Se invece ci troviamo sulla pagina di un'istanza, nel caso avessimo selezionato una tag binaria il cui predicato non corrisponde a nessuna proprietà esistente, o volessimo in tutti i casi crearne una nuova, basta spuntare la checkbox *Create new Property*. Come già spiegato nel Capitolo 3, il primo passo da compiere sarà scegliere nome, dominio e codominio della nuova proprietà. La schermata che si presenterà sarà quella mostrata in Figura 4.12.

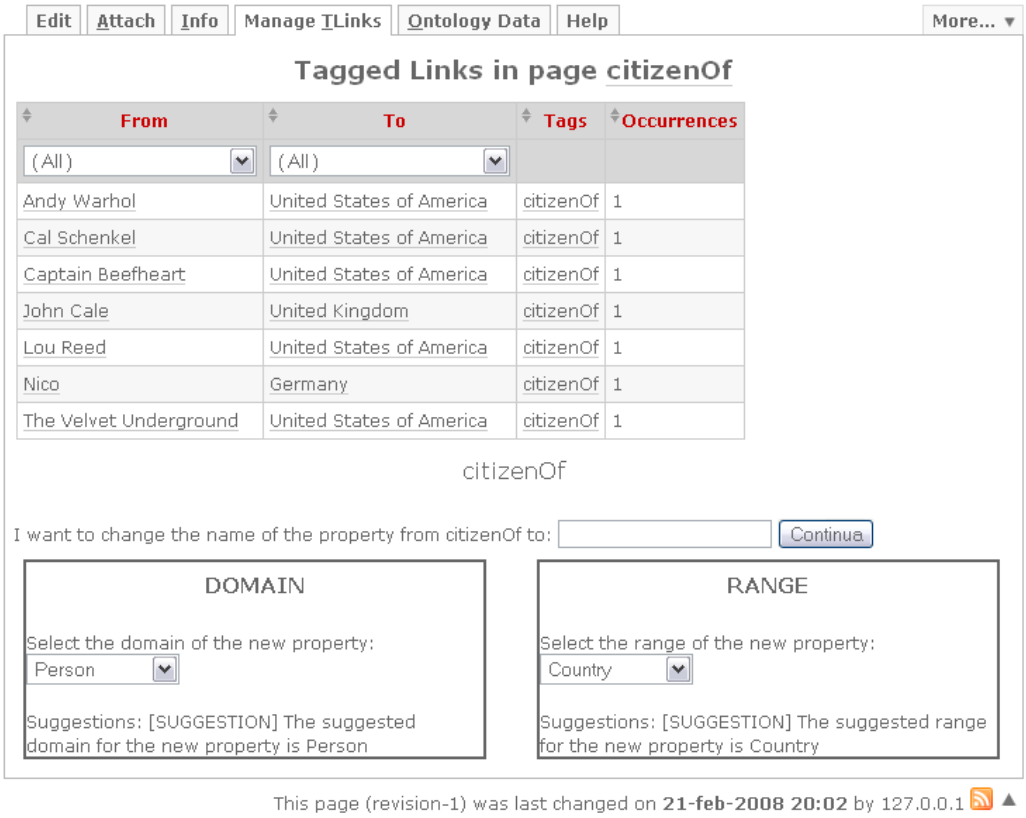

Figura 4.12: Creare una nuova proprietà - scelta di dominio e codominio.

E' presente un campo in cui inserire un nuovo nome (nel caso ci fosse già una proprietà con quel nome la procedura non sarà portata a termine); nel caso il campo fosse lasciato vuoto, verrà usato il nome di default, ossia il predicato della tag binaria di partenza. Vi sono inoltre due box per la scelta di dominio e codominio: in ognuno di essi è presente una lista di possibili valori (creata secondo i criteri visti nel Capitolo 3), il tipo delle pagine di origine e destinazione della tag di partenza, e il valore *suggerito*.

