UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

tesi di laurea

Tools e ambienti per lo sviluppo di ontologie per il web semantico

Anno Accademico [2003/2004]

relatore
Ch.mo prof. Antonio Picariello

candidato
Angelo Zarrillo
matr. 534/36

[Dedica]

Indice

Introduzione

Il Web è nato come un'entità statica in grado di fornire informazioni; inizialmente è stato concepito come un immenso archivio in *sola lettura* al quale tutti potessero accedere e potessero prelevare informazioni: è questo il modello del Web a cui noi tutti siamo ormai abituati.

Negli ultimi decenni, però, il Web sta cambiando, diventando sempre di più un'entità dinamica, in grado cioè di poter interagire, fornire servizi e rispondere alle domande degli utenti. Un esempio evidente è rappresentato dai motori di ricerca quali Google, Yahoo, Hotbot, che sulla base di alcune parole significative (*keywords*) digitate dall'utente producono una serie di risultati concernenti l'argomento della ricerca. E' chiaro però che questi rappresentano solo la punta dell'iceberg di tutta una serie di servizi che il Web oggi mette a disposizione.

Il target di queste applicazioni è la realizzazione di un web in grado di inferire informazioni da un qualunque tipo di dato esso si trovi ad elaborare. Si vuole cioè elevare la condizione di un elaboratore, dal semplice contenitore di informazioni, ad uno strumento capace di *comprendere* tali flussi di dati e di rielaborarli al fine di interagire con gli utenti. Tale cambiamento è battezzato come Concettualizzazione del Web che porta al concetto del cosiddetto *Web Semantico*.

La ricerca e il recupero concettuale/semantico, il supporto alle decisioni, la comprensione del linguaggio naturale e del parlato, fino ai database intelligenti e al commercio elettronico, sono solo alcune delle prospettive del prossimo futuro.

I fini descritti sono i presupposti per la creazione di sistemi di informazione di *comprensione semantica* con l'obiettivo di supportare diverse attività siano esse personali, governative, di impresa.

Esempi di tali sistemi ricoprono numerose aree di applicazione: ricerca del web semantico; la creazione di linee guida mediche per la cura della salute dei pazienti; la mappatura del genoma delle piante e degli animali; la ricerca di specifiche risorse di pubblico dominio; la progettazione ingegneristica collaborativa; lo scambio elettronico automatizzato di informazioni tra partners commerciali.

La possibilità di ottenere un simile risultato è data dalla necessità di scambiarsi informazioni chiare e comprensibili, che portino con sé una semantica, un significato intrinseco non ambiguo, che non possa essere frainteso. L'interoperabilità ha quindi bisogno di un modello unico per la rappresentazione delle informazioni: questo è l'ambito in cui si inseriscono le *ontologie*.

Il termine *ontologia*, preso in prestito dal linguaggio filosofico[1], indica un documento condiviso che contiene la descrizione formale dei concetti di un dato dominio; identifica le classi più importanti, le organizza in una gerarchia, specifica le loro proprietà (che caratterizzano anche gli oggetti appartenenti alla classe) e descrive anche le relazioni più significative, che legano queste classi.

Queste sono le tematiche che saranno trattate nel seguito, per fornire una adeguata conoscenza delle motivazioni alla base della creazione di ontologie e allo sviluppo dei numerosi tool che la supportano.

[1] In termini puramente filosofici il termine ontologia indica la teoria che studia le modalità fondamentali dell'essere in quanto tale al di là delle sue determinazioni particolari. Si può definire dunque come la *scienza* che studia quali tipi di cose esistono.

Prefazione

L'elaborato di tesi procede come segue. La descrizione parte dal concetto generale di un'ontologia (cap. 1), si descrive la necessità di utilizzare ontologie per la rappresentazione delle informazioni, segue un confronto con altri tipi di rappresentazione e si illustra il motivo per cui ha prevalso l'uso delle ontologie.

Si procede poi, nel capitolo 2 con la descrizione del concetto di Semantic Web e di come le ontologie si inseriscono nella sua architettura. Inoltre si fa un breve accenno ai linguaggi per la creazione di ontologie (OWL, Ontology Web Languages) e ai metadati per descrivere il contenuto delle pagine web. Nel capitolo 3 è presente una sorta di guida alla creazione di ontologie: da cosa bisogna partire, quali domande bisogna porsi, quale è il risultato a cui si deve arrivare, e sono presentate regole pratiche per la modellazione del contenuto di un'ontologia. L'analisi è focalizzata sull'intero processo, a partire dall'acquisizione della conoscenza del dominio del problema, fino alla realizzazione dell'ontologia nel suo complesso.

Il capitolo 4 passa in rassegna numerosi tool per le ontologie suddivisi in funzione delle proprie funzionalità: tool di sviluppo, tool di integrazione e tool di valutazione. Di essi si descrivono le caratteristiche e si individuano i principali pregi. Si passa poi ad una riflessione sull'evoluzione futura delle ontologie e dei tool (cap. 5), per poi chiudere con una serie di considerazioni generali.

Capitolo 1

Le ontologie web

I moderni metodi di memorizzazione dell'informazione richiedono un significativo sforzo da parte degli essere umani nel realizzare il processo critico dell'interpretazione delle informazioni. I computer sono estremamente abili nel processare grandi quantità di dati, ma solo nel momento in cui si "dice" loro cosa fare con essi. Ad esempio, ad un computer possono essere trasferiti centinaia di migliaia di numeri a cinque cifre attraverso un file chiamato *codicipostali.txt*.

Un computer sarà del tutto incapace di analizzare queste informazioni come un essere umano, il quale tenterà di assegnare ad ogni codice il nome di una località. Anche un computer sarebbe in grado di farlo, ma solo quando gli venga detto dove cercare e come procedere.

1.1 Cos'è un' ontologia

Un'ontologia definisce i termini usati per descrivere e rappresentare un'area di conoscenza. Le ontologie sono utilizzate dalla gente comune, dai database, dalle applicazioni che hanno bisogno di scambiarsi informazioni di un certo dominio (un dominio è semplicemente una specifica area di conoscenza, come la medicina, la manifattura, i beni immobili, l'automobilistica, la gestione delle finanze ecc.). Le

ontologie includono definizioni dei concetti base del dominio e delle relazioni tra questi, in un *linguaggio* comprensibile e usabile da un computer.

Le ontologie codificano la conoscenza in un dominio e inoltre codificano la conoscenza che ricopre più domini. E' in questo senso che si può dire che esse rendono *riusabile* la conoscenza.

