Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria di Modena

Corso di Laurea in Ingegneria Informatica

COMPARAZIONE DI STRUMENTI SOFTWARE PER LA CREAZIONE, LA GESTIONE E L'INTEGRAZIONE DI ONTOLOGIE

Relatore:
Chiar.mo Prof. Sonia Bergamaschi

Correlatore:
Ing. Serena Sorrentino

Candidato:
Lisa Magnani

Anno Accademico 2007/2008

INDICE

1. INTRODUZIONE pag. 5

1.1 Introduzione pag. 5
1.2 Brevi cenni sulle nozioni fondamentali pag. 6
1.2.1 Il Web Semantico pag. 6
1.2.2 Il Linguaggio OWL pag. 7
1.2.3 La nozione di Ontologia pag. 8

2. ANALISI DEGLI EDITOR pag. 9

2.1 Protégé pag. 9
2.1.1 Layout pag. 9
2.1.2 Creazione, modifica ed eliminazione pag. 13
2.1.3 Esplorazione e visualizzazione pag. 14
2.1.4 Gestione dei contenuti pag. 15
2.1.5 Plugin e supporti esterni pag. 16

2.2 Swoop pag. 22
2.2.1 Layout pag. 22
2.2.2 Creazione, modifica ed eliminazione pag. 23
2.2.3 Esplorazione e visualizzazione pag. 25
2.2.4 Gestione dei contenuti pag. 28
2.2.5 Plugin e supporti esterni pag. 32

2.3 OBO-Edit pag. 33
2.3.1 Layout pag. 33
2.3.2 Creazione, modifica ed eliminazione pag. 34
2.3.3 Esplorazione e visualizzazione pag. 37
2.3.4 Gestione dei contenuti pag. 39
2.3.5 Plugin e supporti esterni pag. 42

3. CONFRONTO DEGLI EDITOR pag. 45

3.1 Usabilità pag. 45
3.2 Scalabilità pag. 46
3.3 Stabilità pag. 46

3.4 Integrazione pag. 48
3.5 Documentazione pag. 49
3.6 Originalità pag. 50

4. CONCLUSIONI pag. 52

5. FONTI E RIFERIMENTI pag. 53

Indice delle figure

Figura 1: Schermata iniziale pag. 10
Figura 2: Pannello delle Entities pag. 11
Figura 3: Pannello Data Properties pag. 12
Figura 4: OWL Viz pag. 17
Figura 5: DL Query pag. 18
Figura 6: Cloud Views pag. 19
Figura 7: Pannello con il comando *Editable* pag. 24
Figura 8: Pannello Absract Syntax pag. 26
Figura 9: Pannello RDF/XML pag. 26
Figura 10: Comando *Bookmarks*, nello sfondo il pannello Ontology Info dell'ontologia selezionata pag. 27
Figura 11: Pannello Resource Holder pag. 29
Figura 12: Pellet Query pag. 30
Figura 13: Pannelli principali di OBO-Edit pag. 35
Figura 14: Graph Editor pag. 36
Figura 15: Pannello Tree Viewer pag. 39
Figura 16: Gestione delle obsolescenze pag. 40
Figura 17: Pannello History Browser pag. 41
Figura 18: Term Image Display Panel pag. 44
Figura 19: Tabella riassuntiva pag. 51

1. INTRODUZIONE

1.1 Introduzione

L'evoluzione dell'informatica nel corso dei decenni non conosce arresto, sia nella moltitudine di applicazioni e di idee, sia nell'espansione dei contesti in cui interagisce. Essa si propone come la scienza che automatizza e sintetizza le informazioni, dove intuizioni di pochi, coloro che immaginavano i computer come elaboratori eterogenei e non solo calcolatori matematici, hanno agevolato la vita di molti.

La sua prerogativa fondamentale è la semplicità espressa in ogni sua forma. Un software applicativo deve essere semplice da usare, l'algoritmo migliore è quello che arriva alla soluzione con il procedimento più semplice, il sistema numerico binario stesso si avvale semplicemente di due cifre. Perfino il mondo di internet e del World Wide Web ha stravolto processi ed aspetti della società nella loro essenza e ha reso virtuale ciò che è reale, portando la comunicazione ad un click di distanza dall'utente.

Ciononostante, quando il progresso procede e la conoscenza matura, le cose si complicano e i problemi sbocciano. Ecco allora veder nascere l'esigenza di interpretare, che nel contesto del WWW si traduce nelle nozioni di Web Semantico e di Ontologia.

Il Web semantico costituisce il passo successivo al Web, "un ambiente in cui i documenti pubblicati sono associati a dati ed informazioni che ne specificano il contesto semantico in un formato adatto all'interrogazione, all'interpretazione e all'elaborazione automatica". [1] Il volume di dati che viaggia nella rete istantaneamente è immenso, nessun esperto sarebbe stato così abile da prevedere il successo esponenziale degli ultimi anni. Le informazioni allora non possono più essere a sé stanti, ma devono essere più articolate e strutturate, subentrano parole chiave capaci di sintetizzarne i contenuti e le macchine, a loro volta, devono essere in grado di riconoscerle, distinguerle e selezionarle, scartando ciò che esula dai contenuti richiesti. Si affronta un problema bivalente, legato all'associazione di dati tra loro e ad una corretta interpretazione nell'effettuare queste relazioni.

Il concetto di ontologia (dal greco *òntolos-lògos*, discorso sull'essere) affonda le sue radici nel mondo della filosofia, la prima vera e propria scienza che ha tentato di dare una risposta alle domande e di spiegare l'esperienza umana. Pertanto la sua prima necessità fu quella di applicarsi nello "studio del fondamento di ciò che esiste, del come esiste, se è solo pensabile, se è costante, universale, accertabile". [1] Ogni domanda ontologica verte attorno all'esistenza di un soggetto, di un oggetto e alle relazioni che intercorrono tra questi.

Se la filosofia descrive la realtà empirica, devono nascere dei linguaggi che possano idealizzarla. Se la filosofia offre le basi per capire la natura e rappresenta l'ambiente ideale per sviluppare le idee, le scienze applicate devono sfruttare queste conoscenze per trovare delle soluzioni e perseverare nella conquista di nuove scoperte.

L'informatica quindi può estendere il concetto di ontologia nel contesto del Web, definendola come "la descrizione esplicita di un dominio" [1] per risolvere le problematiche legate alla rappresentazione della conoscenza. Un'ontologia può essere vista come una struttura dati gerarchica che contiene entità rilevanti, compresi vincoli e relazioni che possono sussistere tra esse, in un dato dominio di esistenza. In una realtà virtuale dove le informazioni diventano sempre più difficili da gestire, catalogare, classificare, a causa della loro quantità spropositata, le ontologie si presentano come una possibile via d'uscita, definendo un vocabolario comune ed una conoscenza condivisa. Il loro obiettivo ultimo, allora, è quello di raffigurare una risorsa unica e riutilizzabile da più utenti, volta alla rimozione di obsolescenze e ridondanze di contenuti nella rete.

Considerato che l'ontologia ambisce a rispecchiare la formalizzazione ordinata della realtà, l'utente esige di strumenti che permettano di progettare e costruire questi mondi virtuali.

In questa tesi, si andranno ad esaminare degli editor per le ontologie, cercando di carpirne le caratteristiche peculiari e distintive di ognuna, ed offrirne una panoramica esaustiva e completa. Infine, andremo ad individuare dei particolari requisiti e costruiremo una discussione in cui gli applicativi verranno messi a confronto. La nostra scelta, per quanto riguarda quali pacchetti software studiare, è caduta su strumenti molto differenti tra loro, ovvero Protégé, Swoop e OBO-Edit, per proporre una visuale eterogenea del mondo delle ontologie e per sottolineare la loro diffusione in ambito Web semantico.

1.2 Brevi cenni sulle nozioni fondamentali

1.2.1 Il Web Semantico

Il Web semantico vuole rappresentare linguaggi in una forma che può essere facilmente comprensibile, interpretabile dalle macchine in un contesto, il Web, in cui le informazioni sono molteplici e di natura complessa. L'accesso a queste ultime avviene mediante l'assegnazione di specifici indirizzi, i cosiddetti URI (Universal Resource Identifier), per poter assicurare corrispondenze univoche con i dati e la loro localizzazione sulla rete.

L'obiettivo fondamentale è quello di dare alla rete la capacità di interpretare, di scegliere e di scartare i contenuti associati ad una determinata informazione, basandosi sulla formulazione di una richiesta specifica, che può concretizzarsi in parole chiave o in indirizzi lanciati in una barra di navigazione, digitata dall'utente. Subentra, allora, il concetto di *significato*, dove i dati non sono più visti come allocazione di memoria o localizzazione particolare di informazioni, ma come entità capaci di descrivere attivamente le informazioni contenute in essi, ed analogamente gli stessi collegamenti devono essere portatori dinamici delle informazioni a cui vengono associati.

Per poter raggiungere i traguardi prefissati, è necessario garantire due livelli di interoperabilità:

- *semantica*: consiste nell'abilità della comprensione dei dati da parte di un software applicativo, aspetto affidato al metalinguaggio XML grazie alla sua versatilità.
- *sintattica*: consiste nella fase di lettura dei dati e nella possibilità di ottenerne una rappresentazione sfruttabile dall'applicazione, intervengono linguaggi come OWL (Web Ontology Language), RDF (Resource Description Framework) e RDF-Schema.

1.2.2 Il Linguaggio OWL

Il Web Ontology Language è uno standard web scritto in XML, estensione del linguaggio RDF. Gli strumenti software che verranno analizzati in questa tesi supportano questo linguaggio. La scelta cade su OWL, piuttosto che RDF, poiché comporta diversi vantaggi: si avvale di un vocabolario più esteso e di una sintassi più funzionale, la sua interpretazione risulta più efficace per i dispositivi software.

Inoltre, OWL si propone come obiettivi fondamentali quelli di associare alle informazioni presenti sul Web significati precisi e privi di ambiguità, nonché di permettere alle applicazioni di creare processi che interagiscano attivamente con le informazioni Web. È un linguaggio, quindi, che lavora a livello macchina e che prevede di garantire un metodo universale per la gestione dei contenuti Web nell'ambito del Web semantico.

1.2.3 La nozione di Ontologia

Nell'ambito informatico con il termine ontologia si intende una descrizione esplicita di un dominio, una parte di mondo reale, che crea concetti ai quali vengono associati attributi, proprietà e vincoli. Essa si propone di individuare un vocabolario e una conoscenza comune che possa essere condivisa e compresa sia tra gli utenti che da sistemi software.

Inoltre, uno degli obiettivi peculiari dello sviluppo di ontologie è la possibilità di rendere i dati riutilizzabili, e perseguire l'idea di integrazione tra domini per evitare ambiguità e ridondanze.

Illustriamo ora i concetti principali che sono parte tipicamente della struttura:

- *Classi*: sono i concetti nel dominio, una collezione di elementi accomunati da caratteristiche condivise, e soprattutto si avvalgono della proprietà di gerarchizzazione.
- *Proprietà*: sono gli attributi e le caratteristiche proprie di una classe, che permettono di distinguerla tra le altre. Possono essere intrinseche, estrinseche, esprimere una componente della classe, un vincolo o una relazione ad un altro oggetto. Le proprietà seguono le strutture e le regole di ereditarietà stabilite nella gerarchia dell'ontologia. Infine, possiedono un dominio di appartenenza, che raccoglie le classi in cui sono contenute, e un range, che indica i valori plausibili di una data proprietà.
- *Individui*: sono degli oggetti generici, senza una particolare classificazione identificativa, ma che costituiscono le entità base dell'ontologia su cui costruire la struttura.

2. ANALISI DEGLI EDITOR

2.1 Protégé

L'analisi delle applicazioni software non può che partire con lo studio di Protégé, progetto *open_source* (indica un software rilasciato con un tipo di licenza per la quale il codice sorgente è lasciato alla disponibilità di eventuali sviluppatori), generato dall'università californiana di Stanford e che, attualmente, costituisce l'alternativa più esaustiva e diffusa nel panorama degli editor per le ontologie. La versione trattata in questa tesi è la 4.0 beta, per poterla scaricare basta consultare la voce dei download del sito web http://protege.stanford.edu/.

2.1.1 Layout

L'impostazione grafica costituisce sicuramente il punto di partenza di qualsiasi applicazione. La scelta degli sviluppatori di Protégé consiste in un'interfaccia grafica di alto livello, sintetica ma intuitiva. Colori, simboli, impaginazione, tutto appare studiato per un approccio che si prefissa l'obiettivo della facilità e della *confidence* nell'utilizzo, non dimenticando l'eterogeneità degli utenti che scelgono questo editor per la progettazione delle loro ontologie. Già dal lancio del programma si può evincere questo impegno all'insegna della semplicità, poiché si propone una finestra, anticipatrice di quella principale, che rappresenta l'incipit su cui articolare l'intero lavoro. In questa schermata iniziale possiamo selezionare la generazione di un nuovo progetto o l'apertura di uno già esistente. (Fig. 1)

Figura 1: Schermata iniziale

Una volta effettuato l'accesso nella finestra principale, notiamo subito due barre di navigazione, dedicate rispettivamente alle directory delle ontologie e alla ricerca di termini o entità nell'ontologia corrente, ed una suddivisione in pannelli. L'utente è libero di personalizzare la struttura a suo piacimento, creando, importando e modificando nuovi *tab*; a questo proposito, segnaliamo la possibilità di inserire dei *plugin* (ovvero dei programmi non autonomi che interagiscono con un altro programma per ampliarne le funzioni) adibiti alla visualizzazione e alla navigazione dell'ontologia, sottoforma di pannelli. Sicuramente è un aspetto positivo da evidenziare, considerati i molteplici supporti esterni a disposizione del software.

Ciascun pannello consente di interagire attivamente con l'ontologia, consentendo di selezionarne e visualizzarne una specifica prospettiva, piuttosto che di evidenziarne la parte legata alle annotazioni e quella legata all'utilizzo. Oltre ad una lista gerarchica delle entità presenti, ogni tab propone uno spazio dedicato alla descrizione di un'entità, in cui è possibile aggiungere, modificare o rimuovere informazioni e caratteristiche proprie dell'entità selezionata.

Analizziamo ora i pannelli che costituiscono il layout di base del programma:

- *Active Ontology*: riporta le informazioni inerenti all'ontologia, come annotazioni, importazioni, l'indirizzo URI e statistiche. Questi dati non sono riportati passivamente ma all'occorrenza possono essere modificati. Inoltre, possiamo consultare i codici sorgenti associati nei linguaggi RDF/XML e OWL/XML.

- *Entities*: (Fig. 2) ritenuto il pannello più importante, raggruppa classi, proprietà ed individui contemporaneamente, senza ignorare le gerarchie e i valori di ciascuna entità. Quest'ultime fanno tutte capo alla stessa radice, *Thing*.