La procedura di suggerimento di dominio e codominio

Illustriamo ora l'algoritmo utilizzato per generare il suggerimento visualizzato nei box di scelta del dominio e codominio. Essendo la procedura la stessa per entrambi, ci limiteremo a illustrare quella riguardante il dominio. Inizialmente, vengono mandati all'Ontology Manager il predicato da formalizzare e, eventualmente, il nome della pagina di origine (se ci stiamo

basando su una particolare tag binaria). Se quest'ultimo parametro viene inviato, interrogando l'ontologia, viene determinato il tipo della pagina. In seguito, vengono recuperate dall'indice (tramite il LinkManager) *tutte* le tag binarie aventi quel particolare predicato; queste tag vengono utilizzate per creare una lista, su cui poi opereremo. Ogni elemento di questa lista sarà costituito da una classe dell'ontologia e da un valore pari al numero di pagine di origine (estratte dalle tag precedentemente recuperate) aventi come tipo quella classe. Per ogni tag binaria:

1. Estraiamo il tipo della pagina d'origine (di destinazione, se ci occupiamo del codominio).

2. Se la lista non contiene questo tipo, vi aggiungiamo un nuovo elemento costruito come <tipo, numero di occorrenze della tag binaria>; Altrimenti, aggiorniamo l'elemento trasformandolo in <tipo, valore precedente + numero di occorrenze della tag binaria>.

In questo modo avremo una lista contenente tutti i tipi "di dominio" e il numero di volte che essi appaiono. Il numero di occorrenze da qui in poi verrà chiamato, per un generico tipo x, n(x).

Su questa lista vengono eseguiti i seguenti passi, finchè nella lista non rimarrà un solo elemento:

1. La lista viene ordinata per occorrenze, in ordine decrescente. I primi valori saranno quindi quelli relativi ai tipi più usati.

2. Se vi è un valore preponderante questo viene scelto come suggerimento. In genere un valore viene considerato preponderante se rappresenta il 70% o più della somma di tutti i valori (cioè, il 70% delle pagine coinvolte hanno quel particolare tipo).

3. Viene costruito un elenco di termini. Per capire cos'è un termine, immaginiamo la gerarchia delle classi dell'ontologia come un albero, avente come nodo radice `owl:Thing`. Un termine sta a indicare che due nodi x e y possono assorbirsi l'un altro o *fondersi* in un altro nodo z con

$$n(z) = n(x) + n(y)$$

ottenendo comunque dei nodi più *significativi*. La struttura di un termine è la seguente:

`<nodo1><nodo2>---><nodo3>; <n(nodo1) + n(nodo2)>`

L'elenco di termini viene costruito in questo modo: tutti gli elementi della lista vengono confrontati fra di loro; per due generici elementi x e y, abbiamo che:

- Se $x \subset y$, viene costruito il termine `x,y-->y; n(x) + n(y)`;
- Se $y \subset x$, viene costruito il termine `y,x-->x; n(x) + n(y)`;
- Se $x, y \subset z$ (dove z può essere un qualsiasi elemento della gerarchia non necessariamente compreso nella lista iniziale), viene costruito il termine `x,y-->z; n(x) + n(y)`;

Per costruzione l'elenco avrà in cima i termini aventi come risultato i nodi di valore maggiore. Se l'elenco dei termini è vuoto, si esce dal ciclo.

4. Si inizia a scorrere l'elenco dei termini, in cerca di sottoclassi dirette uniche. Una sottoclasse diretta unica è una foglia dell'albero che ha un unico padre (ricordiamo che in OWL è ammessa l'ereditarietà multipla tra le classi). Verranno considerati quindi i soli termini del tipo `x,y-->y` e in cui x non abbia alcuna sottoclasse. Se risulta che x è sottoclasse diretta unica di y, si elimina x dalla lista di partenza e si pone $n(y) = n(x) + n(y)$. Si torna al punto 1. In questo modo vengono mano a mano l'albero viene compattato, eliminando tutte le foglie aventi le caratteristiche sopra descritte (a meno che non si esca dal ciclo prima).