La parola ontologia è stata usata per descrivere "artefatti" con differenti livelli di struttura. Questi vanno dalle semplici tassonomie (come le gerarchie) agli schemi per metadati(dati per descrivere altri dati) e alle teorie logiche. Le ontologie possono essere definite come una *specificazione di una concettualizzazione*, [1] nel senso che esse stabiliscono una terminologia comune tra i membri di un dominio di interesse. Questi agenti possono essere umani o automatici.

Per rappresentare una concettualizzazione è necessario un linguaggio. Esistono numerosi linguaggi e sistemi di rappresentazione perlopiù proprietari. D'altra parte per le applicazioni web è importante poter usufruire di un unico linguaggio con una sintassi ben standardizzata. Poiché XML (eXstended Markup Language) sta assumendo il ruolo di linguaggio standard per lo scambio dei dati sul web, sarebbe desiderabile potersi "scambiare" ontologie usando una sintassi XML-like. Ciò porta alla definizione di un linguaggio basato su XML ad un livello più alto nella definizione delle pagine HTML (HyperText Markup Language). Sono stati proposti vari tipi di linguaggi XML-Based, la cui sintassi è molto simile all'XML, ma con lievi differenze nei nomi dei tag.

Il Web Semantico ha bisogno di ontologie con un significativo livello di strutture; queste devono possedere elementi per descrivere i seguenti tipi di concetti:

- Le classi dei principali domini di interesse;
- Le relazioni che tra esse possono esistere;
- Le proprietà (o attributi) che le classi possono avere.

Le ontologie sono solitamente espresse in un linguaggio "basato sulla logica", in modo da

essere dettagliato, accurato, consistente, e in maniera tale che "distinzioni espressive" possono essere effettuate tra le classi, tra le proprietà e le relazioni. Alcuni tool possono effettuare ragionamenti automatizzati usando le ontologie e ciò fornisce servizi avanzati per le applicazioni intelligenti quali: ricerca semantica/concettuale, supporto alle decisioni, comprensione del parlato e del linguaggio naturale, database intelligenti, e commercio elettronico.

Le ontologie svolgono un ruolo fondamentale nell'emergente Web Semantico come modo di rappresentare la semantica di documenti e fanno sì che tale semantica possa essere utilizzata da parte delle applicazioni web e da agenti intelligenti. Le ontologie possono rivelarsi estremamente utili per una società come modo di strutturare e definire il significato dei termini dei metadati che vengono "collezionati e standardizzati". Usando le ontologie, le applicazioni del domani potranno essere "intelligenti", nel senso che esse potranno lavorare più accuratamente al livello concettuale dell'uomo.

Le ontologie sono critiche per le applicazioni che vogliono cercare o integrare informazioni tra diverse communities. Ciò che si vuole evitare è che uno stesso termine possa essere usato con diversi significati in contesti diversi, o che termini differenti possano avere un identico significato. Le ontologie, con la loro semantica, garantiscono l'interoperabilità tra schemi di rappresentazione sviluppati autonomamente da agenti diversi e sono dunque l'ideale per progetti di grandi dimensioni e che coinvolgono numerosi gruppi di lavoro.

1.2 Perché le ontologie

Le ontologie potrebbero essere definite come le eredi dell'intelligenza artificiale. Evolvendo dalle nozioni della semantic network (la rete semantica), le moderne ontologie si stanno rivelando piuttosto utili e ciò senza basarsi sul *miscuglio* delle tecniche basate su regole, molto comuni nei primi sforzi per la rappresentazione delle informazioni. Queste descrizioni strutturate o "modelli dei fatti noti"(known facts models) sono costruite per rendere un buon numero di applicazioni capaci di gestire le informazioni più complesse e

disparate. Le ontologie web sono più efficaci quando le separazioni semantiche, che per le menti degli uomini sono naturali, sono cruciali nell'ambito dell'applicazione da realizzare. Ciò include la capacità di riuscire a comprendere ciò che è "nascosto" in un estratto di un discorso. La struttura semantica realizzata dalle ontologie si differenzia dalla superficiale composizione e disposizione delle informazioni (i dati) che i database relazionali e XML permettono. Con i database, virtualmente, tutto il contenuto semantico deve essere "catturato" nella logica della applicazione. Le ontologie, invece, sono spesso in grado di fornire una specifica oggettiva delle informazioni del dominio, ponendosi come un *modello comune di rappresentazione dei concetti e delle relazioni* caratterizzanti il modo in cui la conoscenza è espressa in quello specifico dominio.

Questa specifica può essere il primo passo nella costruzione di sistemi di informazione semantically-aware (forniti di semantica) per supportare attività personali, governative e d'impresa.

1.2.1 Tipi di Ontologie

Le ontologie possono variare non solo nel contenuto, ma anche nella loro struttura ed implementazione.

Livelli di descrizione

La costruzione di un'ontologia assume un diverso significato per diversi professionisti. Alcuni descrivono qualcosa che riflette una progressione nelle ontologie a partire da semplici termini o vocaboli, fino a glossari organizzati in categorie e alle tassonomie dove i termini sono relazionati gerarchicamente, ai quali è possibile assegnare proprietà distintive. Altri progettisti preferiscono costituire ontologie complete, dove queste proprietà possono definire nuovi concetti e dove i concetti hanno relazioni con altri concetti, etichettate ad esempio come "cambia l'effetto di" oppure "compra da".

Scopo concettuale

Le ontologie differiscono anche rispetto allo scopo e all'ambito del loro contenuto. La principale separazione che è possibile effettuare, distingue le ontologie di dominio (*domain ontology*) che forniscono gli oggetti specifici della nostra applicazione, ad esempio in campo medico, dalle ontologie di scopo o di alto livello (*task/upper level ontology*) che rappresentano la struttura dei processi, descrivendo i concetti di base e le relazioni esistenti tra essi sulla base delle informazioni sul dominio, espresse in linguaggio naturale. La sinergia tra le ontologie – utilizzabile nelle applicazioni di tipo verticale – si evince dai riferimenti incrociati tra le ontologie di alto livello e le ontologie dei vari domini applicativi.

Istanziazione

Tutte le ontologie hanno una componente che storicamente è stata chiamata la componente terminologica (*the terminologic component*). Questa componente è praticamente analoga a uno schema per un database relazionale o un documento XML. Essa definisce i termini e la struttura dell'ontologia dell'area di interesse. La seconda componente, quella asserzionale, popola l'ontologia con delle istanze che manifestano la definizione terminologica. La linea di separazione tra trattare un'entità come un concetto e trattarlo come un'istanza è solitamente una decisione specifica dell'ontologia.