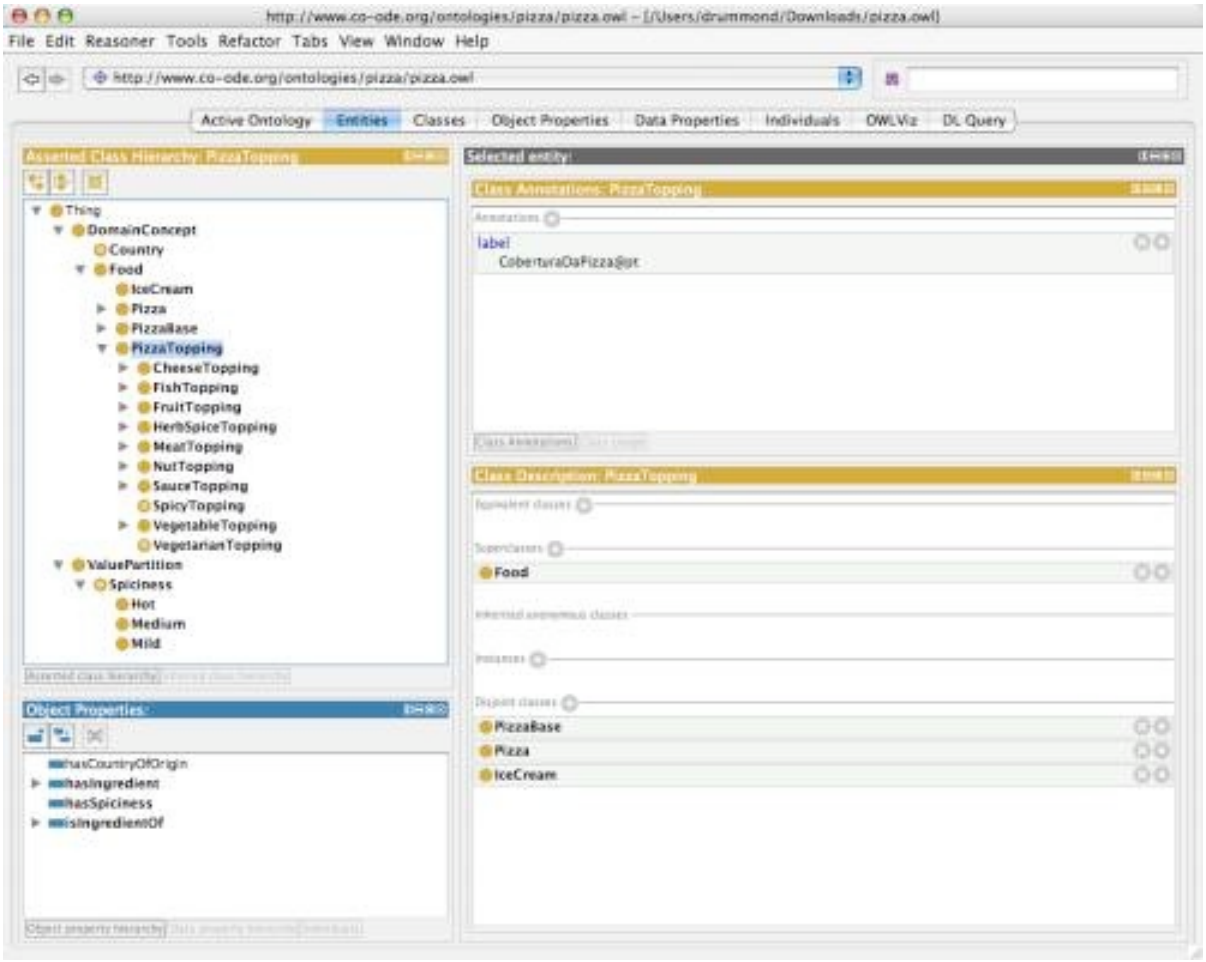

Figura 2: Pannello delle Entities

- *Classes*: mostra le classi e le loro relative strutture e gerarchie. Le visualizzazioni per tipologia presenti sono *Class Annotations* e *Class Usage*. Nel riquadro per la descrizione si possono modificare i campi di classi equivalenti, superclassi, classi anonime ereditate, membri e classi disgiunte.
- *Object Properties*: sono relazioni tra due individui. Le visualizzazioni per tipologia sono *Annotations* e *Object Property Usage*; inoltre per ogni proprietà è possibile taggare le seguenti caratteristiche: funzionale, funzionale inversa, transitiva, simmetrica, asimmetrica, riflessiva e non riflessiva. Nella descrizione possiamo indicare il dominio, il range dei valori, l'esistenza di proprietà equivalenti, disgiunte, inverse, eventuali superproprietà e restrizioni.
- *Data Properties*: (Fig. 3) riporta queste particolari proprietà, che stabiliscono relazioni tra un individuo e valori prestabiliti. Le visualizzazioni per tipologia sono *Data Property*

Annotation e *Usage Property Annotation*. Possiamo dichiarare se è funzionale e descrivere il dominio, il range, potenziali proprietà equivalenti o disgiunte, l'esistenza di superproprietà.

- *Individuals*: riassume gli individui presenti nell'ontologia. Le visualizzazioni per tipologia sono *Individual Annotations* e *Individual Usage.* Nella descrizione possiamo inserire eventuali tipi o precisare se esistono individui diversi o analoghi. Infine, possiamo usufruire del riquadro delle *Property Assertions*, dove segnalare asserzioni di *Object Properties* e *Data Properties* positive o negative.
- *OWL Viz*: pannello che ospita il plugin GraphViz, verrà descritto nella sezione dedicata.
- *DL Query*: ulteriore plugin compreso nel pacchetto software, anch'esso descritto successivamente.

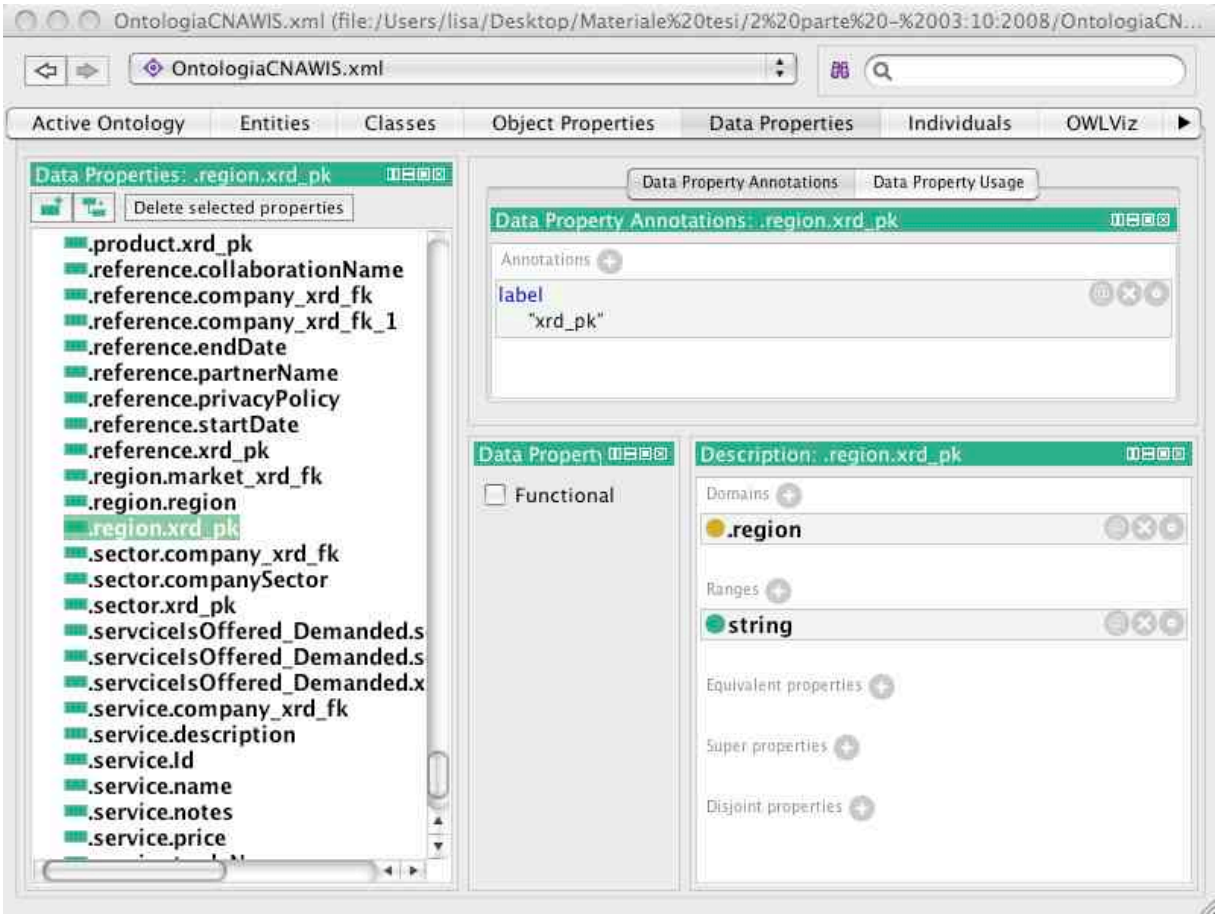

Figura 3: Pannello Data Properties

2.1.2 Creazione, modifica ed eliminazione

La finestra anticipante la struttura principale del programma rappresenta anche in questo paragrafo il punto di partenza della descrizione. Questa schermata, infatti, è a tutti gli effetti la genesi del nostro progetto, la cui scelta verte tra la creazione di una nuova ontologia e l'apertura di una già esistente, indicando la directory che la localizza o l'indirizzo URI. Il programma, inoltre, riporta alcuni dei links in cui sono reindirizzate le directory delle ontologie aperte recentemente, un accesso veloce ai nostri operati. Solo dopo aver precisato queste modalità, l'utente può accedere alla finestra principale dell'applicazione e cominciare, a tutti gli effetti, il proprio lavoro.

La sezione dedicata alle modifiche applicabili all'ontologia in analisi, è particolarmente ricca poiché Protégé propone svariati comandi oltre ai classici "taglia, copia ed incolla". In primo luogo sono presenti disposizioni precise per la creazione di entità; si possono distinguere le realizzazioni di entità generiche e, se nel momento della creazione stiamo selezionando una particolare entità, quella nuova verrà creata della medesima tipologia. Analogamente, in riferimento alla selezione corrente, si possono creare entità figlie o gemelle: le prime saranno sottoclassi o sottoproprietà, mentre le seconde saranno classi o proprietà dello stesso livello aventi il medesimo padre e le sue eventuali caratteristiche ereditate.
Per quanto concerne le classi, sono presenti opzioni additive, tra cui la duplicazione, la quale effettua una copia di una data classe selezionata includendo ipotetici legami con superclassi ed assiomi di classi equivalenti, ma senza considerare le annotazioni. Per creare queste relazioni di equivalenza, è necessario inserire assiomi, generanti classi con caratteristiche equipollenti e comprese l'unione delle loro sottoclassi. Inoltre, tra le possibili azioni, sono previste possibilità di conversione, intese in modo bivalente: data una classe, si possono spostare tutte le classi ad esse equivalenti in superclassi e dunque dividere le intersezioni esistenti; in alternativa, tutte le superclassi associate alla classe selezionata sono convertibili in una classe equivalente e, in questo caso, formano un'intersezione. Qualora si ritenesse opportuno, sono consentite le operazioni inverse per separare una classe e le sue gemelle con la genesi di un assioma che le contenga, in generale esiste una procedura analoga per tutti gli individui.
Infine, un accenno alle procedure di rimozione delle entità: avviene tramite il comando dedicato e consente di eliminare l'ultima entità selezionata, inclusi i riferimenti associati ad essa.

Bisogna precisare che in questo paragrafo ci si riferisce solamente alle opzioni applicabili all'ontologia e alle sue entità in modo generico, poiché è abbastanza immediato intuire che qualsiasi tipo di attività o interazione di fatto preveda un processo di creazione, modifica o eliminazione. Per una maggiore comprensione dei contenuti, ritengo sia più opportuno suddividere questa categoria, accorpandola parzialmente ad altri aspetti non meno rilevanti del dispositivo (situazione verificatasi anche nella sezione del Layout), e rendere così la lettura più fluida e coerente.

2.1.3 Esplorazione e visualizzazione

Esplorare un'ontologia con Protégé risulta molto intuitivo. Uno degli aspetti fondamentali della navigazione è la scelta dell'impostazione del layout, di cui abbiamo già avuto occasione di discutere, descrivendone la struttura a pannelli.
Un'ulteriore caratteristica meritevole di citazione è la barra di ricerca situata sulla finestra principale sopra i tabs, in modo da rimanere in primo piano in qualsiasi momento. Essa è in grado di effettuare una scansione globale nell'ontologia e di riportare ciascuna entità contenente la parola chiave digitata.
In ciascun tab possiamo inserire, dimensionare, modificare o rimuovere molteplici viste, raggruppate sotto alla barra *View* del programma. Quest'ultime, in buona parte, sono già presenti nel layout di default, ma la possibilità di personalizzare è estrema, nel tentativo di agevolare efficacemente l'uso del programma. La visualizzazione di base proposta è quella di dedicare, ad ogni pannello, un particolare tipo tra le entità esistenti, fornendone le caratteristiche principali. Il comando *View* stesso suddivide le varie alternative per categoria, partendo dalle Classi e proseguendo con le Data Property, le Object Property, gli Individui, le viste dedicate all'ontologia e infine due generiche alternative: la Axiom Annotation e la Query. La struttura proposta è prettamente gerarchica, ciò non toglie che si possano creare nuovi tab contenenti invece gli stessi generi per ogni entità, come per esempio un pannello adibito alle annotazioni di ogni tipo di entità o un altro disposto per le descrizioni.

Soffermiamoci ora sull'importanza che gli sviluppatori di Protégé hanno dato alle parole, intese sia come strumento di navigazione che come compromesso interpretativo tra il linguaggio tecnico e l'utente.

Per quanto riguarda il primo aspetto si evince che nella maggior parte delle viste è implementata la navigazione ad ipertesto, le parole quindi assumono il ruolo fondamentale di consentire collegamenti tra un termine e tutte le caratteristiche correlate.
Nel secondo caso, le parole si fanno traduttrici dal linguaggio tecnico informatico al linguaggio naturale, con il vantaggio di rendere il software comprensibile ad una scala più ampia di utenti. Protégé, infatti, è in grado di sostituire ai nomi delle entità (spesso una stringa non particolarmente intuitiva) con delle *label*, ovvero delle etichette generate automaticamente prendendo spunto da eventuali annotazioni, valori o frammenti dell'indirizzo URI. Se l'operazione non dovesse andare a buon fine, il programma si troverà costretto a riportare eccezionalmente l'ID dell'entità.

2.1.4 Gestione dei contenuti

Perché un dispositivo software sia ben progettato, non può permettersi di trascurare la sezione riferita alla gestione delle ontologie. Protégé, anche in questo ambito, presenta una soluzione diversificata e più che esaustiva al problema.

In primo luogo riportiamo quattro operazioni particolarmente interessanti, situate nella barra *File* del programma:

- *Gather Ontologies*: dall'inglese "raccogliere", colloca tutte le ontologie aperte in quell'istante in una cartella. È una funzione molto comoda specialmente per quelle ontologie che sono state lanciate dal Web o da librerie.
- *Export Inferred Axioms as Ontology*: letteralmente significa "esporta gli assiomi dedotti come ontologia", di fatto salva sottoforma di ontologia una gerarchia indotta dopo la classificazione dei componenti, consentendo all'ontologia originaria di non subire modifiche.
- *Ontology Libraries*: è un comando che si dedica alla gestione della locazione di ontologie comuni e condivise in più progetti dall'utente.
- *Loaded Ontologies Sources*: mostra un breve riassunto circa la locazione fisica (directory) delle ontologie aperte e caricate di recente dall'utente.

In questo caso, ho voluto elencare nel dettaglio i comandi poiché non solo rappresentano metodi di gestione, bensì soprattutto una proposta di semplificazione del lavoro e di riutilizzo di ontologie preesistenti, o perlomeno parti di esse.

Il concetto di riutilizzo è fondamentale, e rappresenta l'obiettivo più ambizioso preposto da coloro che hanno pensato all'ideazione delle ontologie come possibilità di archiviare, raccogliere e condividere informazioni. Solamente partendo da questo nucleo, le ontologie possono essere sviluppate correttamente e quindi le fasi di gerarchizzazione e catalogazione subentrano in un secondo momento.

Protégé mette a disposizione due *reasoner* per la classificazione di ontologie:

- *FaCT++*: è il successore di FaCT OWL-DL reasoner, eredita i suoi efficaci algoritmi ma si rimodernizza nella sua architettura interna e viene implementato in C++ per aumentarne l'efficienza.
- *Pellet*: è un reasoner open source in Java e supporta una completa espressività in OWL-DL, assicurando i principali servizi forniti dalla maggioranza dei reasoners e un'approfondita analisi.

L'uso di uno dei due dispositivi non avviene automaticamente né risulta essere obbligatorio. Ogni volta che vengono lanciati, inizializzano un'indagine dettagliata dell'ontologia ed effettuano una classificazione delle entità presenti. Qualora il reasoner non riesca a classificare una particolare classe, questa viene riportata in rosso sotto la superclasse *nothing*, diversamente ognuna compare all'interno della struttura sotto le rispettive superclassi.

Sono previsti ulteriori comandi che riguardano la gestione dell'indirizzo URI. Per esempio, quando vogliamo rinominare un'entità, possiamo modificarne lo *User Resource Identifier* anziché l'etichetta. Inoltre è possibile modificare l'URI dell'ontologia senza che gli URI delle singole entità subiscano modiche.
Per concludere con le operazioni legate alla gestione dell'ontologia, citiamo la possibilità di dividere o unire gli assiomi in sottoclassi, e di plasmare più ontologie creandone una nuova o sfruttandone una già esistente.