5. Se il punto precedente non ha apportato modifiche alla lista, l'elenco dei termini viene scansionato nuovamente, stavolta cercando i generici elementi `x,y-->y` (in questo caso la x avrà più di un nodo padre). A questo punto potremmo semplicemente compattare i nodi e andare avanti: tuttavia, sappiamo che y non è *l'unica* superclasse di x (altrimenti avremmo eliminato x già al passo 4). E se compattare x con un'altra delle sue sopraclassi fosse più vantaggioso? Per capire quale termine contenente x convenga prendere, operiamo nel seguente modo: recuperiamo tutti i termini che coinvolgono x (quindi tutti i termini del tipo `x,z-->z` o `x,y-->z`); per ognuno di essi, costruiamo una lista *provvisoria* che rappresenta la situazione che avremmo se avessimo scelto quel termine. Tra tutte le liste provvisorie, scegliamo quella che

offre le potenzialità maggiori, ossia quella che crea nodi di valore maggiore. Il termine corrispondente verrà quindi utilizzato per modificare la lista di partenza: se è del tipo `x,y-->y` si opererà come nel passo 4; altrimenti, se è del tipo `x,y-->z`, si crea un nuovo elemento z nella lista (se non c'è già) con $n(z) = n(x) + n(y) + n(z)$, dove $n(z) = 0$ se z è un nuovo elemento. Si torna al punto 1.

6. Se i due passi precedenti danno esito negativo (ossia non vi sono più termini del tipo `x,y-->y`) prendiamo il primo termine `x,y-->z` e lo compattiamo come già visto nel punto 5. Si torna al punto 1.

Quando vi è rimasto un solo elemento nella lista o non vengono generati più termini, l'algoritmo è giunto al suo termine. Viene estratto il primo elemento della lista, e viene restituito come suggerimento. Se la procedura appena vista non viene svolta in una pagina rappresentante una relazione, la base di partenza sarà una tag binaria, della cui origine avevamo estratto il tipo in precedenza: se questo è compatibile con la classe della pagina d'origine della tag da cui eravamo partiti, il corrispondente tipo viene suggerito. In caso contrario viene visualizzato oltre al suggerimento anche un messaggio per indicare che il tipo suggerito, non essendo compatibile con la pagina di origine, non può venire usato (altrimenti non sarebbe poi possibile formalizzare la tag binaria di partenza).

Un esempio

Illustreremo qui un esempio di utilizzo dell'algoritmo di suggerimento. Supponiamo di voler creare la proprietà `citizenOf`, intendendola come la relazione di cittadinanza tra un individuo e uno stato. Creiamo la lista dei tipi relativa a `citizenOf`; in Figura 4.13, l'albero delle classi (una parte, quella che ci interessa):

Iniziamo l'iterazione:

1. Poniamo che la lista ottenuta sia la seguente, ordinata:

```
Musician: 10
Person: 10
MusicGroup: 7
Guitarist: 6
VisualArtist: 4
```

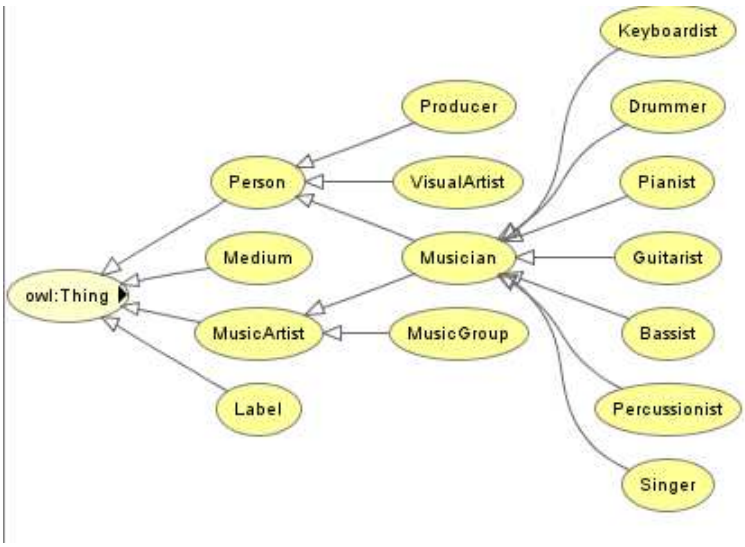

Figura 4.13: Una porzione dell'albero delle classi dell'ontologia

```
Label: 4
Singer: 3
Producer: 1
```

Come si può vedere, alcune tag `citizenOf` sono state usate anche per definire la provenienza di alcuni gruppi musicali e alcune etichette discografiche. Per il nostro modo di intendere il predicato da formalizzare quest'uso è errato: tuttavia, come si vedrà, il contributo di queste tag non sarà globalmente significativo.