Linguaggi specifici

Le ontologie non sono tutte costruite allo stesso modo. E' possibile usare una serie di linguaggi, inclusi linguaggi generali di programmazione logica come Prolog. Più comuni sono invece linguaggi che si sono evoluti per supportare la costruzione di ontologie. Il modello Open Knowledge Base Connectivity (OKBC) e i linguaggi come KIF sono diventati la base per la costruzione di altri linguaggi per la costruzione di ontologie. Ci sono anche numerosi linguaggi basati su un tipo di logica pensati per essere computabili,

conosciuti come description logics (logiche di descrizione); questi includono DAML+OIL il quale sta per diventare il linguaggio standard per le ontologie web.

La comparazione tra i linguaggi ontologici si effettua solitamente sulla base dell'espressività e della semplicità, il che non è sempre una scelta sbagliata. Tuttavia, un linguaggio ha bisogno di essere ricco ed espressivo quanto basta per rappresentare le sfumature e la complessità delle informazioni che lo scopo dell'ontologia e i suoi sviluppatori richiedono. In ambiente W3C si ricorre agli schemi RDF come livello di linguaggio, gli schemi XML per la definizione dei dati e l'RDF per l'asserzione dei dati.

1.3 La descrizione del web e i metadati

Uno dei problemi più comuni che i navigatori del web si trovano a dover affrontare è la ricerca di informazioni in maniera accurata. Per fare ciò occorrerebbe classificare le pagine Web ricorrendo a descrizioni del contenuto delle pagine stesse, tramite quelli che tecnicamente vengono chiamati metadati. I metadati sono dati che descrivono altri dati. Ad esempio, i campi titolo, autore, genere, ecc. sono metadati che descrivono libri (o film, articoli) e che vengono comunemente utilizzati nella definizione di tabelle in un database. Estendere questo stesso concetto al web non è così immediato, in quanto la natura non centralizzata di Internet non permette di definire in maniera univoca dei campi da utilizzare per la descrizione delle risorse. Tanto più che il tipo di informazioni da classificare è talmente vasto e imprevedibile che qualsiasi insieme di campi si tenti di definire, sarebbe insufficiente.

1.3.1 Confronto con altri schemi di rappresentazione

Per lo scambio dei dati e per la ricerca delle informazioni è possibile utilizzare altri tipi di rappresentazione e descrizione dei contenuti, sotto forma di schemi; i più usati tra questi sono gli schemi XML e RDF (XML e RDF Schemas).

1.3.2 XML

XML è l'acronimo di eXtended Markup Language. A dispetto del nome non si tratta propriamente di un linguaggio, ma di un meta linguaggio, cioè un linguaggio per costruire altri linguaggi.

Questo linguaggio nasce in risposta alle limitazioni espressive dell'HTML nella descrizione delle pagine web. Con XML è possibile dare un'organizzazione strutturale alle informazioni.

```
<?xml version="1.0" standalone="yes"?>
   <rubrica>
    <persona>
     <nome>Giampiero</nome>
     <cognome>Granatella</cognome>
     <telefono_casa> 081123456 </telefono_casa>
     <telefono_ufficio> 081654321 </telefono_ufficio>
     <indirizzo_email> giampi@unina.it </indirizzo_email>
   </persona>
   <persona>
    <nome>Giovanni </nome>
    <cognome>Bianchi</cognome>
    <telefono_casa> 010123456 </telefono_casa>
    <telefono_ufficio> 010654321 </telefono_ufficio>
    <indirizzo_email> giovanni.bianchi@unina.it </indirizzo_email>
   </persona>
</rubrica>
```

Esempio di costruzione di una rubrica in XML

1.3.2.1 Schemi XML

Gli schemi XML sono documenti utilizzati per definire e convalidare il contenuto e la struttura dei dati XML, nello stesso modo in cui uno schema di database definisce e convalida le tabelle, le colonne e i tipi di dati da cui è costituito un database.

Uno schema XML consente di definire e descrivere determinati tipi di dati XML mediante il linguaggio XSD (XML Schema Definition). Gli elementi degli schemi XML, ovvero elementi, attributi, tipi e gruppi, vengono utilizzati per definire una struttura valida, un contenuto dei dati valido e le relazioni di determinati tipi di dati XML. Gli schemi XML possono inoltre fornire valori predefiniti per attributi ed elementi..

Lo schema XML viene utilizzato per garantire la coerenza tra determinati tipi di dati XML

condivisi tra applicazioni e organizzazioni. È possibile utilizzare uno schema XML come *contratto* per lo scambio dei dati tra due applicazioni. Le organizzazioni possono pubblicare gli schemi che descrivono il formato XML prodotto e utilizzato dalle applicazioni. Altre organizzazioni e applicazioni interessate allo scambio dei dati possono quindi generare le applicazioni basandosi su questi schemi in modo che sia possibile interpretare i messaggi XML.

È ad esempio possibile convalidare un ordine di acquisto rappresentato in XML con uno schema XML prima che venga scambiato tra l'acquirente e il rivenditore. Questa convalida consente di verificare che tutti i singoli elementi dei dati siano disponibili, siano nella sequenza prevista e del tipo di dati corretto, garantendo in tal modo che il destinatario dell'ordine di acquisto sia in grado di interpretare correttamente i dati alla ricezione.

Di seguito sono riportati alcuni dei vantaggi offerti dagli schemi XML rispetto alle tecnologie precedenti:

Negli schemi XML viene utilizzata la sintassi XML, per cui non è necessario apprendere una nuova sintassi per definire la struttura dei dati.

Negli schemi XML sono supportati tipi riutilizzabili ed è consentita la creazione di nuovi tipi mediante ereditarietà.

Gli schemi XML consentono di raggruppare elementi per controllare la ricorrenza di elementi e attributi.

1.3.3 RDF

Allo scopo di fornire un meccanismo flessibile per la descrizione delle risorse sul Web, il W3C ha definito un modello astratto denominato **Resource Description Framework** (RDF). Questo modello di descrizione si basa su tre tipi di oggetti:

Risorse (resources)

Una risorsa può essere rappresentata da una pagina, un gruppo di pagine, un'immagine, un server o un qualsiasi altro elemento che abbia un Universal Resource Identifier (URI), cioè

un meccanismo di identificazione, come l'indirizzo Web.