2.1.5 Plugin e supporti esterni

OWL Viz. È parte integrante del pacchetto standard di Protégé 4.0, ma richiede l'installazione di GraphViz; consente di visualizzare la classificazione delle entità in un grafo, e risulta rappresentare un supporto particolarmente interessante nonché utile nello studio delle

relazioni tra le varie entità. Il grafico di navigazione ottenuto può essere salvato ed esportato come file immagine. (Fig. 4)

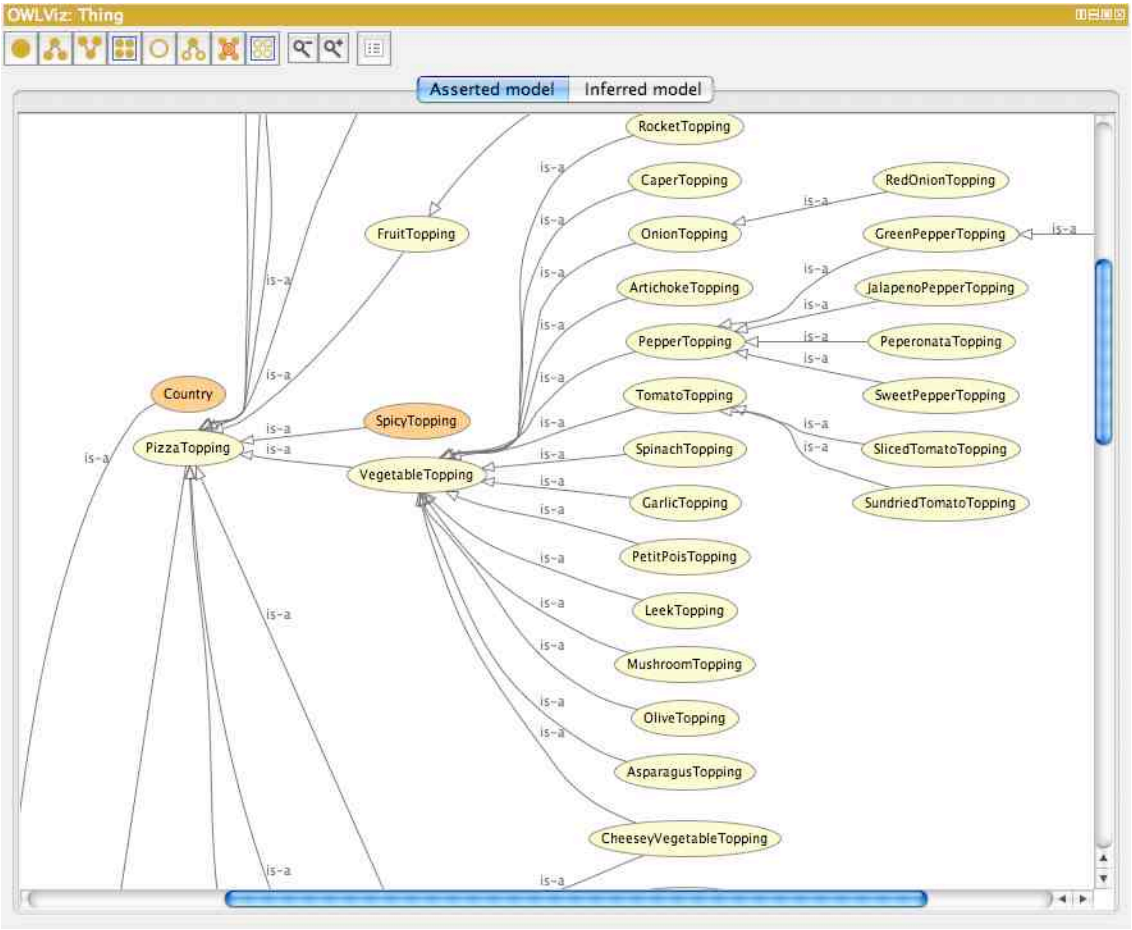

Figura 4: OWL Viz

DL Query. Secondo, e ultimo, supporto esterno che è parte del pacchetto standard di Protégé, permette di implementare delle query per controllare la correttezza della struttura dell'ontologia. Basato sulla sintassi di OWL, può lavorare solamente se l'ontologia è già classificata, dunque uno dei due reasoner a disposizione deve essere attivo quando questo supporto viene invocato. Verificata questa condizione fondamentale, si può procedere con il formulare ed eseguire query, il plugin raccoglierà tutte le rispettive informazioni di classi, proprietà o individui in un unico costrutto, chiamato *frame*, comprese le relazioni di gerarchia per poter localizzare l'entità selezionata nel contesto dell'ontologia. (Fig. 5)

Matrix. Consente una vista tabulare di classi, proprietà ed individui, mostrandone qualsiasi combinazione di annotazioni, attributi e caratteristiche. Il comando più utile è il trascinamento

di oggetti, che vengono aggiunti alla matrice sottoforma di nuove colonne, può essere usato anche per aggiungere campi di riempimento, domini o range. Inoltre, il plugin di filtri consente una visualizzazione parziale dei contenuti della matrice, in particolare è possibile eseguire un filtro sulle annotazioni in base alla lingua usata oppure sulle colonne specificando il tipo di limitazione che si intende imporre.

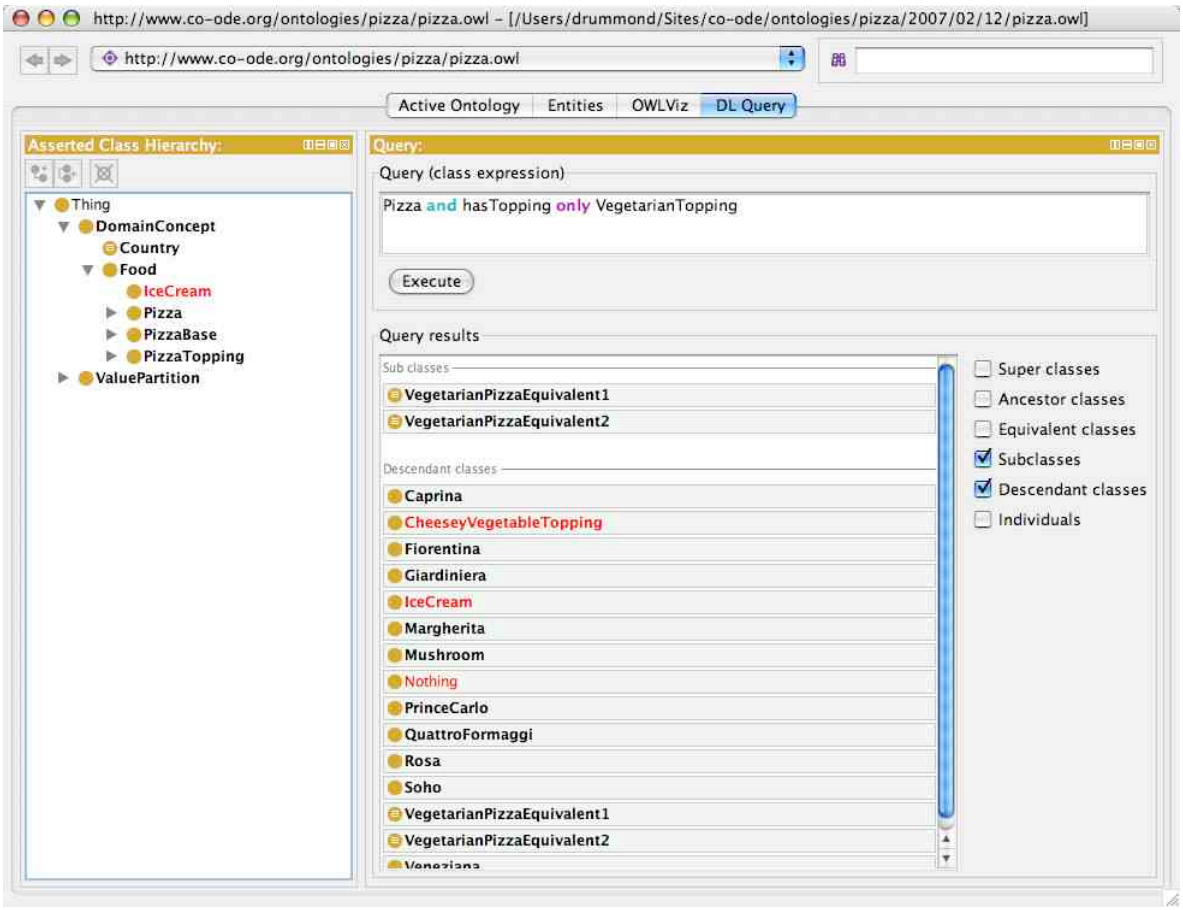

Figura 5: DL Query

Sub Ontology. Plugin ancora in fase di sviluppo, ma prevede funzioni molto utili. Permette di copiare o trascinare una gerarchia di classi da un'ontologia all'altra e consente di estrarre, anche parzialmente, una gerarchia di classi per poi allocarla in un'ontologia separata; infine copia eventuali annotazioni che erano presenti nell'ontologia nel momento in cui si decide di applicare una delle due operazioni appena descritte.

Excel Import. Il plugin carica dei file Excel o CVS nell'ontologia ed è in grado di creare delle classi che racchiudono i loro contenuti; inoltre consente di applicare restrizioni e vincoli arbitrari.

Cloud Views. Il plugin descrive la forma e il modo d'uso dell'ontologia. Il principio di base è semplice: implementa una finestra in cui vengono riportate le entità principali, più il carattere usato è grande, più il punteggio associato sarà alto. I criteri di valutazione usati di un singolo termine sono la frequenza dell'uso, la posizione nella gerarchia, il numero di sottoentità che contiene, il numero di entità gemelle, il numero di superentità anonime con cui è relazionato. (Fig. 6)

Figura 6: Cloud Views

Bookmarks. Dall'inglese segnalibro, crea una vista dove si possono trascinare classi e proprietà, che potranno essere consultate in qualsiasi momento con un click. Inoltre, le stesse entità vengono salvate come annotazioni, per poter essere riutilizzate.

Cardinality View. Il plugin genera una vista alternativa che mostra le restrizioni in relazione alla loro cardinalità. È munito di suggeritore.

Taxonomy Cut+Paste. Il plugin implementa due viste: una delle sottoclassi, l'altra dei nomi discendenti. Nella prima viene illustrata la gerarchia della classe selezionata, nella seconda viene riportata la medesima gerarchia senza rientranze per poter essere inserita in documentazioni sottoforma di immagine.

Outline/Esistential Tree. Il plugin offre un'alternativa per la visualizzazione della struttura dell'ontologia, in cui la gerarchia non ne costituisce più la spina dorsale. Nella vista *Existential Tree* l'asse di riferimento può diventare una proprietà specificata. La vista *Outline Tree* invece si riferisce all'*Existential Tree* costruendo una vista della classe selezionata, basandosi sulla sua struttura e sui nuovi riferimenti.

Browser (OWLDoc). Plugin composto da due parti: *OWLDoc View* e *OWLDoc Export*. La prima è una vista dinamica che crea una presentazione HTML di qualsiasi classe, proprietà o individuo selezionato e viene riaggiornata in caso di modifiche; mentre la seconda prevede di esportare la presentazione in una directory, pronta per un essere aperta da un browser una volta finita. OWLDoc raccoglie una serie di pagine HTML statiche in modo tale da poter essere pubblicate su internet o distribuite agli altri utenti.

The Protégé Nerd. Il plugin suggeritore di Protégé, il corrispettivo di *Clippy* in Microsoft Word.

TerMine. Il plugin usa degli strumenti di "estrazione" per estrapolare dei termini da un testo e prevede un'interfaccia in grado di inserire questi termini nell'ontologia. L'accesso a questo supporto avviene tramite un servizio Web via internet.

OWL lint. È una struttura pensata affinché sia un modo flessibile per definire e usare testi sullo sfondo dell'ontologia.

Annotation Search View. Vista per cercare delle entità in base a parole chiave presenti nelle loro annotazioni, in alternativa la ricerca può essere effettuata sfruttando come filtro gli indirizzi URI delle entità.

Annotation Template View. Vista che può essere configurata per mostrare un set di campi predefiniti dedicati ad annotazioni per qualsiasi entità. Velocizza la stesura delle annotazioni, la loro modifica nonché la loro navigazione.

Change View. È una vista adibita al *debug*, serve per verificare eventuali cambiamenti che potrebbe subire il modello standard durante la costruzione e l'importazione di un dato plugin.

SKOSEd. È un plugin che permette di creare e modificare un dizionario dei sinonimi, che si riallaccia al progetto *Simple Knowledge Organisation System*, sviluppato e riconosciuto tra le attività ufficiali del Web semantico (si può consultare il sito www.w3c.org) e si prefigge come obiettivo quello di pubblicare vocabolari strutturati.

Beanshell View. Vista che propone un accesso diretto a Protégé per formulare interrogazioni o manipolazioni, tramite un pannello che permette l'implementazione di linguaggio codice.

2.2 Swoop

Il secondo dispositivo proposto in questa tesi è Swoop, un applicativo open source sviluppato dall'università americana del Maryland, scalabile e distribuito, è considerato l'unico editor per ontologie disponibile che è implementato interamente ed esclusivamente per OWL e con un approccio *Web-oriented*. La versione trattata è la 2.3 beta 4, consultabile al sito web http://code.google.com/p/swoop/.

2.2.1 Layout

Il punto di forza di Swoop vuole essere proprio nella novità del layout, poiché una volta lanciato il programma, possiamo notare immediatamente il rimando grafico alla classica interfaccia di un browser Web, con l'obiettivo di rendere l'ambiente di lavoro familiare all'utente. Attualmente, in realtà, molti degli applicativi software per le ontologie hanno adottato questo approccio grafico, compresi quelli analizzati in questa tesi, ma sicuramente Swoop è stato il primo ad avere questa geniale intuizione.

Il programma si sviluppa principalmente in una finestra, suddivisa in diversi settori. Dedichiamo un accenno alla *Option Bar*, costituita da tre comandi muniti di *flag*, e alla barra di navigazione, chiamata *Navigation SideBar*, che discuteremo nei paragrafi successivi, ma che costituiscono un punto di riferimento visivo per l'utente nella disposizione delle componenti.
Proseguendo con l'analisi, notiamo un piccolo riquadro nel quale vengono riportati i nomi delle ontologie caricate, ed introdotte nel pannello principale con due schede:

- *Ontology Info*: riporta le informazioni generali associate all'ontologia, tra cui il nome, eventuali annotazioni, il conteggio delle entità presenti e delle statistiche relative a particolari caratteristiche.
- *Species Validatio*n: illustra tutte le entità presenti e descrive le caratteristiche ad esse affiliate, o meglio la natura da loro assunta nell'ontologia (classe, proprietà o individuo).

Nella finestra è presente un ulteriore riquadro che espone le entità presenti nell'ontologia, con tre diverse alternative di visualizzazione. Possiamo interpellare rispettivamente le gerarchie delle classi o delle proprietà, proposte con una struttura ad albero, oppure consultare una lista

completa di tutte le entità esistenti, ordinata alfabeticamente, nella quale è possibile applicare vincoli che consentano visualizzazioni parziali. Ad ogni termine, inoltre, viene associato un simbolo che ne identifica la tipologia (classe, proprietà o individuo).

Il pannello principale della finestra propone schede differenti nel momento in cui si seleziona una generica entità, in ciascuna viene rappresentata con una particolare sintassi: *Concise Format, Abstract Syntax, RDF/XML e Turtle*. Lo scopo di questa scelta è quello di soddisfare la versatilità di esigenze che sorgono da un'utenza eterogenea.

2.2.2 Creazione, modifica ed eliminazione

Se in Protégé ciò che risalta è l'esaustività in ogni singolo aspetto nella trattazione delle ontologie, in Swoop sicuramente prevale la semplicità, a partire proprio dalle operazioni di base che non possono mancare in un editor.

Innanzitutto, bisogna decidere come cominciare il nostro lavoro, scegliendo tra la creazione di una nuova ontologia o l'utilizzo di una già esistente, che possiamo rintracciare in locale "sfogliando" nelle nostre directory oppure inserendo l'indirizzo URL nella barra di navigazione. Inoltre Swoop permette di sviluppare i nostri progetti in un proprio formato, detto *Workspace*, adibito a considerare più ontologie contemporaneamente per il medesimo lavoro. La chiusura di determinate ontologie può essere effettuata selezionando l'ontologia desiderata e proseguendo l'operazione con il comando *Remove Ontology*.

La parte che concerne l'interazione attiva di modifica è particolarmente interessante, ogni comando è proposto sottoforma di bottone nei relativi pannelli. Possiamo aggiungere classi, proprietà o individui all'entità selezionata, rimuoverli o rinominarli.
Inoltre si può inserire un GCI, ovvero un *Generic Concept Illusion Axiom*, strumento molto intuitivo usato per rapportare classi e proprietà fra loro e per specificare le modalità della loro interazione. Il comando apre una finestra a sé stante che riporta gli elenchi delle entità e le relazioni consentite.
In realtà, la parte in assoluto più caratteristica di questo paragrafo è rappresentata dalla *Option Bar*, un riquadro sottostante alla barra di navigazione che riporta tre semplici comandi con i quali è davvero possibile cambiare radicalmente la nostra ontologia:

- *Show Inherited*: comando con il quale è possibile visualizzare ciò che è stato ereditato nell'entità selezionata da parte di altri componenti.
- *Changes/Annotations*: riporta l'evoluzione dei cambiamenti avvenuti durante il processo di sviluppo e modifica dell'ontologia, inoltre propone le annotazioni presenti, dette *Annotea Annotations*. Entrambi gli aspetti verranno proposti in paragrafi successivi ed esposti in modo più approfondito.
- *Editable*: rappresenta sicuramente il comando più importante. Se selezionato, nella scheda *Concise Format* compaiono dei comandi in ipertesto grazie ai quali, con un semplice click, è possibile aggiungere, modificare e cancellare dati associabili a qualsiasi caratteristica di un'entità. Allora per le classi potremo interagire su eventuali annotazioni, sottoclassi o superclassi, modificare i campi referenti a classi equivalenti o disgiunte, indicare intersezioni, unioni, domini o range, perfino indicare eventuali istanze. (Fig. 7)

  Per quanto riguarda le proprietà, distinte in proprietà degli oggetti e proprietà dei dati, possiamo inserire domini e range, superproprietà e sottoproprietà, equivalenze esistenti, annotazioni ed attributi. Infine, i comandi previsti per gli individui sono quelli che consentono di lavorare sulle asserzioni di *Datatype* e *Object*, sulle annotazioni e le istanze, oppure sull'indicazioni di elementi analoghi o diversi da quello selezionato.