2. Non vi è un valore preponderante: sia `Musician` che `Person` rappresentano solo il 22% del totale.

3. Costruiamo i termini:

```
Musician:Person->Person (20)
Musician:MusicGroup->MusicArtist (17)
Guitarist:Musician->Musician (16)
Musician:VisualArtist->Person (14)
Musician:Label->Thing (14)
Singer:Musician->Musician (13)
Musician:Producer->Person (11)
Person:MusicGroup->Thing (17)
```

```
Guitarist:Person->Person (16)
VisualArtist:Person->Person (14)
Person:Label->Thing (14)
Singer:Person->Person (13)
Producer:Person->Person (11)
MusicGroup:Guitarist->MusicArtist (13)
MusicGroup:VisualArtist->Thing (11)
MusicGroup:Label->Thing (11)
MusicGroup:Singer->MusicArtist (10)
MusicGroup:Producer->Thing (8)
Guitarist:VisualArtist->Person (10)
Guitarist:Label->Thing (10)
Guitarist:Singer->Musician (9)
Guitarist:Producer->Person (7)
VisualArtist:Label->Thing (8)
VisualArtist:Singer->Person (7)
VisualArtist:Producer->Person (5)
Label:Singer->Thing (7)
Label:Producer->Thing (5)
Singer:Producer->Person (4)
```

4. Il primo termine che incontriamo che del tipo `x,y-->y` con y unico padre di x è `Guitarist:Musician->Musician (16)` . Cancelliamo Guitarist, il valore di Musician viene aggiornato a 16.

Il ciclo ricomincia.

1. Ordiniamo:

```
Musician: 16
Person: 10
MusicGroup: 7
VisualArtist: 4
Label: 4
Singer: 3
Producer: 1
```

2. Non vi è ancora un valore preponderante. `Musician` rappresenta solo il 35% del totale.

3. Costruiamo i termini:

```
Musician:Person->Person (26)
Musician:MusicGroup->MusicArtist (23)
Musician:VisualArtist->Person (20)
Musician:Label->Resource (20)
Singer:Musician->Musician (19)
Musician:Producer->Person (17)
Person:MusicGroup->Resource (17)
VisualArtist:Person->Person (14)
Person:Label->Resource (14)
Singer:Person->Person (13)
Producer:Person->Person (11)
MusicGroup:VisualArtist->Resource (11)
MusicGroup:Label->Resource (11)
MusicGroup:Singer->MusicArtist (10)
MusicGroup:Producer->Resource (8)
VisualArtist:Label->Resource (8)
VisualArtist:Singer->Person (7)
VisualArtist:Producer->Person (5)
Label:Singer->Resource (7)
Label:Producer->Resource (5)
Singer:Producer->Person (4)
```

4. Selezioniamo il termine `Singer:Musician->Musician (19)`; cancelliamo `Singer`, aggiorniamo `Musician`.

Omettiamo i due cicli seguenti in quanto simili ai precedenti: la computazione continua selezionando i termini `VisualArtist:Person->Person` e `Producer:Person->Person` e compattandoli. Dopo di che, un nuovo ciclo ha inizio:

1. La lista ordinata che si ha a questo punto è la seguente:

```
Musician: 19
  Person: 15
MusicGroup: 7
   Label: 4
```