Proprietà (properties)

Una proprietà è una specifica caratteristica o attributo di una risorsa; una proprietà può anche descrivere relazioni con altre risorse.

Asserzioni (statements)

Una asserzione è costituita dall'insieme di una risorsa, una proprietà e uno specifico valore per quella proprietà e descrive le caratteristiche di una risorsa e le relazioni con altre risorse.

Ad esempio, la frase:

La pagina all'indirizzo www.cplusplus.it/tipetriks.htm ha per argomento trucchi

è un'asserzione costituita dalla **risorsa** pagina all'indirizzo www.cplusplus.it/tipetriks.htm, dalla **proprietà** argomento e dal **valore** trucchi

1.3.3.1 Schemi RDF

RDF è un modello di descrizione astratto e non pone vincoli sulla sintassi e sul significato delle descrizioni di una risorsa. Questo vuol dire che ognuno potrebbe proporre un qualsiasi meccanismo per descrivere risorse utilizzando le astrazioni previste da RDF. Ad esempio, una descrizione RDF potrebbe essere espressa tramite una rappresentazione grafica delle risorse, degli attributi e delle asserzioni. Tuttavia, per ragioni pratiche, la definizione formulata del W3C propone XML come metalinguaggio per la rappresentazione del modello RDF, cioè propone RDF come linguaggio basato su XML. In pratica, una descrizione RDF è un documento XML contenente alcuni tag predefiniti.

Nella descrizione RDF si fa uso della tecnica dei *namespace* di XML. Un namespace è un contesto che definisce un insieme di tag da utilizzare in un documento XML. Nell'esempio riportato, il tag **<rdf:RDF>** indica che tale tag è definito nel namespace identificato dal prefisso rdf, la cui dichiarazione si trova nella prima riga del documento XML. In tale riga è presente la direttiva di elaborazione che indica anche l'indirizzo presso cui è pubblicata la definizione del namespace.

I tag propriamente descrittivi di una descrizione RDF sono liberi, nel senso che ciascuno è libero di inventarsi i tag che preferisce. Tuttavia questo comporterebbe la possibilità di ritrovarsi tag diversi con lo stesso significato. Ad esempio, il tag **<argomento>** potrebbe essere definito come **<subject>** da un anglosassone. Per evitare il sovrapporsi di tag diversi con lo stesso significato, RDF prevede un meccanismo per creare vocabolari o schemi, cioè insiemi di proprietà ed elementi che definiscono un contesto per la descrizione di determinate categorie di risorse. Un noto esempio di schema RDF è il *Dublin Core Metadata*: esso definisce i tag da utilizzare per la descrizione di documenti elettronici (libri, articoli, rapporti, ecc.).

```
<?xml:namespace ns="http://www.w3.org/1999/02/22-rdf-syntax-ns#" prefix="rdf" ?>
<?xml:namespace ns="http://purl.org/metadata/dublin_core#" prefix="dc" ?>
<rdf:RDF>
  <rdf:Description about="http://www.html.it">
    <dc:Title>HTML.IT</dc:Title>
    <dc:Description>
Il portale italiano su HTML e dintorni
</dc:Description>
<dc:Publisher>HTML.IT srl</dc:Publisher>
<dc:Format>text/html</dc:Format>
<dc:Language>it</dc:Language>
    </rdf:Description>
</rdf:RDF>
```

Esempio di descrizione del sito HTML.IT in RDF utilizzando lo schema Dublin Core Metadata

1.3.4 La vittoria delle ontologie

Sebbene gli schemi XML siano sufficienti per lo scambio di informazioni tra parti che si siano accordate preventivamente sulle definizioni dei dati(e sui namespace), la loro mancanza di semantica non consente alle macchine di effettuare questa operazione una volta aggiunti nuovi "vocaboli". Presentano inoltre problemi nella scelta dei termini: uno

stesso termine può essere usato con significati (a volte leggermente) diversi in differenti contesti, e termini differenti possono essere usati per definire elementi che hanno il medesimo significato.

L'RDF e gli schemi RDF iniziano a interessarsi al problema consentendo l'associazione di semantiche attraverso degli identificatori. Con gli schemi RDF si possono definire classi che hanno molte sottoclassi e superclassi, si possono definire proprietà, le quali possono avere sottoproprietà, domini e intervalli. In questo senso uno schema RDF è un semplice linguaggio per le ontologie. In ogni caso, però, per consentire l'interoperabilità tra schemi che siano numerosi, sviluppati e gestiti autonomamente, sono necessarie semantiche più ricche. Ad esempio uno schema RDF non può specificare una relazione di disgiunzione, o la cardinalità di proprietà e di attributi di classe.

Capitolo 2

Il Semantic Web

2.1 Cos'è il Web Semantico

Internet è un insieme di testi, un insieme vastissimo di documenti nato con lo scopo di descrivere contenuti. L'esistenza di una raccolta di testi non rappresenta in se stessa una novità, in quanto fin dall'antichità sono esistiti vasti corpi di testi come le antiche biblioteche. La novità che ha portato Internet è ciò che lo ha reso quello che oggi è: un fenomeno di massa che è in continua ascesa e non accenna a diminuire. L'innovazione apportata è la capacità che questi documenti hanno di richiamarsi l'un l'altro, in modo rapido. E' il *link ipertestuale*, l'elemento nuovo che l'HTML ha saputo proporre. L'ambiente Internet non si è fermato alla semplice raccolta di dati e alla loro connessione attraverso link ipertestuali, ma sono nati nuovi servizi in grado di fornire eccezionali capacità al web. Tra questi si possono segnalare i motori di ricerca, che tentano di accedere direttamente al contenuto dei testi e alle strutture dei siti che attraverso database, script cgi, fogli di stile, rendono sempre più *user-friendly* l'approccio con gli utenti. Tuttavia, un navigatore si orienta nel web principalmente grazie a due cose: la sua **esperienza** di navigazione e la capacità di evocazione che possono avere talune parole o **espressioni chiave**. L'esperienza è un aspetto molto importante; impariamo che determinati contenuti si possono reperire in determinati portali, che l'aspetto di un sito può, in maniera formale o informale, dirci qualcosa sul genere dei contenuti trattati. Ciò

nondimeno, l'esperienza non è legata ad aspetti tecnici, né al codice e alle applicazioni che costituiscono un sito o un portale. L'altro aspetto, quello delle parole chiave, è invece strettamente collegato alla costruzione di applicazioni e servizi.