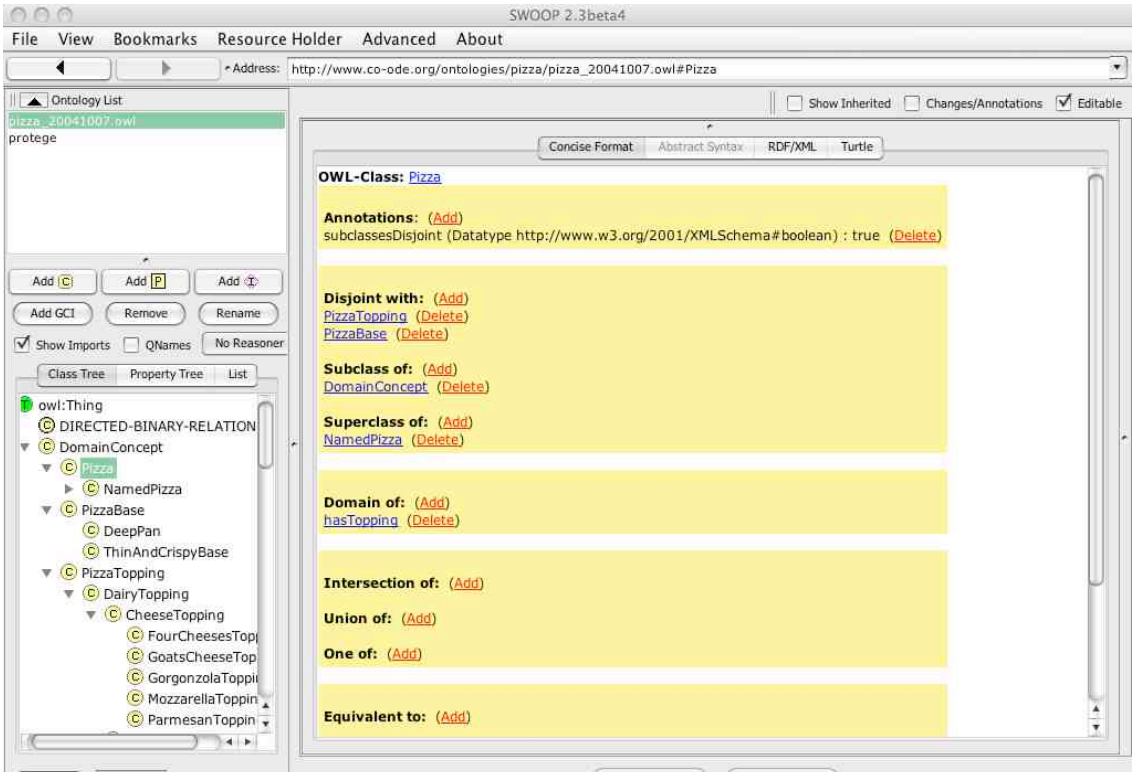

Figura 7: Pannello con il comando *Editable* attivo

2.2.3 Esplorazione e visualizzazione

La ricerca dell'essenzialità di Swoop viene rispettata anche in questo frangente, senza dimenticare l'aggiunta di qualche idea originale e pensata per agevolare l'uso del programma agli utenti.
Nella finestra principale è presente un campo dedicato alla ricerca delle parole chiave, l'esito dell'operazione si manifesta in una nuova finestra dove vengono riportate tutte le entità in cui compare il termine digitato.

Per ciò che riguarda l'esplorazione dei contenuti all'interno dell'ontologia, Swoop si concentra prepotentemente in una navigazione ad ipertesto, trasformando la navigazione in un vero e proprio *browsing*. Inoltre, come abbiamo già avuto modo di discutere, l'approccio ipertestuale è fortemente sfruttato per interagire attivamente con il nostro progetto. Le parole, dunque, sono prima di tutto uno strumento di analisi della nostra ontologia, ed il significato sintattico assunto passa in secondo piano.
L'importanza del linguaggio è una tematica che ricompare anche nel pannello di visualizzazione principale, dove si possono visualizzare le entità selezionate con diverse modalità in quattro schede:

- *Concise Format*: è un linguaggio semplificato e molto vicino a quello naturale, in cui i concetti vengono esposti sinteticamente e senza reminescenze al codice. Si potrebbe ricondurre al concetto di *label* in Protégé.
- *Abstract Syntax*: dall'inglese "sintassi astratta", si intende una rappresentazione dei dati universalmente riconosciuta ed interpretabile da una diversità molteplice di macchine. In questo caso, gli sviluppatori si riferiscono ad OWL, considerato il linguaggio fondamentale tra quelli proposti. (Fig. 8)
- RDF/XML: per esteso *Resource Description Framework/eXstensible Markup Language*, è un linguaggio, un metodo generale (RDF) per modellare dati attraverso diversi formati sintattici, in questo caso il metalinguaggio XML. (Fig.9)
- *Turtle*: per esteso *Terse RDF Triple Language*, è un formato in grado di convertire i grafi RDF in un linguaggio proprio, non fa parte di nessuno standard ufficiale ma si è conquistato una buona popolarità tra gli sviluppatori di Web semantico.

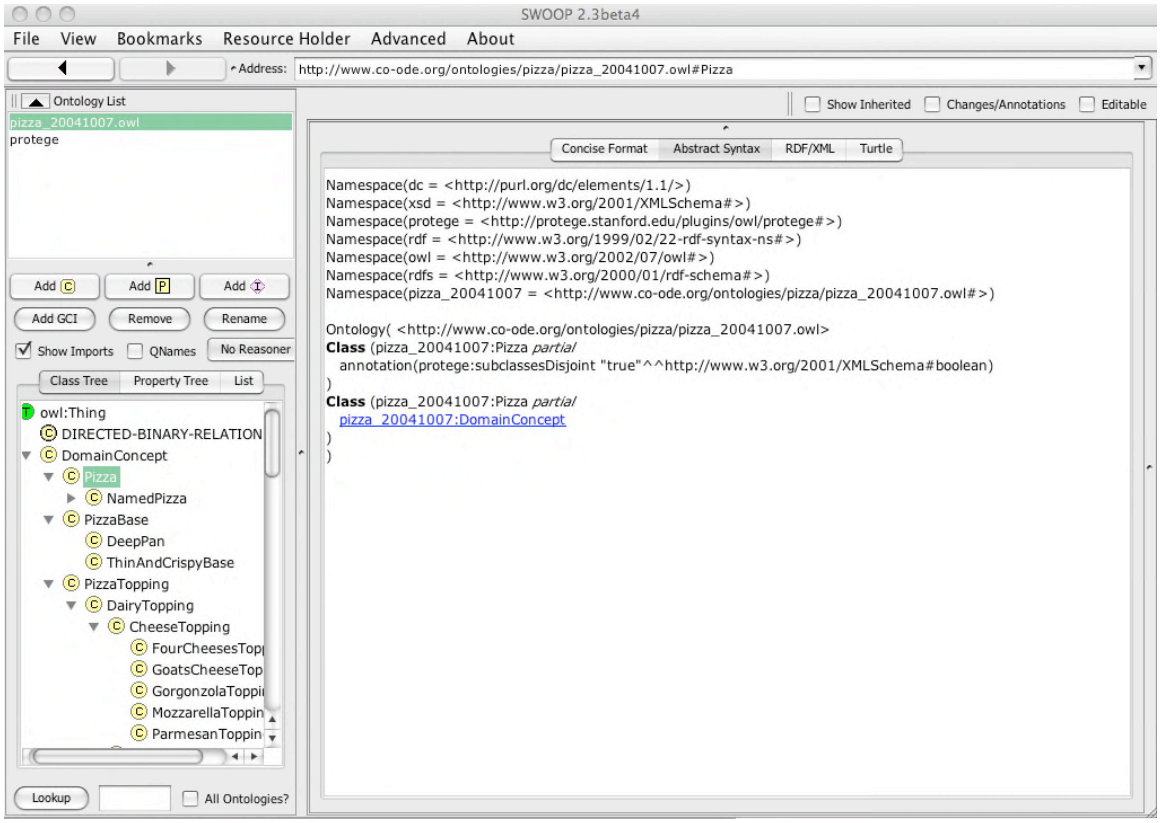

Figura 8: Pannello Abstract Syntax

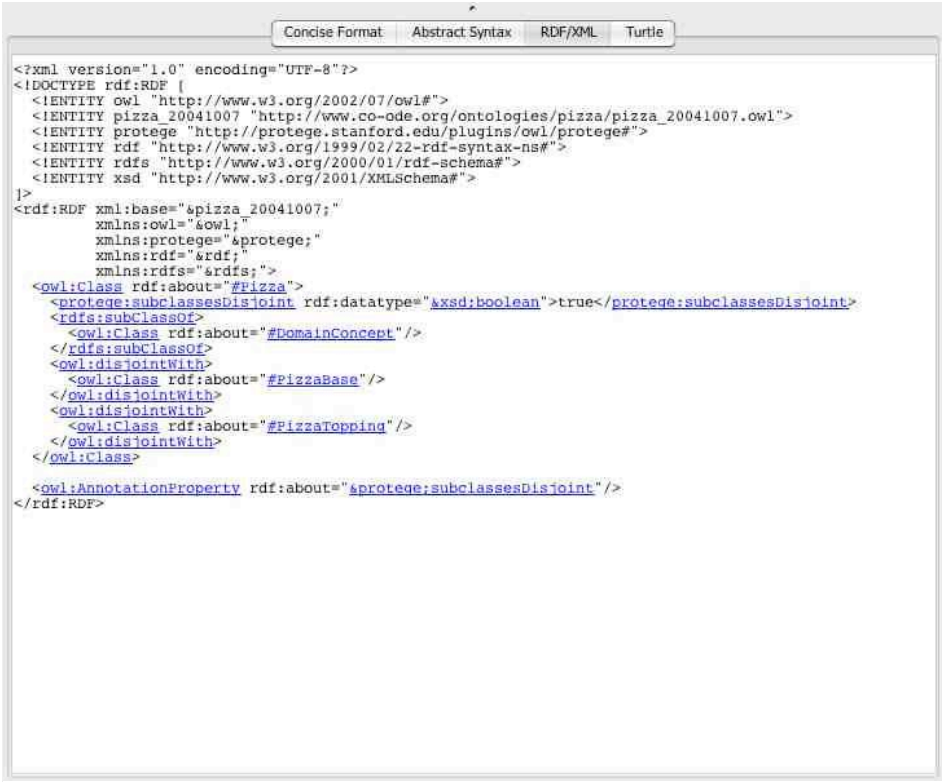

Figura 9: Pannello RDF/XML

Questa politica di molteplicità nel linguaggio è sicuramente molto costruttiva, poiché permette al programma di rivelarsi come una risorsa versatile a tutte le esigenze, favorendo in questo modo una maggior adozione del prodotto da parte degli utenti. Inoltre, è in grado di contribuire positivamente alla diffusione della filosofia del riutilizzo e della condivisione nell'ambito delle ontologie.

Un ulteriore aspetto che rende la navigazione più semplice è il comando *Bookmarks*, tradotto letteralmente "segnalibri". Esso costituisce una delle voci nella barra a comparsa del software, in cui vengono memorizzati gli indirizzi di ontologie, per poter avere un accesso ed un caricamento rapido dei dati sul nostro applicativo. Swoop ne propone diversi inseriti di default, ma l'utente è libero di organizzare i suoi segnalibri come meglio crede aggiungendone dei nuovi, eliminandone oppure semplicemente disporli in ordine alfabetico. Questa funzione non può che ricordarci la volontà degli sviluppatori di riprodurre un ambiente familiare per chi utilizza il programma, difatti questa caratteristica rimanda fortemente al pannello "Preferiti", componente immancabile di tutti i browser Web. (Fig. 10)

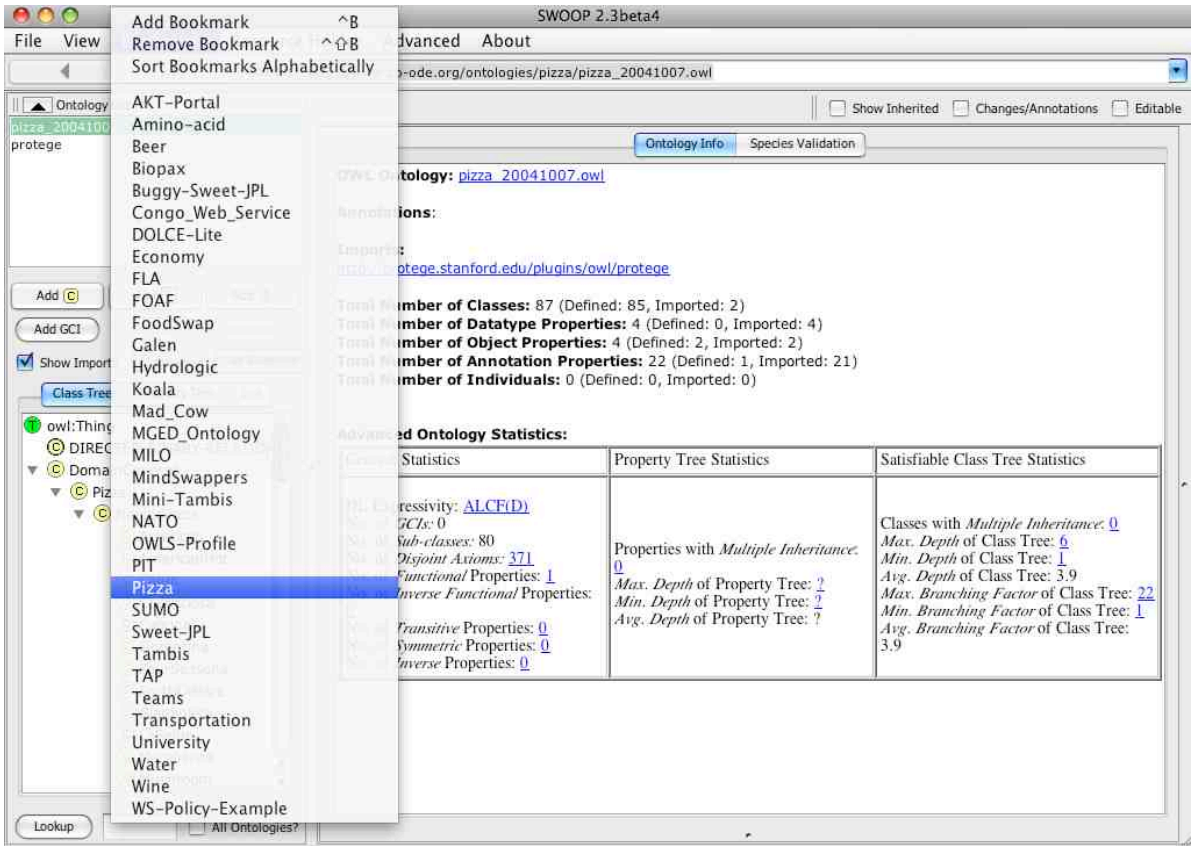

Figura 10: Comando *Bookmarks*, nello sfondo il pannello Ontology Info dell'ontologia selezionata

Infine, analizziamo i due comandi presenti nella barra *Advanced* che si rivolgono all'esplorazione e alla visualizzazione dei contenuti.
Il primo corrisponde alla voce di *Fly The MotherShip* e crea una rappresentazione grafica molto singolare della nostra ontologia; si tratta di una raffigurazione ad insiemi, dove le nostre entità sono esplicitate con la forma del cerchio e, a seconda delle loro relazioni, ogni singola entità può intersecare, contenere o essere contenuta in altri cerchi.
La seconda, invece, è titolata *Show Class Expression Table* e si limita a raccogliere informazioni circa espressioni usate nel programma, facendone il conteggio e proponendo statistiche piuttosto rudimentali.