2. Non vi è ancora un valore preponderante: `Musician` rappresenta solo il 42% del totale.

3. Costruiamo i termini:

   ```
   Musician:Person->Person (34)
   Musician:MusicGroup->MusicArtist (26)
   Musician:Label->Resource (23)
   Person:MusicGroup->Resource (22)
   Person:Label->Resource (19)
   MusicGroup:Label->Resource (11)
   ```

4. Non vi è alcun termine del tipo `x,y-->y` con y unico padre di x; passiamo al punto 5.

5. Scegliamo il termine `Musician:Person->Person (34)` e notiamo come `Musician` sia sottoclasse sia di `Person` che di `MusicArtist`; occorre costruire delle liste provvisorie. I termini coinvolti sono:
   `Musician:Person->Person (34)`
   `Musician:MusicGroup->MusicArtist (26)`
   `Musician:Label->Resource (23)`
   che corrispondono alle tabelle:

   (a) `Person: 34`
   `MusicGroup: 7`
   `Label: 4`

   (b) `MusicArtist: 26`
   `Person: 15`
   `Label: 4`

   (c) `Resource: 23`
   `Person: 15`
   `MusicGroup: 7`

   Ci conviene prendere quindi il primo termine. Torniamo al punto 1.

A questo punto notiamo che `Person` rappresenta il 75% degli elementi della lista; il risultato dell'iterazione sarà quindi il tipo `Person`, che è ragionevole. Questo valore perciò verrà ritornato come *suggerito*.

La scelta degli attributi

Una volta scelti nome, dominio e codominio, passiamo alla selezione degli attributi. La schermata si presenta come in Figura 4.14:

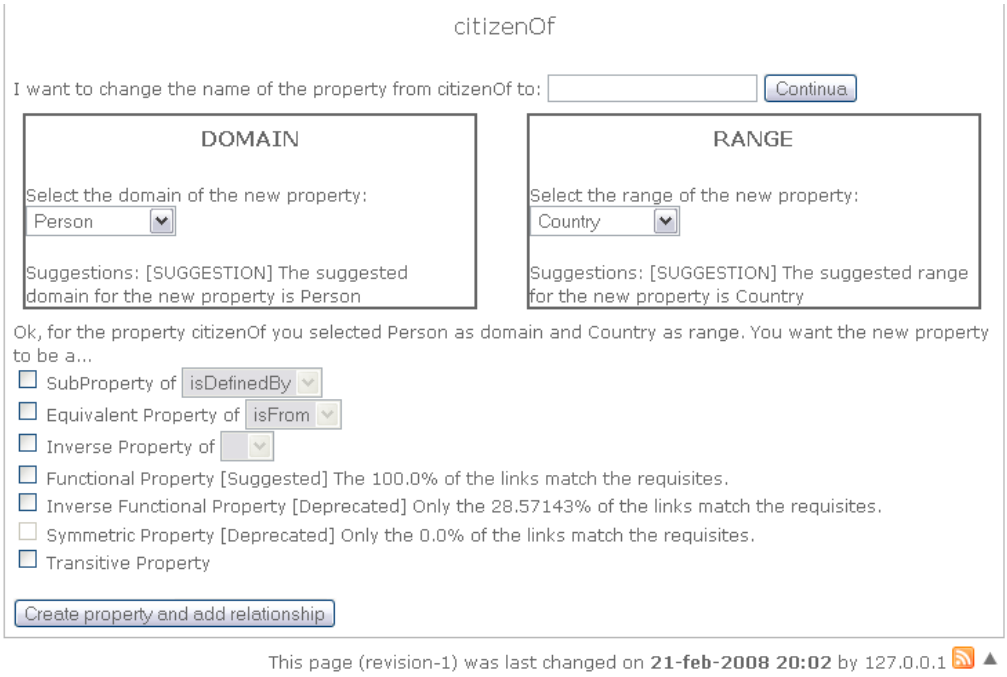

Figura 4.14: Scelta degli attributi.