La capacità espressiva di un link ipertestuale dipende dalla applicazione che lo gestisce. Nelle ricerche e nelle attività tipiche del web, si è sempre esposti a un rischio di ambiguità. Da una query effettuata attraverso un motore di ricerca, si ottengono risultati che possono coprire vari ambiti; si pensi alla parola "albero". La ricerca presenterà collegamenti relativi all'informatica, alla botanica, alla nautica e non allo specifico settore di interesse. Ciò che si vuole evidenziare è semplicemente il fatto che i collegamenti di Internet che legano fra loro i documenti, sono deboli, nel senso che essi sono troppo generici e vaghi. I collegamenti a cui si fa riferimento non sono quelli sintattici, cioè quelli relativi al funzionamento del codice di programmazione delle applicazioni, ma quelli legati alla capacità di esprimerne il significato, la semantica. Sarebbe auspicabile l'esistenza di link che oltre a condurre al documento collegato, siano in grado anche di *descriverne* il contenuto. Questo si intende per **capacità semantica.**

La sfida è dunque quella di scrivere codice in grado di compiere operazioni semantiche. Ma come? Nella costruzione di un sito web è possibile definire una certa struttura delle informazioni, ad esempio inserirle in un database relazionale, in modo da rendere più semplice per l'utente la ricerca delle stesse con l'aiuto di vincoli. Si è stabilito di utilizzare uno schema preciso per archiviare quei dati. Lo schema è un insieme di regole che stabiliscono come i dati debbano essere organizzati. Poiché uno schema definisce anche le relazioni fra i dati, è anche in grado di esprimere vincoli che legano o oppongono due classi di dati. L'idea del web semantico nasce dunque estendendo l'idea di utilizzare schemi per descrivere domini di conoscenza. Un dominio deve essere descritto da un particolare schema: devono trovar posto metadati (dati per descrivere dati) per mappare le informazioni rispetto a classi o concetti di questo schema di dominio.

Quando si parla di web semantico si intende un web che possieda delle strutture di collegamenti più espressive di quelle attuali. Il termine *semantic web* è stato proposto per

la prima volta nel 2001 da Tim Berbers Lee. Da quel momento il termine è stato usato per identificare un web in cui agiscano agenti intelligenti: applicazioni in grado di comprendere il significato di testi presenti in rete e pertanto capaci di guidare l'utente verso l'informazione cercata o di sostituirsi a lui nello svolgimento di alcune operazioni. Gli obiettivi che si pone il web semantico sono molto interessanti e hanno affascinato la comunità informatica: il W3C (*World Wide Web Consortium*) ha attivato immediatamente un gruppo di lavoro. Anche le Università e le istituzioni private hanno aperto numerosi programmi di ricerca legati a queste tematiche.

Per soddisfare i requisiti voluti, il web semantico deve comporsi di tre livelli fondamentali. Al livello più basso troviamo i dati, un passo sopra sono presenti i metadati che riportano questi dati ai concetti di uno schema e al livello più alto vi sono gli schemi (chiamati ontologie) che esprimono le relazioni fra concetti che diventano classi di dati. Le ontologie, solitamente, hanno lo scopo di organizzare un dominio specifico, ma come spesso accade un'informazione non ha valore solo in un contesto, ma può essere utile in situazioni diverse. Più di frequente è solo una parte dell'informazione che va recuperata e non l'intero contenuto informativo. Questo richiede uno schema di organizzazione dei dati in grado di suddividere il dominio in tutte le classi di oggetti che hanno un ruolo nei suoi processi. Per garantire la diffusione globale di questa proposta è necessaria la costruzione di una architettura basata su concetti e regole accettate da tutti. Come sempre accade, ciò non è facile e richiede tempo e compromessi; si deve inoltre tenere presente che queste tecnologie sono state sviluppate dai professionisti del settore e la loro diffusione nelle comunità di Internet sarà lenta e graduale.

2.2 Le ontologie nell'architettura del Web Semantico

L'architettura del web semantico, deve avere la proprietà di essere universale, deve cioè consentire a tutti i costruttori di ontologie di potersi scambiare informazioni e di integrare applicazioni ontologiche. E' da notare che ogni operazione applicata ad un dominio,

utilizzerà una sua propria ontologia; nasce quindi la difficoltà di collegare fra loro le varie ontologie, conciliando le diverse semantiche utilizzate. L'operazione di collegare ontologie che definiscono stessi oggetti con una semantica differente si chiama *Mapping*. Si potrebbe pensare di risolvere il problema, cercando di ricorrere ad una "ontologia globale" che richiederebbe un unico mapping e avrebbe il vantaggio di operare come una lingua universale, una traduzione dei diversi linguaggi usati. Questo tipo di soluzione richiederebbe un accordo totale ad ogni livello del significato della semantica usata. Ciò non è praticabile, almeno per il semplice fatto, che qualora si riuscisse a creare un'unica ontologia, essa sarebbe statica, non sarebbe cioè in grado di adattarsi a quei mutamenti del dominio descritto, che inevitabilmente nascono col passare del tempo.

Un'alternativa concreta sarebbe la creazione di standard detti solitamente *interlingua*. Alcuni gruppi di ontologie relative a domini e a scopi simili o collegati tra loro parzialmente, condividono una stessa ontologia, detta appunto interlingua. Ciò minimizzerebbe il numero di mappature, poiché non sarebbero più tutte le singole ontologie a dover essere conciliate, ma si ricorrerebbe alle interlingua che da par loro, contengono ontologie mappate per dominio e/o scopo.

2.3 I metadati

I metadati sono componenti fondamentali dell'architettura del web semantico. Tutti i livelli dell'architettura necessitano di avere legami o utilizzare i metadati. L'utilità di questi ultimi sfugge a coloro che cercano di farsi una prima opinione sul web semantico. In realtà, senza i metadati l'intera architettura del Semantic Web semplicemente non funzionerebbe.

La scelta effettuata dai progettisti del web semantico è quella di non interrogare i testi, ma i metadati. Questa scelta è motivata dal fatto che i dati nel web non hanno nessuna struttura; i metadati invece sono informazioni su dati che sono state prodotte seguendo una struttura ben precisa. Il potersi rifare ad una struttura permette di manipolare i dati, conoscendo le relazioni che intercorrono fra essi.