2.2.4 Gestione dei contenuti

La gestione dei contenuti delle ontologie costituisce in Swoop un momento di analisi molto ricco. Possiamo aggiungere che anche in questo caso, così come nei paragrafi discussi finora, sono proposte idee molto diverse rispetto a quelle viste in Protégé e che, seppur meno rigorose, non sono affatto scartabili.

Innanzitutto, a fronte di analisi di ontologie complesse ed articolate, Swoop ci aiuta con il *Resource Holder*, una finestra in cui possiamo salvare delle entità di cui non vogliamo perdere traccia, la loro permanenza è statica e possono essere rimossi o sostituiti solo manualmente. Questo pannello è utile se si vogliono effettuare confronti veloci tra analogie e differenze tra le terminologie usate, oppure per rintracciare agevolmente entità che vengono usate assiduamente. All'interno di questa struttura, oltre all'inserimento e alla rimozione, è consentito esportare una certa entità selezionata in due modalità: in un primo caso, viene semplicemente duplicata nell'*holder*, mentre nel secondo viene copiata sia nell'*holder* che nella lista delle ontologie caricate.
Proseguendo con lo studio del programma, notiamo diverse operazioni legate alla gestione del nostro lavoro sono situate nella barra *Advanced*. Per prima cosa, è possibile esportare entità con il medesimo procedimento svolto nel *Resource Holder* (Fig. 11); inoltre, si può effettuare un comando in grado di dividere l'ontologia o di eseguire una partizione automatica, qualora si ritenesse opportuno.

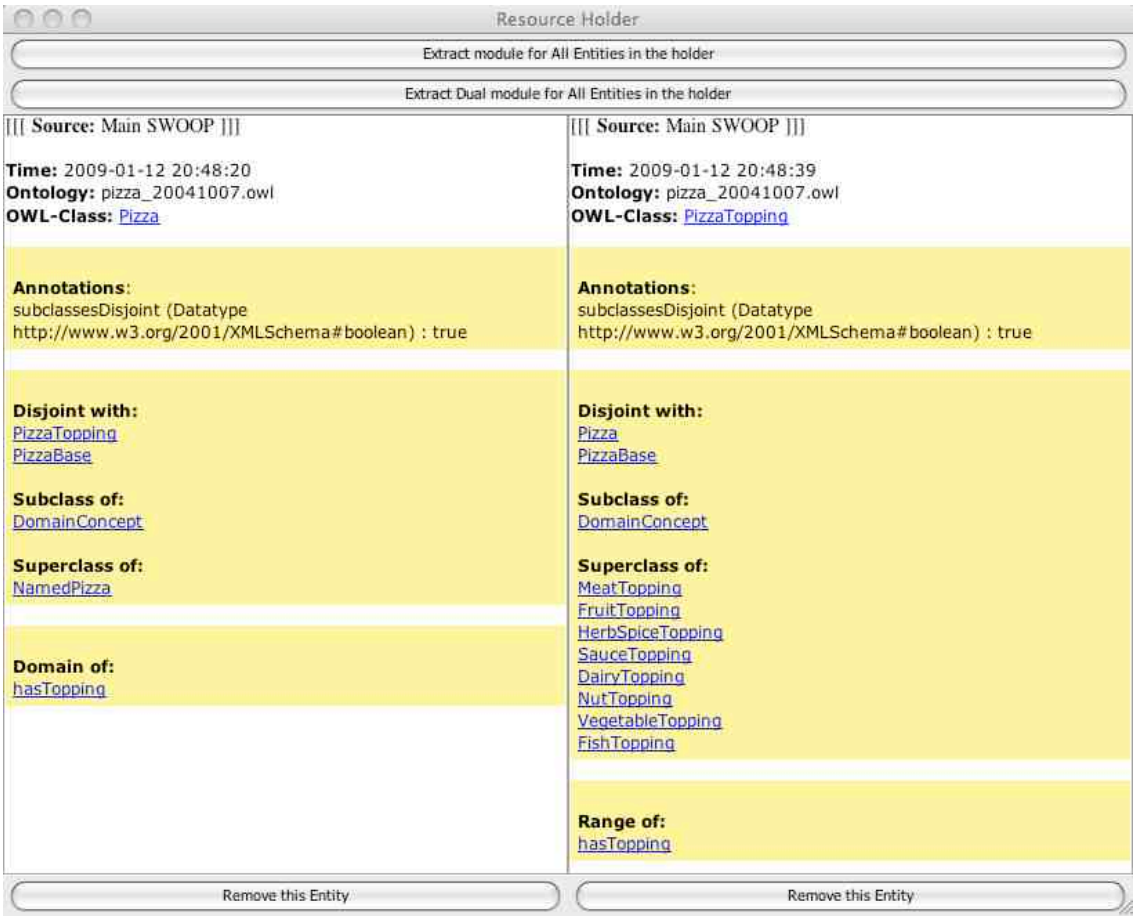

Figura 11: Pannello Resource Holder

I reasoner a disposizione per il controllo e la classificazione dei dati sono due:

- *Pellet*: è un plugin che fa parte del pacchetto standard del dispositivo, considerato uno tra i più completi. Non ci soffermeremo oltre poiché è un reasoner presente di default anche in Protégé ed è già stato trattato. In merito, ci limitiamo a citare il comando *Pellet Query* (Fig. 12), interpellabile dalla barra *Advanced*, che consente di formulare delle interrogazioni in linguaggio RDQL. I risultati visualizzati sono muniti di un collegamento ipertestuale con la struttura ad albero proposta nella finestra principale, se selezionati vengono aperti e localizzati nella gerarchia.
- *RDFS-like*: basato sulla semantica RDFS, è più leggero e veloce rispetto a *Pellet*. Nonostante possa risultare inadeguato ed approssimativo per certi problemi, poiché non considera la totalità delle soluzioni possibili, risulta comunque un'utile e veloce alternativa, durante la costruzione dell'ontologia.

Figura 12*:* Pellet Query

I due dispositivi coprono le diverse necessità degli utenti e da ciò che è emerso possiamo affermare che possiedono qualità radicalmente differenti. Tuttavia, entrambi soddisfano le caratteristiche fondamentali di realizzabilità, classificazione e completezza.

In aggiunta, i due *reasoner* sono in grado di supportare delle operazioni dedicate al controllo dell'ontologia, alla ricerca di inconsistenze nelle definizioni dei concetti. Solitamente, i dispositivi riportano quali classi non risultano soddisfacenti, ma senza spiegarne il motivo ed indicare eventuali entità ad esse collegate che potrebbero subire alterazioni a causa di queste difettosità.

Queste problematiche allora portano al conseguimento di una soluzione ausiliaria ai reasoner, che è quella di un processo di *debugging*. La tecnica adottata in Swoop per la diagnosi delle entità è chiamata *Blackbox*, dove i *reasoner* assumono il compito di formulare un certo numero di domini e la struttura dell'ontologia è interpellata per cercare di isolare l'origine del problema. In caso di conflitto, il programma mostra nel campo di descrizione dell'entità

incriminata la causa dell'evento usando un linguaggio semi-formale, inserendo degli assiomi considerati responsabili.
A questo proposito, nella barra *Advanced* possiamo applicare i comandi *Debugging/Explenation*, *Run Debug Tests* e *Repair Ontology*.

Quando le nostre ontologie hanno bisogno di una buona manutenzione, oltre che al controllo della correttezza delle sue entità, possiamo usare il *Version Control*. Questo strumento consente di archiviare versioni differenti della medesima ontologia e il suo raggio di azione comprende due tipi operazioni, i *changelog* e i *checkpoint*. I primi esplicitano il percorso evolutivo dell'ontologia, ovvero riportano i cambiamenti e le modifiche avvenute cronologicamente; mentre gli altri permettono di passare direttamente da una versione all'altra mostrando tutta la varietà di modelli esistenti. Sia i *changelog* che i *checkpoint* possono essere salvati in corrispondenza di tre livelli diversi: entità, ontologia o *workspace*, i quali ne esprimono l'ambito di applicazione.

Infine, analizziamo l'ultimo argomento legato alla gestione dei contenuti, quello del riutilizzo dei dati attraverso le importazioni, che costituisce un problema ancora in fase di risoluzione. In Swoop tutti i costrutti in RDFS/OWL possono essere usati per relazionare entità a diverse ontologie, ma allo stato attuale la semantica per collegare le entità dipende strettamente da un'importazione reciproca tra ontologie. Il modo per poter riutilizzare queste entità esterne è quello di collegarle direttamente al progetto interessato, o in alternativa quello di collegare l'entità esterna ed importare interamente l'ontologia di cui fa parte. Gli utenti, quindi, sono obbligati ad utilizzare dei collegamenti al posto di pure importazioni per poter riutilizzare entità già esistenti in altri lavori, oppure sono costretti ad associare a questo evento l'intera semantica dell'ontologia. Al momento operare con delle importazioni parziali in Swoop non è possibile, sebbene esistano soluzioni non ufficializzate e direttamente implementate nel programma. Una prima possibilità si concretizza con un "semplice copia-incolla" delle parti interessate da un'ontologia all'altra, mentre una soluzione più elegante potrebbe consistere nel partizionare l'ontologia esterna, preservandone la semantica, e quindi importare solo le parti significative. In quest'ultimo caso si sta formando un percorso per progettare algoritmi che possano essere inseriti nel dispositivo, creando la figura delle *E-connections*: un'ontologia che interagisce con diversi domini viene partizionata in modo che ogni sub-ontologia si riferisca ad un singolo dominio. In questo modo sarà possibile effettuare importazioni parziali.

2.2.5 Plugin e supporti esterni

Swoop sfrutta i plugin soprattutto per le operazioni di controllo svolte dai reasoner, come abbiamo già visto, e per processi interpretativi nell'analisi dell'ontologia. Questa politica è stata adottata per incoraggiare gli sviluppatori esterni a contribuire lo sviluppo del software.

Ontology Renderer Plugin. Sono supporti esterni dedicati all'interpretazione dell'ontologia, in grado di leggere l'espressività della *description logic* e il numero di entità presenti, in più individuano le annotazioni esistenti. Infine riconoscono i costrutti logici che determinano a quale livello delle specie OWL appartiene l'ontologia.

Annotea Framework. È un plugin che adotta l'idea di separare le annotazioni nelle ontologie dal nucleo delle ontologie stesse, permette l'uso di un vocabolario RDF specifico estendibile alle annotazioni e procura un protocollo per la loro pubblicazione in server interni al gruppo Annotea. In questo modo crea una nuova classe di annotazioni, ovvero la Annotea Schema, che riporta tutti i cambiamenti dell'ontologia e, qualora alcune di queste modifiche non risultino applicabili, il plugin avvisa con un messaggio di avvertimento.

In conclusione, citiamo due plugin ancora in fase di elaborazione, ma che in futuro potrebbero rivelarsi molto utili come supporti ausiliari all'applicativo software, ovvero *Natural Language Entity Renderer* e *Ontology Graph Visualization Plugin*.

2.3 OBO-Edit

Il terzo dispositivo che andremo ad analizzare è OBO-Edit, una componente di un progetto più esteso chiamato Gene Ontology. Si tratta di uno studio in cui diverse università danno il loro contributo per raccogliere la più vasta terminologia possibile in campo scientifico-biologico, in particolare OBO-Edit cura tre argomenti precisi, ovvero i processi biologici, le componenti cellulari e le funzioni molecolari.

Questo applicativo è un tool per le ontologie a tutti gli effetti, ma, diversamente dagli altri analizzati in questa tesi, è stato pensato e sviluppato perché venga utilizzato in un ambito ben determinato, da un target ristretto di utenti e per conseguire obiettivi mirati nella ricerca scientifica. Esso vuole rappresentare un modo per raccogliere tutte le informazioni possedute di un dato argomento in un unico archivio, inserito in un dominio pubblico nella rete per poter essere consultato ed eventualmente modificato.

La versione trattata in questa tesi è la 2.00 beta 49, il sito web di riferimento che si può interpellare è http://www.geneontology.org/index.shtml#downloads.

2.3.1 Layout

Abbiamo sempre sottolineato l'importanza per un editor di ontologie di possedere un layout estremamente semplice ed intuitivo, in questo frangente diventa caratteristica fondamentale per la buona riuscita del software, poiché sappiamo essere indirizzato sicuramente ad un'utenza che non svolge alcun tipo di attività nel campo tecnico-informatico.

La grafica di OBO-Edit si concentra in un'unica finestra, nella quale è consentita la più totale personalizzazione grazie ad una libera scelta dei pannelli da collocare, ed eventuali schede situate in un medesimo pannello. Oltre a quelli proposti dal programma, è possibile inserirne nuovi che racchiudono dei plugin importati; inoltre, l'utente è in grado di salvare o eliminare il layout corrente a suo piacimento. (Fig. 13)

Ciascun pannello è munito di comandi precisi, collocati in alto a destra in ogni singola finestra, riconoscibili attraverso particolari simboli:

- *Configure*: comando dedicato alla configurazione e alla personalizzazione dei criteri di utilizzo del singolo pannello. Il simbolo associato è quello della chiave inglese.

- *Display help*: questo comando permette di ricondursi alla guida per gli utenti nella sezione che concerne quel particolare pannello, nella quale vengono spiegate le sue caratteristiche peculiari. Il simbolo associato è il punto interrogativo.
- *Save*: con questa operazione possiamo salvare, sottoforma di formato immagine, un'istantanea in una directory esterna di ciò che è visualizzato del pannello. Il simbolo associato è la macchina fotografica.
- *Dock/Undock*: comando che estrae il pannello selezionato in una finestra a sé stante oppure lo incorpora nella finestra principale. Il simbolo associato è una freccia con la punta rivolta verso l'alto.
- *Minimize*: è il classico comando "Riduci ad icona", il simbolo associato è un trattino.
- *Maximize*: questo comando permette di ingrandire le dimensioni del pannello a tutto schermo. Il simbolo associato è un quadrato.
- *Close*: operazione che chiude il pannello o la scheda selezionati. Il simbolo associato è una croce.

Per ciò che riguarda i contenuti dei singoli pannelli, ci avvaliamo di analizzarli man mano nei prossimi paragrafi.

2.3.2 Creazione, modifica ed eliminazione

Dati gli obiettivi di condivisione e riutilizzo delle informazioni, OBO-Edit già nella fase preliminare presenta una profonda diversità se confrontato a Protégé e Swoop. L'utente può decidere di iniziare il proprio lavoro creando una nuova ontologia, oppure caricando quella preesistente che raccoglie il progetto finora sviluppato. Le modalità di accesso sono molteplici, infatti egli può decidere di scaricare dal sito Web determinati pacchetti referenti a precise parti dell'ontologia (i quali vengono assiduamente aggiornati e sottoposti a manutenzione dagli sviluppatori), oppure può allacciarsi all'indirizzo ftp://ftp.geneontology.org/pub/go. Personalmente, trovo la seconda alternativa molto più efficace, poiché attraverso il comando *Load Ontology*, situato nella barra *File*, si può caricare l'intera ontologia o una parte di essa, in più l'indirizzo ftp rappresenta un database di informazioni eterogenee che esulano dall'ontologia stessa, come una cartella dedicata ai tutorial e all'apprendimento, delle versioni snelle dell'ontologia adatte ai più svariati requisiti hardware, un archivio contenente dei dati vecchi e ancora diverse sezioni mirate ad una conoscenza più approfondita del pacchetto software.

Dopo aver scelto come cominciare, OBO-Edit interagisce con le informazioni trattate attraverso i *Data Adapter*, moduli di codice che le convertono in file con formato *.obo*; esse possono essere di natura eterogenea e comprendono le classiche entità presenti in un'ontologia, compreso il materiale in linguaggio OWL. L'unica condizione che il programma si impone è quella in cui i file possono essere letti da un disco locale o un indirizzo URL, ma possono essere scritti, e quindi salvati, solamente su disco fisso, isolando così la fase di aggiornamento del database condiviso in un secondo momento.

Più in generale, l'operazione svolta dai *Data Adapter* costituisce una traduzione dei dati in entrata e in uscita dall'applicativo in uno schema semplificato rispetto ai linguaggi tecnici, per verificarlo basta consultare i file di testo immagine che OBO-Edit crea nella directory locale, in cui i termini sono elencati in modo "pulito" e sintetico.