Sarà possibile selezionare tutti gli attributi descritti nel Capitolo 3. In questo caso non vengono generate liste per i primi 3 attributi: ne vedremo qualcuna più avanti. Viene invece suggerito che la relazione `citizenOf` possa essere una relazione funzionale, e viene sconsigliato l'uso degli altri 2 attributi. Vediamo come vengono generati questi suggerimenti:

- Per la proprietà funzionale, vengono recuperate tramite il Link Manager tutte le tag binarie avente come predicato `citizenOf`; da queste tag viene creato un insieme di tutte le pagine di origine. Per ogni pagina, viene contato il numero di tag binarie presenti aventi predicato `citizenOf`: se ve ne è una sola, viene incrementato un contatore. Alla fine, si calcola il rapporto in punti percentuali tra il contatore e il numero totale di tag binarie: se è maggiore del 90%, l'attributo viene suggerito.

- Per la proprietà inversamente funzionale, il procedimento è lo stesso, tranne che vengono prese in considerazione le pagine di destinazione;
- Per la proprietà simmetrica invece il procedimento è il seguente: dopo aver recuperato le tag binarie, per ogni tag si verifica se esiste nell'elenco la tag "simmetrica": se si, il contatore viene aumentato. Se il rapporto tra il contatore e il numero totale di tag binarie è maggiore del 60%, l'attributo viene suggerito.

Proviamo con un altro esempio: supponiamo di voler creare la proprietà `bandmate` a partire dalla tag binaria `John Cale --> bandmate --> Nico`; scegliamo come dominio e codominio i valori suggeriti, ossia `Musician` come domain `Musician` come range. I suggerimenti saranno quelli in Figura 4.15:

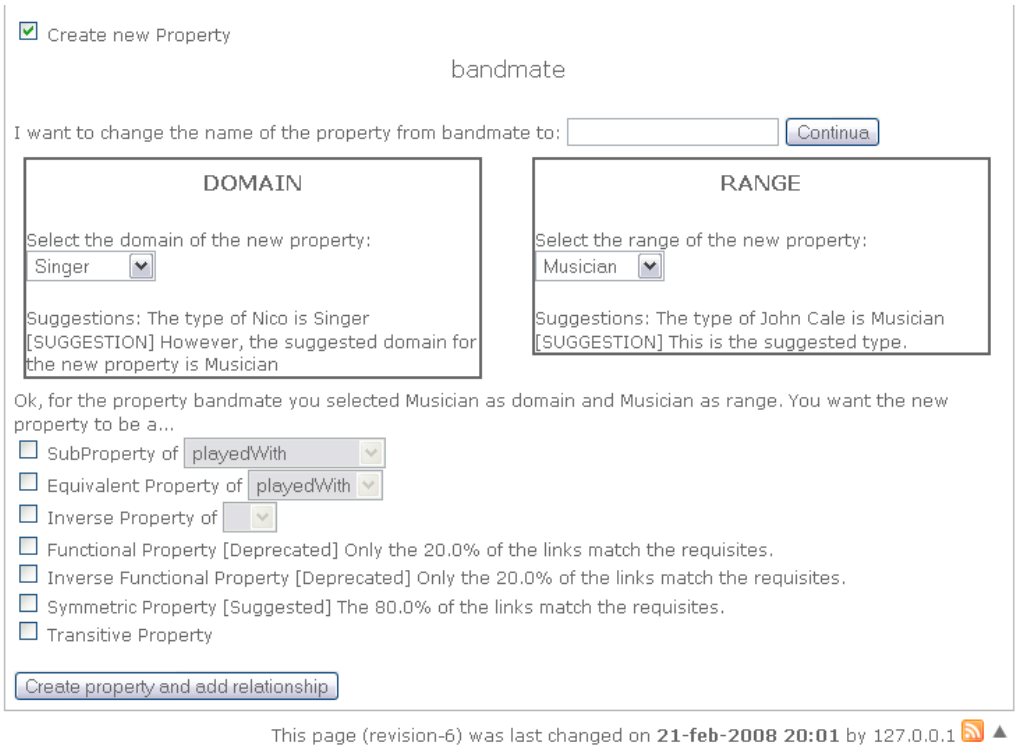

Figura 4.15: Scelta degli attributi - secondo esempio.