Intuitivamente è semplice capire come definire una strutturazione dei dati (costruire metadati). Se ad esempio si ha a che fare con l'anagrafica di un individuo, è lecito aspettarsi una descrizione dei dati di questo tipo.

Dato	Metadato
Cristina	Nome
Fiotti	Cognome
Via Giuseppe Verdi 25	Indirizzo
Milano	Città

Tuttavia, pur essendo la base su cui costruire tutto il sistema, i metadati costituiscono anche il punto debole del web semantico. I metadati, infatti, sono molto costosi da produrre e possono risultare imprecisi. Se la produzione dei metadati avviene attraverso un'*indicizzazione* manuale, questa risulterà costosa in termini di tempo e costi-uomo. Questa può apparire imprecisa se non è realizzata da persone esperte e motivate. Inoltre, la creazione dei metadati è un'attività molto importante ma allo stesso tempo molto complessa e ciò è dovuto alle molteplici interpretazioni che un metadato può presentare.

Vale la pena precisare dunque cosa sono i metadati e cosa si può fare con essi. Innanzitutto occorre tener presente che i metadati non sono dati. Questi non possono limitarsi a riportare in modo stringato i contenuti di un testo. I metadati dovrebbero esprimere relazioni di un documento in un certo contesto e definirne relazioni trasversali.

I metadati non hanno tutti lo stesso valore, alcuni valgono più di altri. Un metadato descrive la struttura di una classe, di un tipo di dato, ecc. Ciò può farlo in due modi molto diversi. Il modo più agevole consiste nel riempire il valore di un campo, di una proprietà della classe (una Persona che si chiama "Luca"). In questo caso si attribuisce solo un valore ad una struttura, ma non si indica nulla di più preciso. Se si trovasse un altro metadato per la persona di nome Luca, non si saprebbe mai se è la stessa persona o sono due diverse. Una seconda scelta è far sì che un metadato punti ad una risorsa e utilizzi

quella per definire il valore della proprietà di una classe (una persona che si chiama "http:/dominio.nomi.com/#Luca"). In questo caso il metadato compie un'operazione molto più accurata, in quanto individua qualcosa di univoco, di cui si potrà sapere con esattezza quali altri metadati sono stati espressi. Il meccanismo è ancora più potente se si indica la localizzazione di risorse fisiche, come pagine web, immagini o documenti.

Il problema che subito si pone, nella localizzazione dei contenuti, è quello dell'evoluzione dei contenuti stessi. Come sappiamo il Web è dinamico, cioè cambia molto rapidamente, dal momento che si interviene quotidianamente su di esso (aggiunta di nuove pagine, modifica delle pagine esistenti). Tali variazioni possono avvenire ad intervalli regolari o in qualunque momento, e riguardare parti non inerenti al contenuto informativo (come la correzione di errori di sintassi, il cambiamento di formattazione) oppure coinvolgere dati significativi quali ad esempio la rimozione di pagine o di interi siti. Tutto questo, in aggiunta alla mancanza di un controllo centralizzato, fa sì che i dati possano essere (i) *inconsistenti*, per la mancanza di vincoli di qualunque tipo, (ii) *inaffidabili*, perchè non sono controllati in alcun modo, (iii) *non disponibili*, se la pagina in cui sono contenuti è rimossa. Pertanto è possibile distinguere i metadati, in base ai dati che essi strutturano, in metadati *vivi* e *morti*. E' diverso definire un metadato per un documento di cui si possiede in locale una copia piuttosto che di un documento di cui si possiede una URL (Unified Resource Locator) ovvero una localizzazione nella rete. Nel primo caso si è certi di poter effettivamente disporre di quel documento, nel secondo caso si rischia che l'URL non localizzi nulla; i link invecchiano piuttosto facilmente. Nel primo caso è possibile che si forniscano informazioni vecchie, nel secondo i dati saranno aggiornati in tempo reale. Le scelte dipendono molto da cosa si intende fare e dai mezzi che si hanno a disposizione. Una via di mezzo può essere quella di registrare sia una URL sia una copia in cache.

2.4 I linguaggi per la costruzione di ontologie

Per essere utili, le ontologie devono essere espresse in una notazione concreta. Un "linguaggio per ontologie" è un linguaggio formale con cui un'ontologia viene costruita.

Un linguaggio, per soddisfare le necessità delle ontologie e del web semantico menzionate in precedenza, deve possedere una serie di requisiti: (i) deve *estendere* standard Web esistenti (XML, RDF) per semplificare il suo utilizzo (ii) deve essere *facile da capire e da usare* (iii) deve essere specificato in modo *formale* e (iv) deve possedere un *potere espressivo* adeguato al dominio da descrivere. Due linguaggi importanti per la costruzione di ontologie sono OWL e DAML+OIL.

OWL è un linguaggio per definire ontologie strutturate basate sul web che permettano maggiore integrazione ed interoperabilità di dati tra le applicazioni. I primi ad adottare questi standard comprendono bioinformatici e comunità mediche, gruppi industriali e governi. OWL permette di eseguire una gamma di applicazioni descrittive come la gestione di portali Web, la gestione di collezioni, ricerche basate sul contenuto, abilitando agenti intelligenti e servizi web. I primi linguaggi sono stati usati per sviluppare strumenti e ontologie per specifiche comunità di utenti (in particolare nelle scienze e in applicazioni di commercio elettronico specifiche delle aziende); essi, però, non erano stati definiti per essere compatibili con l'architettura del World Wide Web in generale e del web semantico in particolare.

OWL ovvia a questo inconveniente utilizzando degli URI (Universal Resource Identifier) per identificare una risorsa e il linking fornito da RDF per aggiungere le seguenti caratteristiche alle ontologie:

- Capacità di essere distribuite tra più sistemi;
- Scalabilità per le necessità del Web;
- Compatibilità con gli standard Web per quanto riguarda l'accessibilità e l'internazionalizzazione;

- Apertura ed estendibilità.

OWL è un linguaggio per definire ontologie strutturate basate sul Web che permettono una integrazione ed una interoperabilità maggiore di dati tra comunità che descrivono il loro dominio di conoscenza. OWL si basa sul modello "RDF e RDF Schema" e aggiunge un vocabolario più ampio per descrivere proprietà e classi: questo comprende relazioni tra classi (ad esempio disgiunzione), cardinalità (ad esempio "esattamente uno"), uguaglianza, tipizzazione più ricca delle proprietà, caratteristiche di proprietà (ad esempio simmetria) e classi enumerate.