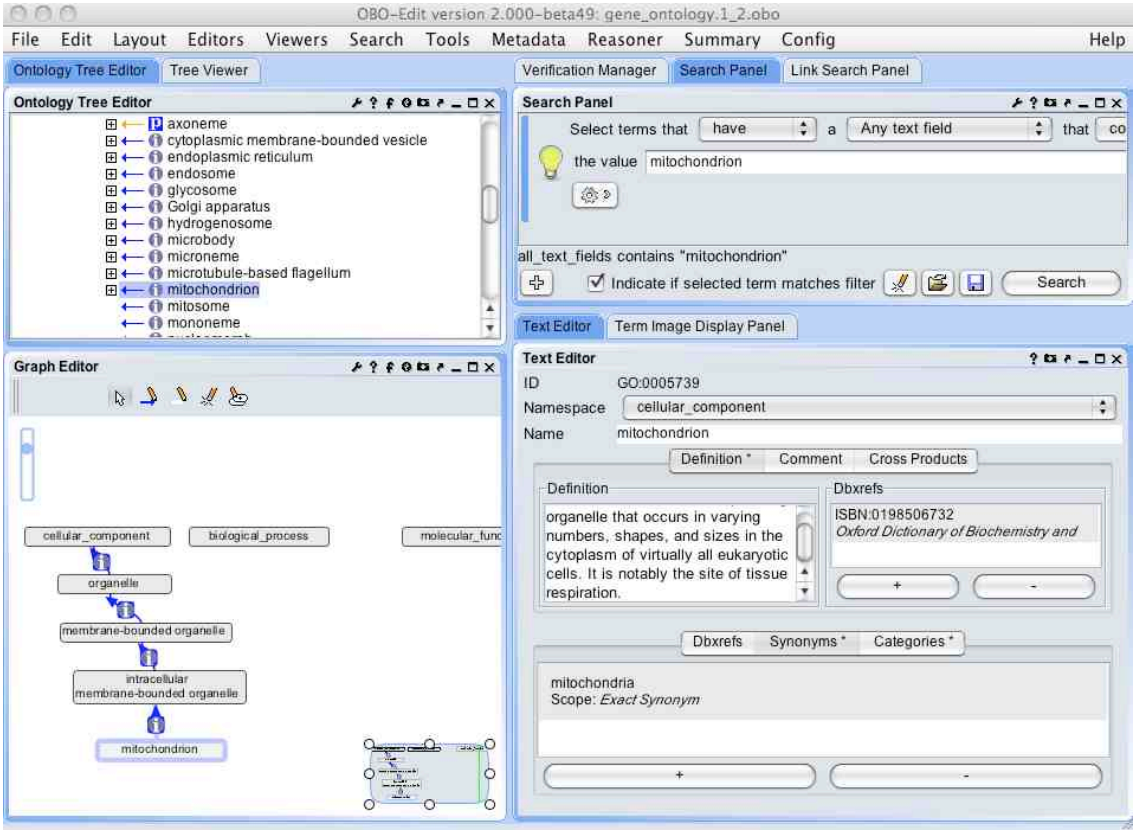

Figura 13: Pannelli principali di OBO-Edit

Nella fase di modifica e di sviluppo dell'ontologia possiamo avvalerci di diversi editor, messi a disposizione nell'applicativo, i quali presentano molteplici soluzioni grafiche e funzionali:

- *Ontology Tree Editor*: interfaccia classica, in cui le entità sono elencate in modo strutturato e gerarchico, costituisce il pannello in assoluto più importante del programma poiché è fondamentale anche per la navigazione e la gestione dei contenuti. Ciascun

termine possiede di fianco al nome un simbolo grafico che ne descrive la tipologia. Le operazioni consentite sono il trascinamento e quelle indicate in una finestra a comparsa, che riporta i soli comandi applicabili all'entità selezionata tra quelli presenti nella barra *Edit*.

- *Graph Editor*: (Fig. 14) pannello che propone uno schema a diagrammi rappresentante la morfologia dell'ontologia in esame, molto utile anche nel contesto di esplorazione. La sua vera particolarità, però, consiste nella possibilità di modificare le entità e le relazioni esistenti tra queste direttamente sul grafico. Le operazioni risultano più intuitive ed efficaci, e naturalmente le modifiche sono realmente apportate anche nel file in cui è salvata l'ontologia. I comandi consentiti sono quelli di trascinamento dei termini da un'entità all'altra, di creazione di collegamenti e nodi, di cancellazione di un oggetto o di clonazione.
- *Text Editor*: è il pannello dedicato alle modifiche delle caratteristiche delle singole entità. Ogni termine è munito di un ID, un *namespace* e un nome in linguaggio naturale; inoltre, si aggiunge la possibilità di inserire una breve descrizione, un commento o l'inserimento di relazioni quali i termini sinonimi ed eventuali categorie di appartenenza.
- *Parent Editor*: pannello che mostra superentità "padre" dell'entità selezionata e che permette di lavorare sulle caratteristiche della suddetta superentità, partendo con i tipi di relazioni che la legano a ciò che abbiamo selezionato, cancellandone i collegamenti, e ultimando con il controllo dei termini e dei *namespace* associati.

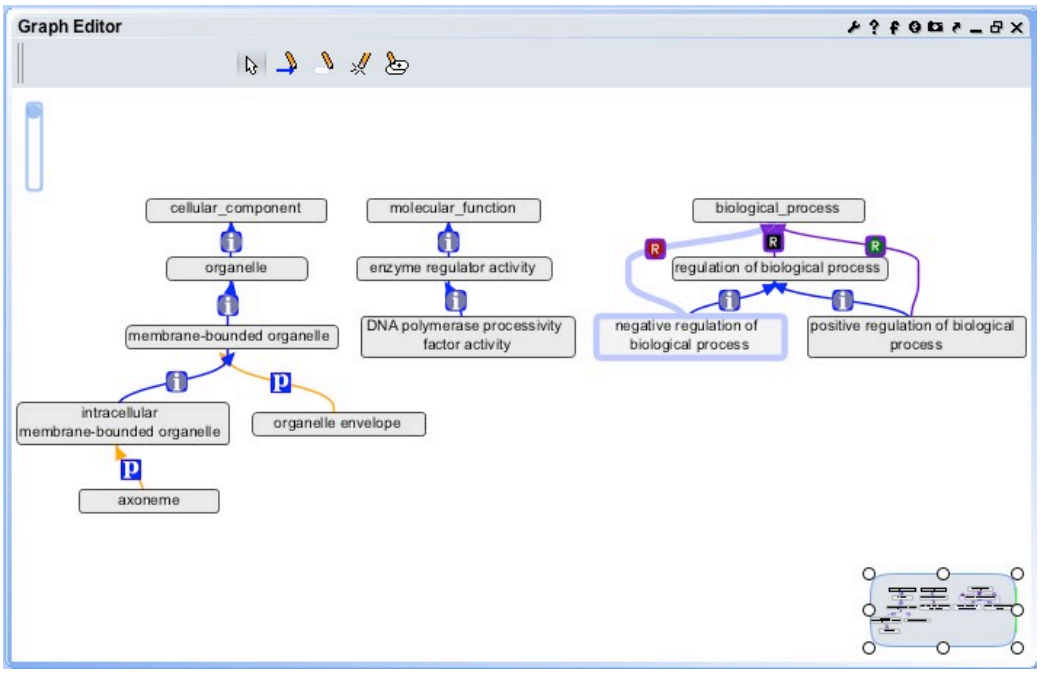

Figura 14: Graph Editor

Proseguendo con l'analisi della parte dedicata allo sviluppo e all'evoluzione dell'ontologia, possiamo consultare i comandi della barra *Edit*. Le operazioni previste sono quelle canoniche che abbiamo incontrato anche negli altri due tool, come ad esempio la modifica dei tipi di relazione, la creazione di entità gemelle o di entità clonate. In aggiunta OBO-Edit propone anche procedure più particolari, come quelle legate alla modifica dei namespace e degli ID delle entità, all'inserimento di radici nel percorso dell'ontologia, alla possibilità di imporre le proprietà booleane vero/falso in determinate proprietà.

L'ultimo aspetto che affrontiamo è quello dell'eliminazione, che in questo applicativo trova una soluzione alquanto originale, poiché esistono due distinti concetti di cancellare. La prima versione è rappresentata dal comando *Deleting*, con il quale viene rimossa una relazione dall'ontologia e l'entità corrispondente viene spostata nei termini obsoleti; se questa possiede delle sottoentità l'operazione non può essere portata a buon fine. L'altra alternativa è costituita dall'operazione di *Destroying*, che elimina definitivamente un'entità presente nell'ontologia; è quindi buona norma usarla con parsimonia e con entità aventi poche relazioni e collegamenti, poiché la creazione di buchi potrebbe rivelarsi pericolosa.

2.3.3 Esplorazione e visualizzazione

La navigazione con OBO-Edit può rivelarsi molto complessa, data la vastità ed eterogeneità di informazioni presenti nell'ontologia. Andremo allora ad analizzare tutte le proposte pensate dagli sviluppatori per l'esplorazione dei contenuti.
Innanzitutto esaminiamo la sezione della ricerca, che nell'applicativo si avvale di due pannelli:

- *Search Panel*: è la scheda dedicata alla ricerca dei termini all'interno dell'ontologia.
- *Link Search Panel*: è la scheda dedicata alle relazioni che intercorrono fra le entità dell'ontologia. Per ogni collegamento rilevato vengono riportati gli ID dell'entità figlio, dell'entità padre e della tipologia di relazione, i nomi delle entità figlio e padre.

Entrambi i pannelli si avvalgono di caratteristiche aggiuntive per affinare i criteri di ricerca, i filtri, suddivisi in due categorie distinte, ovvero i cosiddetti *term filters* e i *link filters*. I primi pongono limitazioni sui termini da cercare; ciò non toglie che i risultati della ricerca siano totalmente eterogenei, essi dovranno contenere o meno le tipologie dei nominativi indicati oppure una loro particolare caratteristica. I secondi, invece, costruiscono dei vincoli sui

collegamenti vigenti tra i termini, e i risultati dell'interrogazione non riporteranno le entità coinvolte, bensì i collegamenti tra queste.

L'esplorazione dei dati in OBO-Edit non può avvalersi solamente di criteri di selezione delle entità. Data la vastità di informazioni, l'utente deve avvalersi di ulteriori strumenti, che si concretizzano nell'esistenza di viste di diversa natura, con le quali è in grado di esplorare con più facilità ed intuizione l'ontologia.
In primo luogo, bisogna fare una distinzione tra i pannelli dedicati soltanto alla visualizzazione dell'ontologia, in particolare delle entità selezionate con le rispettive proprietà e relazioni di parentela con altre classi, ma che svolgono un ruolo passivo per ciò che concerne l'interazione con le informazioni presenti. Come controparte abbiamo i pannelli dove, oltre alla visualizzazione, è possibile esplorare e modificare le componenti dell'ontologia, che sono rappresentate da tutte le schede aventi *editor* come suffisso e che abbiamo già avuto modo di descrivere nel paragrafo precedente.
Quindi, non ci resta che analizzare le viste interpellabili dalla barra *Viewers*:

- *Graph Viewer*: è un pannello che scatta un'istantanea dal Graph Editor e illustra il grafico personalizzato dell'entità selezionata, riportandone la struttura gerarchica che la lega alla radice dell'ontologia.
- *Graphviz Viewer*: è un pannello di natura simile al Graph Viewer, che contiene il plugin GraphViz e che quindi descriveremo nei paragrafi successivi.
- *Term Image Display Panel*: anche questo pannello si avvale di un supporto esterno per il corretto funzionamento, lo illustreremo successivamente.
- *Tree Viewer*: (Fig. 15) è un pannello che riporta i percorsi effettuati durante la navigazione all'interno dell'ontologia tramite l'Ontology Tree Editor, avvalendosi in egual modo della visualizzazione delle entità tramite una struttura gerarchica ordinata.

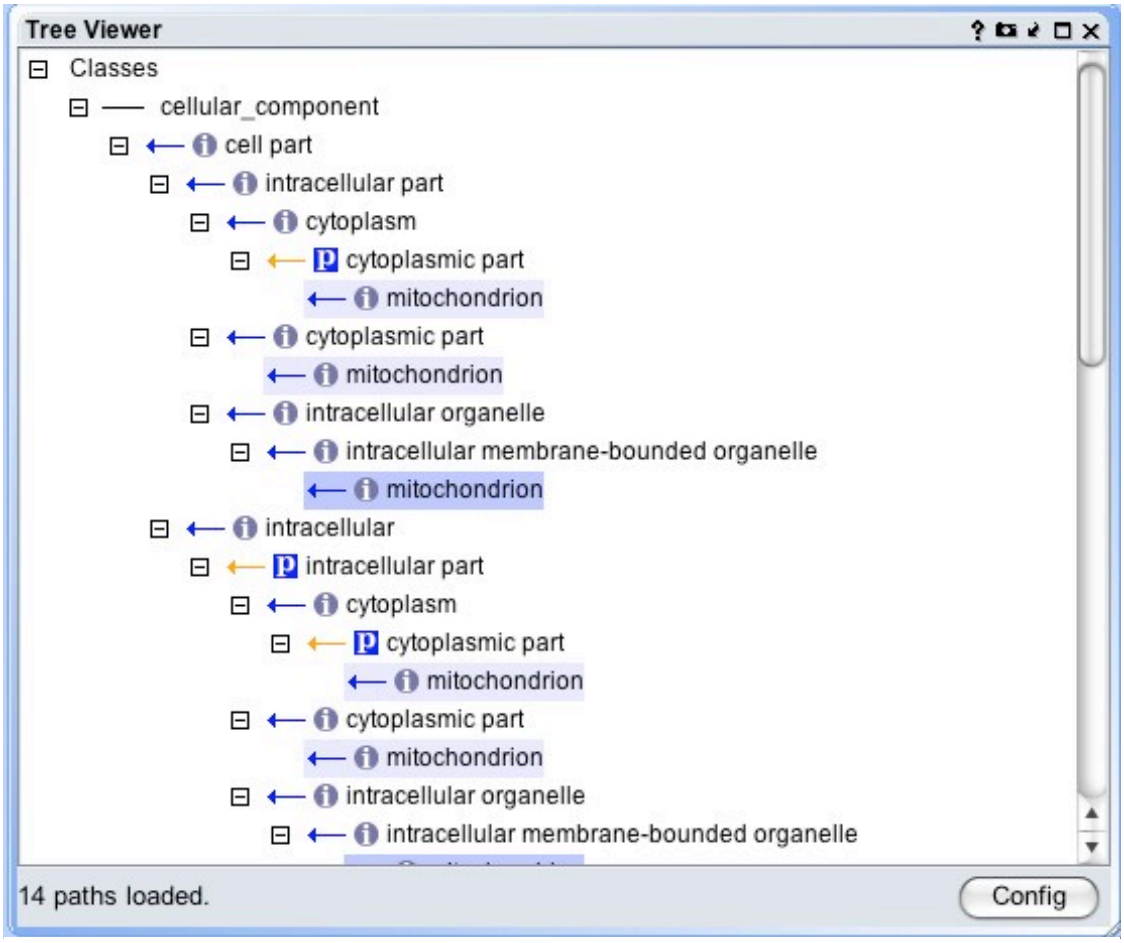

Figura 15: Pannello Tree Viewer

2.3.4 Gestione dei contenuti

Come abbiamo potuto constatare negli applicativi precedentemente descritti, la gestione dei contenuti di un'ontologia rappresenta uno dei punti peculiari per la buona riuscita di un progetto software. OBO-Edit si impegna altrettanto nel successo di questo aspetto, nella consapevolezza che l'ontologia sviluppata è particolarmente complessa ed articolata.

Innanzitutto, affrontiamo un'analisi dell'Ontology Tree Editor in modo differente rispetto a quanto fatto nei paragrafi precedenti, ovvero descrivendo il significato semantico delle tre radici fondamentali.

Il primo termine incontrato è *Classes*, che esplode la gerarchia delle entità presenti nell'ontologia; data la spiegazione più che esaustiva già effettuata, non è necessario indugiare ulteriormente. Proseguendo, troviamo il vocabolo *Relations*, in cui vengono elencate tutte le

relazioni e le proprietà che compaiono almeno una volta nelle classi, organizzate anche esse in una struttura gerarchica ad esplosione.