Questa volta sarà possibile selezionare i campi *SubPropertyOf* e *EquivalentPropertyOf*: nel primo compaiono le proprietà `playedWith` e `supportingMusician`, nel secondo la sola relazione `playedWith`. Inoltre,

viene suggerita la proprietà simmetrica. Ora proviamo a creare le proprietà. Una volta impostati tutti gli attributi, basta cliccare sul pulsante *Create property and add relationship* e l'ontologia verrà modificata. Il risultato, in codice RDF/OWL, della creazione della proprietà `citizenOf` sarà

```
<owl:FunctionalProperty rdf:about="<namespace>#icitizenOf">
   <rdfs:range rdf:resource="<namespace>#Country"/>
   <rdfs:domain rdf:resource="<namespace>#Person"/>
   <rdf:type rdf:resource="<namespace>#ObjectProperty"/>
</owl:FunctionalProperty>
```

Il risultato della creazione di `bandmate` a partire da
`John Cale --> bandmate --> Nico` sarà: sarà invece:

```
<owl:SymmetricProperty rdf:about="<namespace>#bandmate">
   <rdfs:subPropertyOf rdf:<namespace>#playedWith"/>
   <rdfs:range rdf:resource="<namespace>#Musician"/>
   <rdfs:domain rdf:resource="<namespace>#Musician"/>
   <rdf:type rdf:resource="#ObjectProperty"/>
</owl:SymmetricProperty>

<MusicGroup rdf:about="<namespace>#TheVelvetUnderground">
   <member>
     <Musician rdf:about="<namespace>#JohnCale">
       <bandmate rdf:resource="<namespace>#Nico"/>
     </Musician>
   </member>
   <member rdf:resource="<namespace>#LouReed"/>
   <member rdf:resource="<namespace>#Nico"/>
</MusicGroup>
```

Notiamo come in questo caso oltre a creare la nuova proprietà sia stata formalizzata anche la tag binaria di partenza.

Capitolo 5

Conclusioni e sviluppi futuri

Conclusioni

In questo lavoro ci siamo posti sostanzialmente il problema di coniugare un sistema collaborativo e partecipativo dai contenuti instabili e in continua mutazione come i wiki con un ambiente rigido e formale come quello delle ontologie. L'obiettivo era quello di creare un software che permettesse agli utenti di creare liberamente informazione, ma allo stesso tempo di permettere alla struttura di "capire" le informazioni immesse e di generare nuove informazioni basandosi sui dati di partenza. La principale problematica presentatasi è stata quella di stabilire il campo d'azione degli utenti: lasciarli liberi di modificare l'ontologia, o impedirglielo? Siamo giunti a un compromesso: lasciare che gli utenti "suggeriscano" le relazioni tra le pagine wiki, ma permettere solo a un ristretto numero di utenti esperti di modificare la base semantica; allo stesso tempo, fare in modo che gli amministratori debbano basarsi sui suggerimenti degli utenti per assicurare un'evoluzione dell'ontologia che rispecchi il volere della comunità. I dati immessi dagli utenti (sotto forma di tag binarie, quindi con un mezzo sostanzialmente libero da restrizioni semantiche) vengono quindi processati dal sistema per generare dei suggerimenti che rispecchino ragionevolmente le intenzioni della collettività; questi dati saranno le basi su cui gli amministratori gestiranno l'evoluzione dell'ontologia, impedendo allo stesso tempo il nascere di incoerenze all'interno di quest'ultima. Il sistema, ben lungi dall'essere maturo, presenta dei limiti, dovuti anche all'ambiente "ristretto" in cui ci si è posti, ossia quello delle proprietà; tuttavia i risultati ottenuti sono discretamente positivi e ci portano a pensare che questa soluzione possa essere un buon compromesso

tra il rigore formale proprio del Semantic Web e la dinamicità dei wiki.