DAML+OIL è un linguaggio standard che consente la rappresentazione delle informazioni del web in modo che il loro significato sia comprensibile alle macchine. Originariamente si trattava di due linguaggi distinti. DAML(*DARPA Agent Markup Language*) consentiva di descrivere il contenuto semantico dei dati, basandosi sulle ontologie definite con RDFS; OIL(*Ontology Inference Language*) è un linguaggio, web-based, per la rappresentazione e inferenza di ontologie, che unisce i largamente usati linguaggi di modellazione basati su frame con la semantica formale fornita dalla descrizione logica. E' compatibile con gli schemi RDF(RDFS), e include una semantica precisa per la descrizione del significato dei termini (e anche per descrivere informazioni implicite).

E' stato evidente, in seguito, che i due prodotti potevano essere uniti e il risultato é un linguaggio ontologico, che consente di descrivere la struttura di un dominio. DAML+OIL propone un approccio "object-oriented" e la strutturazione è effettuata in termini di classi e proprietà: un'ontologia in questo contesto è un insieme di assiomi, che dichiarano le relazioni di classificazione tra le classi o le proprietà. Un aspetto importante di tale linguaggio, riguarda i tipi di dati: DAML+OIL supporta tutti i tipi degli schemi XML, garantendo così una compatibilità con le applicazioni esistenti e semplificandone l'apprendimento da parte dei costruttori di ontologie.

Capitolo 3

Costruire un'ontologia

3.1 Le fasi di costruzione

I passi principali per costruire un'ontologia sono semplici e lineari. Esistono varie metodologie per guidare l'approccio alle ontologie e sono disponibili numerosi tool che seguono il processo di costruzione di un'ontologia. Il problema è che queste procedure non sono converse in stili o protocolli di sviluppo standard, e i tool non hanno ancora raggiunto in questo campo il livello di qualità che invece essi dimostrano in altre applicazioni software. Inoltre manca un supporto completo per i più recenti linguaggi ontologici.

Un'ontologia è tipicamente costruita seguendo più o meno questi step:

1. **Acquisire la conoscenza del dominio**

   Raccogliere quante più informazioni possibili sul dominio di interesse, comprendere i termini usati formalmente per descriverne le entità in maniera consistente, in collaborazione con gli esperti del dominio. Queste definizioni devono poi essere collezionate per poter essere espresse in un linguaggio comune scelto per l'ontologia. Le domande da porsi sono pressoché le seguenti: (i) quale dominio coprirà l'ontologia? (ii)

qual è lo scopo dell'ontologia? (iii) a quali tipi di domande l'informazione espressa dall'ontologia può fornire risposte? (iv) chi userà e chi sarà il responsabile della manutenzione dell'ontologia?. Risulta evidente che non tutte le risposte sono note a priori.

2. **Organizzare l'ontologia**

Progettare la struttura concettuale complessiva del dominio. Questa operazione semplifica l'identificazione dei principali concetti del dominio e delle loro proprietà, identificando le relazioni tra i concetti, creando concetti astratti, identificando quali di questi hanno delle istanze ecc. Le domande da porsi in questa fase sono: (i) quali sono i termini importanti? (ii) quali sono le proprietà? Vi sono tre passi fondamentali: (1) **flat glossary**, che consiste nel documentare ciascun termine con una definizione in linguaggio naturale (e fornire esempi dove appropriato) in cui i nomi diventano oggetti o attori, e i verbi si trasformano in relazioni o processi, (2) **structured glossary** che consiste nella decomposizione e/o specializzazione dei termini e nell'individuazione degli attributi di un concetto(*predicazione*), (3) identificare tutte le relazioni concettuali fra gli oggetti.

3. **Popolare l'ontologia**

Aggiungere concetti, relazioni, ed entità fino a raggiungere il livello di dettaglio necessario a soddisfare gli obiettivi dell'ontologia. Per individuare nuovi concetti è possibile adottare tre tipi di approcci: (i) top-down, che prevede l'identificazione dei concetti generali e attraverso un raffinamento successivo si procede verso i concetti particolari (es. da computer a workstation), (ii) bottom-up, che procede per livelli di astrazione, partendo dalle entità particolari del dominio per astrarre i concetti generali che racchiudono o fanno uso di quelli particolari (da workstation a computer) e (iii) middle-out (o combinato) che prevede di individuare prima i concetti salienti e poi generalizza e specializza; i concetti "nel mezzo" tendono ad essere i più descrittivi del dominio
(da Computer a: oggetto e entità (is-a), Workstation (has-kind)). I concetti da soli non

forniscono informazioni sufficienti; è importante definire anche le relazioni tra gli oggetti del dominio. Esistono vari tipi di relazioni: di attributi costanti, fisiche e sociali (peso, codice fiscale), di composizione. Inoltre è consigliabile imporre dei vincoli sulle relazioni, quali la cardinalità (ad esempio un computer *ha [1,n]* processori) sul dominio e sul codominio della relazione, sul tipo di valore (un indirizzo *ha un* CAP il cui codominio è di tipo *stringa*, una bottiglia ha un'etichetta con un codominio di tipo *istanza*, un'idea coinvolge concetti con un codominio di tipo *concetto*).

4. **Controllare il proprio lavoro**

   Risolvere inconsistenze sintattiche, logiche e semantiche tra gli elementi dell'ontologia. I controlli di consistenza possono anche favorire una classificazione automatica che definisce nuovi concetti sulla base delle proprietà delle entità e delle relazioni tra le classi.

5. **Considerare il riuso di risorse esistenti**

   E' sempre utile pensare di rifinire ed estendere risorse esistenti, quali glossari, dizionari dei termini e dei sinonimi, documenti, standard e altre ontologie.

6. **Consegnare l'ontologia**

   Al termine dello sviluppo di un'ontologia, così come per qualunque altro sviluppo software, è necessaria una verifica da parte degli esperti del dominio e la seguente consegna del prodotto, assieme a tutti i documenti relativi.