Infine, soffermiamoci sull'ultimo termine *Obsolete*, che potremmo intuitivamente tradurre Obsolescenze, ed è la radice che raccoglie tutte le entità che sono ritenute superate, sprovviste della validità avente in passato. Qualsiasi termine può essere spostato in questo gruppo grazie al comando *Deleting* e per ciascuno vengono indicate le entità da cui è stato rimpiazzato, una sorta di breve cronologia per potersi ricondurre rapidamente alla posizione corretta dell'ontologia in cui è avvenuta la modifica. Personalmente, trovo questa idea molto originale ed intuitiva, particolarmente adatta all'applicativo software sviluppato, soprattutto se si pensa agli obiettivi prefissati e alle modalità del suo funzionamento. (Fig. 16)

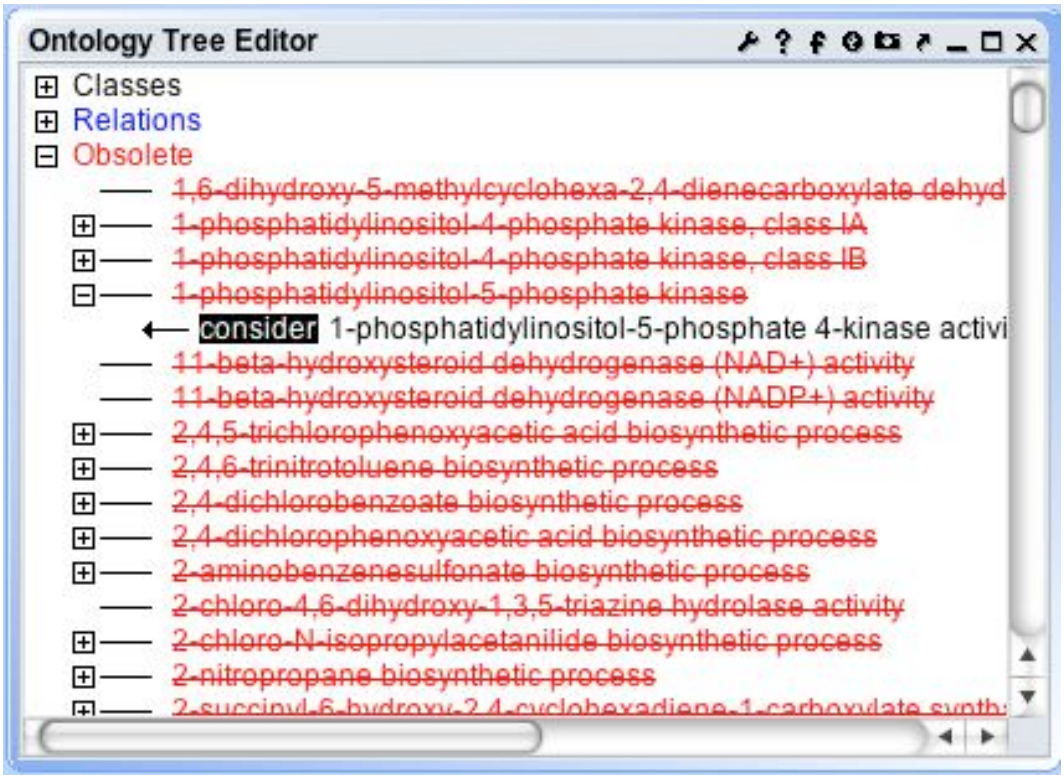

Figura 16: Gestione delle obsolescenze

Proseguendo con lo studio delle caratteristiche dedicate alla gestione dei contenuti, affrontiamo il concetto di cronologia. OBO-Edit, infatti, è in grado di costruire una storia legata allo sviluppo e alla modifica dell'ontologia, ordinando cronologicamente i movimenti svolti. Ogni qualvolta che un utente realizza un cambiamento, il programma crea un *history object* che registra tutti i dati riguardanti l'operazione effettuata e lo inserisce in una lista. Qualora l'utente volesse interpellare questo elenco può consultarlo nell'History Browser, un

pannello che propone l'intera cronologia dell'ontologia, suddivisa per sessioni di utilizzo e organizzata in una struttura gerarchica ad esplosione. Infine, è possibile salvare una data lista in un file locale, per poterla consultare anche quando il programma non è stato lanciato, oppure caricarla da un file salvato in una directory locale. (Fig. 17)

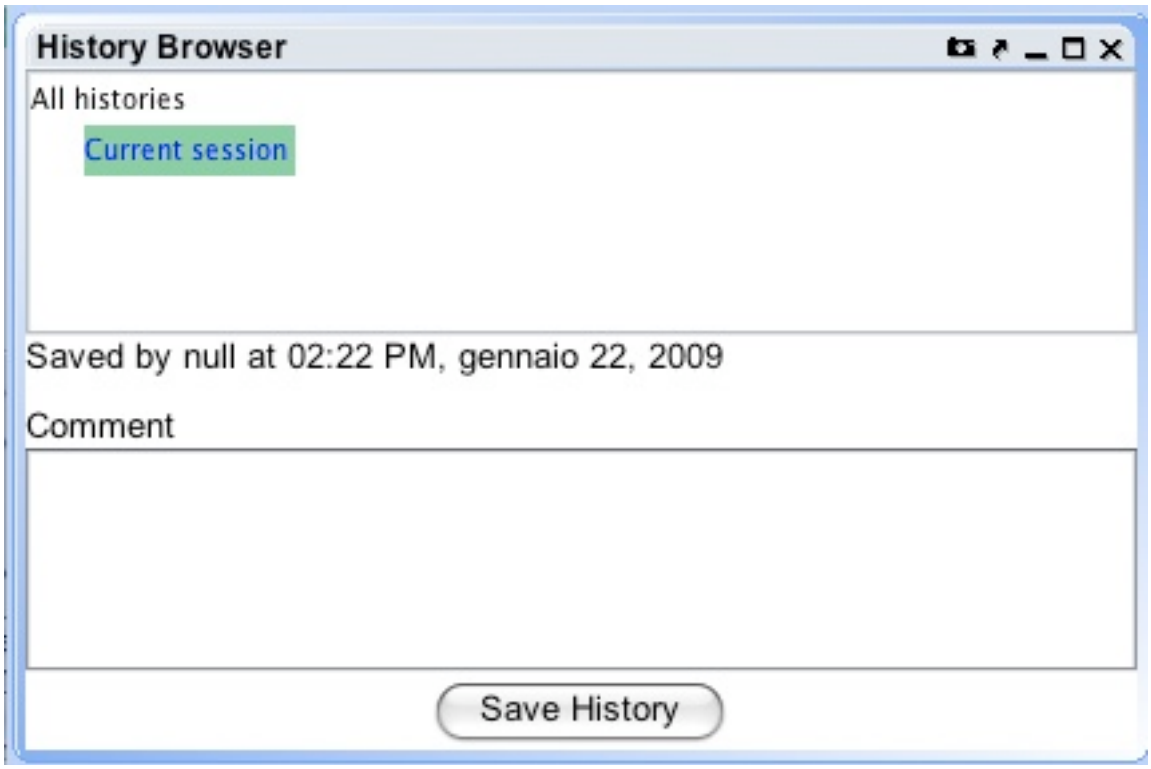

Figura 17: Pannello History Browser

Per quanto riguarda il reasoner, OBO-Edit differisce nelle scelte rispetto a Protégé e Swoop, poiché, sebbene ne possieda uno, esso non è un plugin bensì parte integrante del codice progettato internamente dagli sviluppatori. Oltre all'analisi strutturale dell'ontologia, allo studio delle relazioni interagenti tra le entità e delle proprietà presenti, il reasoner svolge una rilevante funzione di controllo, infatti è munito anche del cosiddetto *Semantic Parser Manager*, un algoritmo che ha la funzione di realizzare analisi puramente sintattiche dei termini esistenti. Una volta attivato, la scelta verte tra due possibilità disponibili:

- *Forward Chaining Reasoner*: è stato da sempre il reasoner utilizzato, veloce ma incompleto per interrogazioni particolarmente complesse.
- *Link Pile Reasoner*: è la nuova proposta offerta dagli sviluppatori, più lento ma decisamente più esauriente.

La sezione di controllo in OBO-Edit non si esaurisce con i reasoner ma compare in due ulteriori situazioni, poiché commettere degli errori all'interno dell'ontologia è molto frequente, nonché direttamente proporzionale alla vastità di informazioni in essa contenute.
In primo luogo, le ispezioni vengono effettuate attraverso *l'Ontology Verification*, che all'utente riporta istantaneamente errori fatali o *warning* in grado di compromettere la correttezza dell'ontologia. In particolare, le verifiche sono eseguite automaticamente in cinque fasi ben precise nella vita dell'ontologia, ossia nel momento di caricamento della medesima, durante la modifica di campi testuali e con l'utilizzo del Text Editor, che in aggiunta registra un *feedback* sulle operazioni svolte. Inoltre, avvengono controlli nella fase di salvataggio dell'ontologia, quando viene interpellato il reasoner, ed infine l'utente può richiedere un riscontro dello stato dell'ontologia con lo specifico comando manuale.
Le analisi di controllo sono molteplici ed composite, spaziano dal controllo dei nomi dei termini e dei rispettivi ID, alla presenza di sinonimi e di eventuali ridondanze. Oltre a ciò, si realizzano verifiche sui commenti e sulle definizioni, e sulle relazioni vigenti tra le entità, in particolare ci si preoccupa dell'esistenza di eventuali cicli chiusi, di collegamenti che non referenziano a nessun oggetto, infine delle proprietà di disgiunzione.
Quando l'utente vuole interpellare manualmente il sistema di controllo del programma, deve affidarsi al *Verification Manager*, pannello adibito a questo tipo di operazione. La finestra elenca tutte le tipologie di ispezioni consentite e tutti i cinque eventi appena descritti, l'utente allora dovrà semplicemente associare le revisioni da compiere nell'istante di attuazione di un determinato evento, ricordando che selezionarle tutte corrisponde al controllo automatico previsto da OBO-Edit .

Infine, citiamo il concetto di *Cross Product*, che è presente anche negli altri due applicativi software, ma che in questo contesto si manifesta con una nozione leggermente diversa e quindi meritevole di nota onde evitare ambiguità ed equivoci. In Protégé e Swoop, con questo termine definiremmo una relazione che intercorre tra entità di due ontologie differenti, mentre in OBO-Edit rappresenta quel particolare legame logico chiamato *intersezione*. La scelta di creare questa relazione assicura un funzionamento efficiente del reasoner, poiché garantisce l'identificazione di tutte corrispondenze gerarchiche tra le classi, nello specifico lo aiuta nell'individuazione di tutti i ruoli che una classe può assumere all'interno dell'ontologia.

2.3.5 Plugin e supporti esterni

In OBO-Edit qualsiasi iniziativa esterna di sviluppo di plugin è ben accolta, nella consapevolezza che non fanno parte dello standard del pacchetto software ma che aggiungono valore e funzionalità alle capacità del programma.
Riportiamo a seguire i supporti esterni principali.

GraphViz. È il plugin di cui si avvale anche Protégé, che realizza dei grafi basandosi sulla morfologia dell'ontologia. In realtà, come abbiamo visto, lo standard di OBO-Edit possiede delle visualizzazioni grafiche, ma ciò non toglie che l'utente possa preferire un'alternativa differente a quelle esistenti.

Shared Libraries. Il programma permette di unire le librerie maggiormente usate per lo sviluppo di ontologie in un'unica directory e di renderle condivise, affinché, da una parte, possano essere sfruttate ogni volta ne venga il bisogno e, dall'altra, non siano fonte di ridondanze o di allocazione di memoria inutile.

Term Image Display Panel. Più che un plugin, è da considerarsi un algoritmo che si avvale di un supporto esterno. Infatti, attraverso questo pannello, è possibile associare un'immagine in corrispondenza di un termine, che viene caricata quando l'entità viene invocata nell'editor. Il meccanismo collega un'immagine localizzata in una directory locale, consultabile nella finestra delle preferenze del programma, con un certo termine, salvandola con l'ID dell'entità a cui si vuole associare. È facilmente intuibile allora che questo supporto presenta una limitazione consistente, e che sarebbe interessante poter sviluppare, ovvero che non può caricare più di un'immagine alla volta in relazione ad una particolare entità. (Fig. 18)

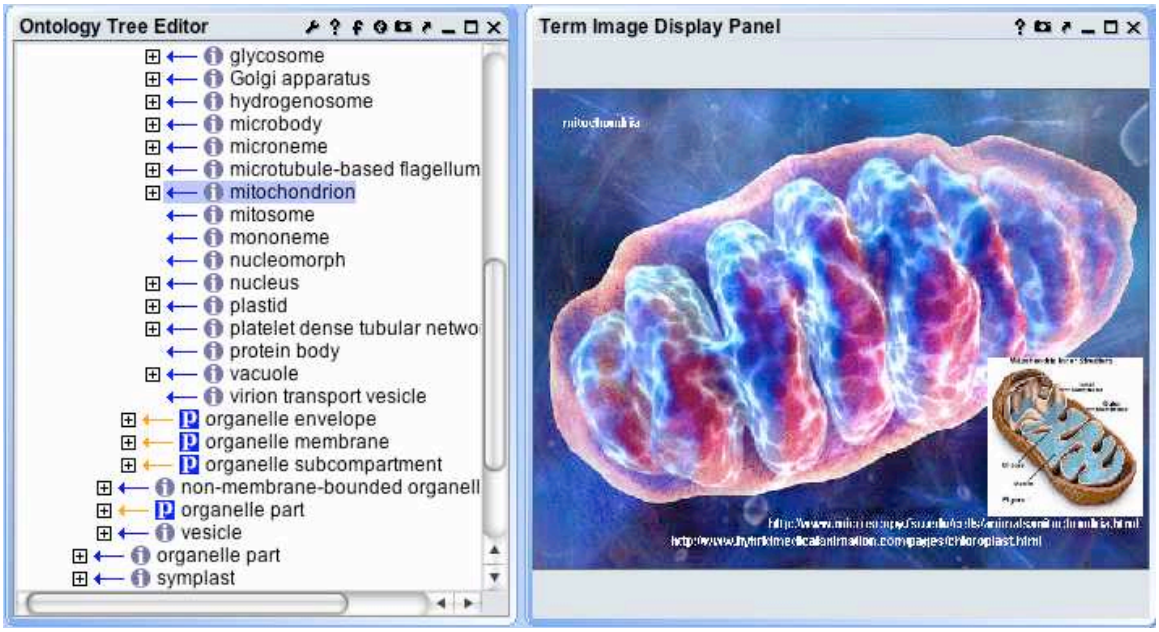

Figura 18: Term Image Display Panel

AmiGO. Anche in questo caso non si parla di plugin, quanto di supporto esterno, che sicuramente è caratterizzato dall'originalità e nell'unicità della sua essenza. Esso è un motore di ricerca che può essere invocato tramite Browser Web e riporta, come risultati della ricerca, le entità presenti nell'ontologia generata da OBO-Edit. Rappresenta, quindi, un'alternativa per coloro i quali necessitano di consultare la terminologia sviluppata in questo progetto e, più in generale, nell'intero progetto di *Gene Ontology*, che però sono sprovvisti di una versione locale degli applicativi e dunque non sono in grado di effettuare alcun tipo di navigazione all'interno dell'ontologia. Possiamo notare che l'interazione con le informazioni, in questo caso, è parziale e soprattutto passiva, ma risulta vincente per quanto concerne gli obiettivi di diffusione e riutilizzo prefissati dagli sviluppatori.

3. CONFRONTO DEGLI EDITOR

3.1 Usabilità

Con il termine usabilità intendiamo la facilità con cui l'utente è in grado di approcciarsi al programma ed imparare ad usarlo, fattore dipende strettamente dalla complessità della morfologia dell'applicativo e dalle scelte di impostazione dell'interfaccia grafica. Partendo da questa nozione, andiamo a confrontare i nostri editor sotto questo aspetto estrapolando pregi e difetti di ognuno.

In primo luogo esaminiamo come si pone Protégé di fronte a questa caratteristica. Abbiamo avuto modo di notare che, senza ombre di dubbio, tra quelli trattati rappresenta il pacchetto software più esaustivo e completo per ciò che riguarda i contenuti e i criteri di interazione con essi. Pertanto, l'applicativo si rivela indicato ad un'utenza esperta nel settore, in grado di maneggiare con sicurezza i concetti legati allo sviluppo di ontologie. Ciononostante, il programma vanta della miglior interfaccia grafica tra quelli studiati in questa tesi, allora possiamo affermare che le scelte degli sviluppatori in materia di layout soccombono alla complessità dell'applicativo, strutturalmente più complesso.
Passando a Swoop, possiamo dire che il fattore usabilità fa parte delle prerogative fondamentali prefissate, per questo motivo esso può considerarsi il programma più intuitivo da utilizzare. Questo obiettivo raggiunto, però, è a scapito di alcune lacune nei suoi contenuti e nelle sue capacità, limitando il *target* di utenti ad un gruppo più ristretto, ipoteticamente a coloro i quali non hanno bisogno di svolgere operazioni particolarmente complesse sulla loro ontologia. Infatti, non a caso, gli sviluppatori hanno progettato un layout, ed un intero applicativo, che ricorda prepotentemente la filosofia di un browser Web.
Infine, arriviamo ad OBO-Edit che, per sua natura, deve essere un dispositivo facile da usare. La sua usabilità è data dalla semplicità delle nozioni tecniche trattate più che dall'interfaccia grafica, anche se in realtà è tutt'altro che complessa da capire. Se però pensiamo allo scopo per cui è stato sviluppato questo applicativo, compresa l'utenza a cui è indirizzato, e ai requisiti funzionali che deve garantire, possiamo concludere che ciò che è stato implementato risulta più che sufficiente per coprire le esigenze dei suoi utilizzatori. Dunque, sebbene formalmente si scopre essere il pacchetto software meno progredito per ciò che concerne la caratteristica dell'usabilità, appare essere più che esauriente per conseguire i suoi scopi.