Sviluppi futuri

In primis, una possibile aggiunta da considerare riguarda quella della definizione di *datatype properties*, ossia di relazioni aventi come oggetto non una risorsa ma un tipo "primitivo"(stringhe, interi, e così via). Ciò permetterebbe di definire *attributi* relativi ai concetti delle pagine: ad esempio l'anno in cui un album è uscito, date di nascita e morte, il numero di corde di uno strumento... Dal punto di vista implementativo non si creano particolari problemi: basta specificare nella sintassi d'inserimento che non ci si sta riferendo a una pagina ma a un valore, e nell'indicizzazione delle tag binarie si dovrà indicare se si tratta di tag verso oggetti o verso valori; la creazione di tali proprietà verrà gestita in modo differente rispetto alla creazione di *object properties*, offrendo una gamma di possibilità e di suggerimenti diversi.

Altra possibile evoluzione è sicuramente quella di implementare la definizione di classi direttamente dal wiki, permettendo anche l'uso delle *restrictions*, e di permettere la definizione di tipi multipli per le pagine. Potrebbero essere create delle procedure di suggerimento per determinare i tipi da assegnare alle pagine, sempre a partire da tag; potrebbe trattarsi in questo caso di tag nel senso più classico del termine ("unarie") da assegnare alle pagine, in modo da categorizzarle. Anche le tag binarie potrebbero essere utili a questo scopo: ad esempio, poniamo che la classe `Musician` implichi che il musicista suoni almeno uno strumento musicale; se volessimo dare un tipo a `Frank Zappa`, la presenza di una tag `Frank Zappa --> plays --> guitar` potrebbe suggerire che uno dei tipi della pagina potrebbe essere proprio `Musician`.

Per la definizione di nuove classi potremmo usare come sopra tag (che in questo caso indicherebbero delle superclassi) e potremmo utilizzare le tag binarie per generare suggerimenti sulle *restrictions* da definire per quella classe.

La definizione di classi in modo più completo porterebbe anche alla generazione di nuove limitazioni e nuove possibilità nella creazione delle proprietà; andrebbe rivista la procedura di suggerimento di dominio e codominio in modo da tener conto anche delle restrictions delle classi, portando quindi alla generazione di nuovi suggerimenti; inoltre la definizione di tipi multipli ci per-

metterebbe di muoverci in un contesto più ampio. Il sistema di avvicinerebbe sempre di più ad essere un tool per la creazione di wiki semantici.

Altra aggiunta può essere quella di creare un tool per gestire l'ontologia dall'interno del wiki, quindi anche di cancellare proprietà, classi e relazioni tra individui velocemente e senza intaccare la coerenza interna dell'ontologia; si potrebbe inoltre supportare l'uso dei reasoner per generare nuove informazioni e di SPARQL per effettuare query sull'ontologia.

Bibliografia

[Berners-Lee and Fischetti, 1999] Berners-Lee, T. and Fischetti, M. (1999). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper, San Francisco.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema. W3c recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[Bricklin, 2000] Bricklin, D. (2000). The cornucopia of the commons: How to get volunteer labor. `http://www.bricklin.com/cornucopia.htm`.

[de Judicibus, 2008] de Judicibus, D. (2008). World 2.0. *L'indipendente*.

[Klyne and Carroll, 2004] Klyne, G. and Carroll, J. J. (2004). Resource Description Framework (RDF): Concepts and abstract syntax. W3C recommendation, W3C.

[Krötzsch et al., 2006] Krötzsch, M., Vrande¡ci´c, D., and Völkel, M. (2006). *Semantic MediaWiki*. AIFB, Universität Karlsruhe, Germany.

[Laningham, 2006] Laningham, S. (2006). developerworks interviews: Tim berners-lee.

[Leuf and Cunningham, 2001] Leuf, B. and Cunningham, W. (2001). *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley, Boston.

[O'Reilly, 2004] O'Reilly, T. (2004). What is web 2.0: Design patterns and business models for the next generation of software.

[Tazzoli et al., 2004] Tazzoli, R., Castagna, P., and Campanini, S. E. (2004). Towards a Semantic WikiWikiWeb. In *3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan.

[van Harmelen Deborah L. McGuinness, 2004] van Harmelen Deborah L. McGuinness, F. (2004). Owl web ontology language overview.