3.2 Scalabilità

Il fattore scalabilità esprime la capacità di un programma di crescere o decrescere in funzione delle disponibilità e delle necessità richieste in un determinato contesto. Nel nostro caso, non abbiamo in esame applicativi che si possono definire scalabili nel senso rigoroso del termine, poiché in realtà si trattano di progetti software molto compatti e non scindibili in parti che possono essere assemblate o sciolte a seconda delle esigenze. Pertanto, in questo frangente con la nozione di scalabilità intenderemo una caratteristica alleggerita e meno strutturale, ovvero la capacità degli applicativi di accorpare a sé dei pacchetti software, rappresentati nel concreto dai *plugin*.

In questo senso, possiamo affermare che tutti e tre i programmi studiati in questa tesi sono proiettati nell'ottica di poter approfondire le loro capacità di prestazione con l'aggiunta di supporti esterni; infatti, per accorgersi di ciò, basta consultare l'elenco di plugin a loro disposizione. L'applicativo dotato di maggior possibilità di arricchimento è sicuramente Protégé, merito della sua alta diffusione e del fatto che è il più importante prodotto del settore. Basti pensare che anche all'interno della nostra università, sono proposte delle attività progettuali mirate alla creazione di plugin per questo pacchetto software. È facile intuire, allora, il suo ruolo da protagonista, assunto nell'ambito dell'istruzione e della ricerca, e del conseguente contributo proveniente al di fuori dei collaboratori interni.
Allo stesso livello, possiamo affermare che Swoop si impegna molto nello sviluppo di plugin. Inoltre, esso affida molta importanza alle collaborazioni provenienti da fonti esterne e ne incentiva lo sviluppo.
L'unico applicativo che affronta questa problematica con meno enfasi è OBO-Edit, e questo evento si può giustificare ricordando che esso non è un editor generico, bensì creato per un'ontologia ben precisa, quindi l'esigenza di allargare le sue conoscenze risulta meno decisa. Le operazioni necessarie per lavorare con il progetto Gene Ontology sono già pienamente implementate.

3.3 Stabilità

Nel mondo informatico, stabilità è sinonimo di affidabilità, di buona riuscita delle operazioni richieste dall'utente; naturalmente, quando degli applicativi software vengono pubblicati o

rilasciati sul mercato, non si può pensare che siano mal funzionanti. Ciononostante, possiamo esaminare comunque questo requisito, riferendoci a come i programmi rispondono alle operazioni più complesse invocate dall'utente.

Protégé, anche in questo frangente, si dimostra l'applicativo più completo e meglio sviluppato. L'utente può svolgere in totale sicurezza ed affidabilità implementazioni complesse, che prevedono un'esigua mole di dati coinvolti e algoritmi dispendiosi. Inoltre, basta notare che i comandi consentiti sono proposti una gamma decisamente più vasta; ciò implica che ci siano alternative più approfondite rispetto agli altri due programmi, e questa differenza sarà a favore dei comandi più complessi, poiché le operazioni di base, o comunque quelle ritenute essenziali, non potranno essere omesse.
Passando a Swoop, possiamo affermare che anch'esso è un applicativo estremamente stabile e sicuro, ma che differisce rispetto Protégé per complessità dei comandi presenti ed applicabili. In realtà, però, non possiamo recriminare questo aspetto come se fosse una lacuna, perché costituisce una scelta volontaria degli sviluppatori quella di creare un pacchetto software più leggero e munito delle operazioni essenziali, senza voler approfondire più di tanto alcuni comandi ritenuti interessanti solo ad una parte ristretta tra gli utilizzatori.
Analogamente, lo stesso discorso si può estendere ad OBO-Edit, che è provvisto delle sole operazioni strettamente necessarie. La scelta degli sviluppatori, in questo caso, può essere interpretata come auspicio di non voler appesantire l'applicativo con comandi macchinosi e lenti, quando in realtà non verrebbero sfruttati dagli utenti; inoltre, ricordiamo le estensioni dell'ontologia implementata, un'ulteriore carico non sarebbe sicuramente d'aiuto per un funzionamento fluido del programma. Bisogna precisare però che, nonostante la suddetta politica adottata, alcune situazioni risultano essere instabili e non particolarmente sicure. Infatti, alcune operazioni sembrano discretamente lente e a rischio di blocco del programma, ma soprattutto compaiono messaggi di errore o *warning* con una frequenza che, a mio avviso, è eccessiva, specialmente se si pensa all'importanza dei contenuti dell'ontologia e ai danni in cui possono intercorrere. Personalmente, cercherei di curare maggiormente l'aspetto di *debugging*, con l'obiettivo di evitare la creazione di eventi che possano in qualche modo mettere a repentaglio la sicurezza dei dati, partendo da una modifica involontaria arrivando ai casi estremi di perdita delle informazioni. Considerando la natura dell'ontologia sviluppata in OBO-Edit, sarebbe molto grave degenerare in situazioni apocalittiche in cui non si ha più la certezza della veridicità dei dati, poiché non solo l'applicativo sarebbe considerato instabile, ma le informazioni stesse diventerebbero del tutto inaffidabili, e l'intero progetto crollerebbe.

3.4 Integrazione

In questo contesto, con il termine integrazione, si intende la fusione di due o più ontologie in una sola, ponendo una particolare importanza al riutilizzo delle informazioni che possono essere condivise; in realtà, questa condizione non è propria degli editor quanto semmai delle ontologie stesse, poiché gli applicativi devono garantire gli strumenti appropriati per poter intervenire in questa direzione, ma è l'ontologia nei suoi contenuti che deve essere predisposta ad un'integrazione con altre strutture. Possiamo affermare, quindi, che per integrare correttamente delle ontologie bisogna lavorare ad un livello più alto, più concettuale, anche se indubbiamente dobbiamo avvalerci di applicativi in grado di tradurre le nostre idee.

Passando all'analisi dei nostri programmi possiamo dichiarare che Protégé, nonostante la sua struttura molto compatta e totalmente autonoma, è sicuramente quello che ambisce maggiormente al conseguimento degli obiettivi di integrazione tra le ontologie, d'altronde non bisogna dimenticare che essa rappresenta uno degli scopi primari prefissati anche dal Web semantico. Infatti, l'applicativo è provvisto di tutti gli strumenti necessari per poter plasmare più ontologie tra loro, garantendo quindi la possibilità di realizzare archivi di informazioni condivise e riutilizzabili.
Proseguendo con Swoop notiamo subito che la predisposizione all'integrazione rispetto a Protégé è meno evidente, o meglio persegue scopi diversi. In realtà, sono previsti comandi per la fusione di ontologie, ma gli sviluppatori hanno preferito curare maggiormente delle operazioni che mirano al riutilizzo dei contenuti all'interno della medesima ontologia, pensiamo ad esempio ciò che rappresenta lo strumento del *Resource Holder*. Ciò non toglie, che questi strumenti possano rivelarsi utili anche nell'integrazione "classica" di ontologie; la differenza in valore aggiunto sarà fatta dall'astuzia dell'utente e dalla sua conoscenza dell'applicativo, poiché davanti ad una *confidence* rispettabile nei confronti del programma, egli sarà in grado di sfruttare al meglio tutti i comandi a proprio favore a seconda della situazione.
Infine, concludiamo con OBO-Edit, dove il concetto di integrazione differisce enormemente rispetto a ciò che abbiamo stabilito. Gli oggetti di integrazione, in questo caso, non sono più le ontologie bensì i contenuti di queste, considerando che lo sviluppo delle conoscenze si compie in un'ontologia unica. Potremmo affermare che il tentativo di risolvere il problema avviene alla radice, generando direttamente un'unica struttura in grado di conservare tutte le informazioni ed evitando la creazione di più strutture ridondanti. Ciononostante, il

programma prevede una struttura minima per queste operazioni volta all'intervento in casi straordinari, onde evitare che eventuali problemi sfocino in situazioni precarie o estremamente gravose.

3.5 Documentazione

Un buon applicativo software non può essere considerato tale se non vanta di una buona documentazione che ne esplori i contenuti e le capacità, poiché deve poter mettere a proprio agio anche l'utente più inesperto. Inoltre, un buon testo sorgente è indice di uno sviluppo del software ordinato e rigoroso, diversamente, davanti ad un applicativo implementato in modo scomposto e poco coeso, non si riuscirebbe nemmeno a costruire una guida in grado di illustrarne le caratteristiche.

Consultare la documentazione di Protégé è molto semplice, basta collegarsi al sito http://protegewiki.stanford.edu/index.php/Main_Page da qualsiasi browser Web. L'indirizzo indicato costituisce la prima pagina di un intero mondo parallelo all'applicativo software, poiché si possono trovare guide per utenti e sviluppatori, suddivise ulteriormente per *release* installata, nonché una parte introduttiva che spiega tutto ciò che concerne la parte teorica e concettuale quando si parla di ontologie e linguaggi per il Web semantico. Il lavoro svolto in questo sito è eccellente, esaustivo e sempre aggiornato, si rivela un ottimo punto di riferimento per tutti gli utilizzatori di Protégé.
Passando a Swoop purtroppo non possiamo essere così entusiasti, infatti l'applicativo non ha nessuna documentazione ufficiale, l'unica reperibile è localizzata dall'indirizzo http://www.mindswap.org/papers/SwoopJWS_Revised.pdf, ma più che una guida per l'utente si tratta di una relazione approfondita. Personalmente, la considero una grave lacuna a cui si dovrebbe porre velocemente rimedio, poiché una documentazione è una "cartina al tornasole" che esprime la qualità del programma. Pertanto, nonostante la sua semplicità e la sua intuitività, Swoop non può permettersi di non possedere un manuale a cui l'utente può attingere.
Infine, per quanto riguarda OBO-Edit, è presente una guida per gli utenti all'interno del programma. Le informazioni fornite sono vaste e ben organizzate, grazie ad una struttura con collegamenti ad ipertesto. Gli unici difetti che si possono recriminare sono, prima di tutto, una chiarezza parziale nelle descrizioni introduttive, che spaziano dalla presentazione dei concetti

dell'ontologia sviluppata propri del programma ai comandi leggermente più complessi. In secondo ed ultimo luogo, alcuni campi sono ancora da completare. Nel complesso, comunque, la documentazione è da considerarsi più che esauriente per l'apprendimento del funzionamento del pacchetto software.

3.6 Originalità

In questo paragrafo, vogliamo riproporre in sintesi gli elementi più singolari ed unici di ciascun editor, quindi in realtà sono concetti e discussioni già trattati nel capitolo precedente, ma che sono parte integrante delle caratteristiche software e che penso valga la pena raccogliere per realizzare un confronto davvero completo. La mia scelta si limiterà all'individuazione di due speciali caratteristiche per ogni programma, di cui una delle due sarà una particolarità propria del layout, onde evitare di cadere nel prolisso e rendere l'analisi dispersiva.

Partendo da Protégé, citiamo come elemento di spicco dal punto di vista grafico il pannello delle *Entities*, in cui la navigazione e l'esplorazione dei contenuti risulta essere particolarmente intuitiva, e allo stesso tempo abbiamo la possibilità di consultare tutte le tipologie di entità presenti nella nostra ontologia. Inoltre, è sicuramente nota di merito tutta la parte che concerne lo sviluppo di integrazione, partendo dai *reasoner* e dalle operazioni per effettuare fusioni tra più ontologie, arrivando all'opportunità di importare una vasta gamma di plugin e supporti esterni, che realmente possono essere implementati senza compromettere il buon funzionamento dell'applicativo.
Analizzando Swoop, invece, possiamo riportare come fattore caratteristico la formulazione del layout con un'architettura *Web-oriented*, per creare un contesto che ricorda i browser Web. Questa scelta ha lo scopo preciso di aiutare l'utente ad approcciarsi con l'applicativo, grazie ad un ambiente che si presuppone essere familiare a qualsiasi persona usi anche solo minimamente un personal computer, dato il successo dilagante del World Wide Web negli ultimi anni. Il secondo elemento che andremo a citare è il *Resource Holder*, una finestra in cui si possono salvare particolare entità di cui si vuole tenere traccia. Questa possibilità risulta essere molto originale ma allo stesso tempo intuitiva, dato il risvolto pratico e le agevolazioni che può trarne l'utente.

Infine, concludiamo con OBO-Edit e, parlando di caratteristiche dell'interfaccia che spiccano sulle altre, non possiamo che citare il *Graph Editor*. Il pannello rappresenta lo strumento che meglio riassume cosa vuol dire lavorare sull'ontologia del gruppo *Gene Ontology*, e l'idea di creare un grafico con il quale, oltre alla visualizzazione canonica è consentito un intervento diretto sulle informazioni, è senza dubbio geniale. Esso, di fatto, è il miglior modo per poter lavorare su ontologie particolarmente vaste, in cui è facile smarrirsi. Potremmo dire che anche gli altri due editor potrebbero pensare allo sviluppo di una situazione simile. L'altra caratteristica scelta che andremo ad esporre è la radice *Obsolete*, unica nel suo genere, in cui vengono raccolte le entità ritenute desuete e superate. Inoltre, citando questo aspetto così singolare, dobbiamo ricordare che argomento strettamente collegato è la distinzione tra i comandi *Deleting* e *Destroying*, dove il primo trasferisce i dati nella radice delle obsolescenze, mentre il secondo cancella definitivamente un'entità dall'ontologia.

Proponiamo ora una tabella che raccoglie tutti gli aspetti analizzati in questo capitolo, in cui ogni caratteristica viene valutata con un punteggio che varia da un minimo di 1 ad un massimo di 3. (Fig.19)

	Protégé	**Swoop**	**OBO-Edit**
Usabilità	○ ○	○ ○ ○	○ ○
Scalabilità	○ ○ ○	○ ○	○ ○
Stabilità	○ ○ ○	○ ○ ○	○
Integrazione	○ ○ ○	○ ○	○ ○
Documentazione	○ ○ ○	○	○ ○
Originalità	Pannello *Entities* Integrazione	Layout *Web-oriented Resouce Holder*	*Graph Editor Obsolete*

Figura 19: Tabella riassuntiva

4. CONCLUSIONI

Nei capitoli precedenti abbiamo cercato di analizzare degli editor molto eterogenei tra loro, facendo risaltare questa diversità e le loro caratteristiche principali. Abbiamo toccato le tematiche fondamentali che qualsiasi applicativo deve curare, e infine abbiamo realizzato un confronto diretto ed incrociato sulla base di una scelta di requisiti primari, che ciascuno sviluppatore deve prendere in considerazione quando decide di implementare un pacchetto software.

L'obiettivo di questa tesi, però, va ben oltre ad una discussione approfondita di editor, bensì vuole sottolineare il ruolo di prima linea che le ontologie stanno guadagnando negli ultimi anni. L'evoluzione del Web è inarrestabile, da struttura rigida e statica si sta trasformando in un contenitore di informazioni sempre più dinamico e flessibile, simulando un'intelligenza desiderosa di assomigliare a quella umana. Pertanto, il Web, e poi successivamente il Web semantico, non possono pensare di progredire senza avere strumenti appropriati alle spalle; essi rappresentano le fondamenta su cui costruire un intero mondo di conoscenze che non possiede limiti di espansione. Le ontologie e gli editor, allora, costituiscono le basi indispensabili per poter progettare una nuova era della rappresentazione della conoscenza, sono il punto imprescindibile su cui generare una novità, un nuovo impero dell'informazione. Risulta davvero difficile pronosticare cosa ci riserverà il futuro in questo ambito, dato che nessuno, o perlomeno pochi individui, avrebbe intuito in anticipo come il Web si sarebbe evoluto, e soprattutto la rapidità di questa sua ascesa. Ciò non toglie che non dobbiamo farci cogliere impreparati da nuove proposte e cambiamenti, per questo ritengo importante fornire alla base, alla radice, dei mezzi che consentano alla nostra creatività di potersi esprimere al meglio.

Con la stesura di questa tesi ho voluto dimostrare quanto ci sia da imparare e da esplorare nel mondo delle ontologie, argomento molto diffuso in ambiti universitari e tecnico-informatici, ma che ben presto prenderà piede nella vita quotidiana di un'utenza più allargata. Sono stati affrontati concetti che apparentemente sembrano lontani e distanti alla maggior parte degli utenti, ma che si diffonderanno nella vita quotidiana di ognuno di noi, come solo le scoperte informatiche in questi anni hanno saputo fare.

5. FONTI E RIFERIMENTI

- [1] www.wikipedia.org

- Materiale didattico del corso di studi “Rappresentazione della conoscenza” tenuto dalla prof. Bergamaschi

La documentazione degli editor è consultabile ai seguenti indirizzi:

- Protégé: http://protegewiki.stanford.edu/index.php/Main_Page

- Swoop: http://www.mindswap.org/papers/SwoopJWS_Revised.pdf

- OBO-Edit: è presente una guida per l’utente all’interno del programma