3.2 Le regole da seguire

Si possono fornire alcune regole generali che possono aiutare a ottenere una buona modellazione. Può essere buona norma **non dare alle classi nomi al volte al plurale, a volte al singolare**. La classe rappresenta solo una categoria, uno schema. Spesso si

assegnano nomi al plurale pensando alla classe come a un contenitore, ma questa è una cattiva interpretazione delle classi nell'ontologia. Un'altra cosa da tener presente è il fatto che l'albero dell'ontologia deve essere **bilanciato nella granularità**. Se una classe A di livello 5, perché ha su un ramo un padre 2, un padre 3, e un padre 4, ha su un altro ramo come padre una classe B di livello 2, significa che probabilmente non si è approfondito sufficientemente la granularità del ramo del concetto B. In pratica ci si verrebbe a trovare con concetti collegati fra loro, ma uno su un ramo molto più profondo dell'altro. Questo indica un "buco" nella definizione dei concetti e suggerisce che si debba colmare questa discrepanza inserendo nuovi concetti sul ramo B. Un ragionamento analogo si può fare nel caso in cui ci si trovasse con una classe A con un numero molto alto di figli. Questa situazione, se non è chiaramente giustificata, è indice di un deficit di definizione. Quello che bisogna fare è quindi ricercare proprietà comuni alle varie classi figlie di A in modo da raggrupparle con l'aggiunta di nuove classi. All'opposto, è una scelta strana e **deprecabile modellare una classe con un unico figlio**. Suddividere una classe significa stabilire che alcune proprietà e caratteristiche devono riferirsi ad oggetti diversi; suddividere perciò con un'unica classe pare pertanto ingiustificato.

Una buona ontologia deve possedere le seguenti caratteristiche:

- Prevede tutte le distinzioni chiave;
- Non fa assunzioni implicite;
- E' chiara e concisa: ciascun concetto è rilevante e non ci sono duplicati;
- E' coerente: permette la presenza di tutte e sole le inferenze che sono consistenti con le definizioni dei concetti;
- Richiede scelte di progetto motivate (design options);
- Aderisce a un *ontological commitment* (accordo sull'uso di un "vocabolario" consistente con il dominio di interesse);
- E' modificabile;

- E' riusabile.

La modellazione è un processo iterativo. Non c'è un solo modo *corretto* di modellare un dominio, la soluzione dipende dall'applicazione e dalle estensioni previste; un'ontologia è un modello di descrizione della realtà e i concetti definiti riflettono questa realtà. Un'ontologia **non** può contenere tutte le informazioni possibili sul dominio, né è in grado di esprimere tutte le possibili proprietà e distinzioni tra concetti nella gerarchia.

Importanti risultano anche le convenzioni di nomenclatura (*naming conventions*), cioè l'operazione di definire una convenzione per concetti, istanze e relazioni a cui aderire. Alcune semplici regole guidano questo processo. Bisogna evitare sovrapposizioni, cioè non assegnare uno stesso nome per un concetto e per una relazione, aggiungere capitalizzazioni e delimitatori (dot-notation, tratto), utilizzare prefissi e suffissi per le relazioni (*ha*-produttore, produttore-*di*), distinguere gli *oggetti* dai *processi* (prenotazione, fatturazione vs. prenotare, fatturare), non usare abbreviazioni (Cod. Avv. Post. per Codice Avviamento Postale, Doc. per Documento ecc.). Risulta chiaro che è soprattutto l'esperienza che aiuta nel processo di costruzione di un'ontologia, ma è ugualmente evidente che documentare ogni passo dello sviluppo, annotare i problemi riscontrati e le eventuali soluzioni, può aiutare gli utilizzatori e gli stessi progettisti in vista di successivi cambiamenti.

Il lavoro della modellazione non è insomma facile. Può essere utile farsi aiutare da un tool. Un tool può venire in soccorso allo sviluppatore sostanzialmente in due modi: (i) fornire una visualizzazione grafica dell'ontologia, e quindi avere una visione d'insieme delle relazioni fra le classi e (ii) evitare di scrivere il codice a mano, riducendo gli errori involontari. Il codice generato, ovviamente, è molto semplice (almeno per ora) e deve quindi essere visionato dal modellatore per essere modificato e reso più adatto agli attuali obiettivi e ad un eventuale utilizzo futuro.

Capitolo 4

Tools per la costruzione di ontologie

4.1 Introduzione

Negli ultimi anni, il numero di tool sviluppati per la creazione di ontologie, dalle comunità Americane ed Europee è elevato. Quando si vuole costruire un'ontologia sorgono numerose domande per scegliere quale tool utilizzare: quale tool darà maggiore supporto al processo di sviluppo di un'ontologia? In che modo vengono memorizzate le ontologie(in database o in file ASCII)? Il tool possiede un motore di inferenza(inference engine)? Il tool ha la capacità di convertire l'ontologia in linguaggi diversi da quello utilizzato? E' cioè corredato da traduttori di linguaggi di ontologie? Qual è la qualità delle traduzioni? Come possono le applicazioni interoperare con i server di ontologie?, etc. Questo capitolo risponde ad alcune di queste domande. Saranno presentati e comparati i più importanti e utilizzati tool di sviluppo che sono apparsi negli ultimi anni. Come prima cosa, nella sezione 4.2 saranno presentati i principali criteri utilizzati per comparare i diversi tool. Nella sezione 4.3 saranno analizzati in dettaglio alcuni di questi tool. Nel paragrafo 4.3.14 si effettueranno una serie di considerazioni generali sull'analisi compiuta.

4.2 Criteri per comparare i tool

Questa sezione presenta i criteri che saranno usati per comparare i tool di sviluppo di ontologie. Possiamo suddividere tali criteri nei seguenti gruppi:

Descrizione generale dei tool, che include informazioni riguardo gli sviluppatori, le versioni e la disponibilità del prodotto.

Architettura del software ed evoluzione del tool, che include informazioni riguardo l'architettura del tool (applicazione standalone, client/server, multi-livello), le modalità con cui il tool può essere esteso con altre funzionalità/moduli, come le ontologie vengono memorizzate (database, file di testo, ecc.) e se esiste un sistema di gestione per il backup dei dati.

Interoperabiltà con altri tool e/o linguaggi, che include informazioni riguardo le capacità di operare con altri tool. Si specificheranno i tipi di interazione con gli altri tool(per fusione, memorizzazione, inferenza, ecc.), e le capacità di traduzione da e verso linguaggi per la specifica di ontologie.

Rappresentazione della conoscenza. Si analizzerà il paradigma che sottende il "knowledge model" del tool. Ciò è molto importante per comprendere quali informazioni e come queste possono essere modellate.

Servizi di inferenza allegati al tool. Si analizzerà se il tool ha un motore di inferenza incorporato(built-in) oppure se può usare motori esterni. Inoltre si verificherà se il tool

é in grado di effettuare controlli dei vincoli e di consistenza, se può classificare in modo automatico i concetti in tassonomie e se può gestirne le eccezioni.

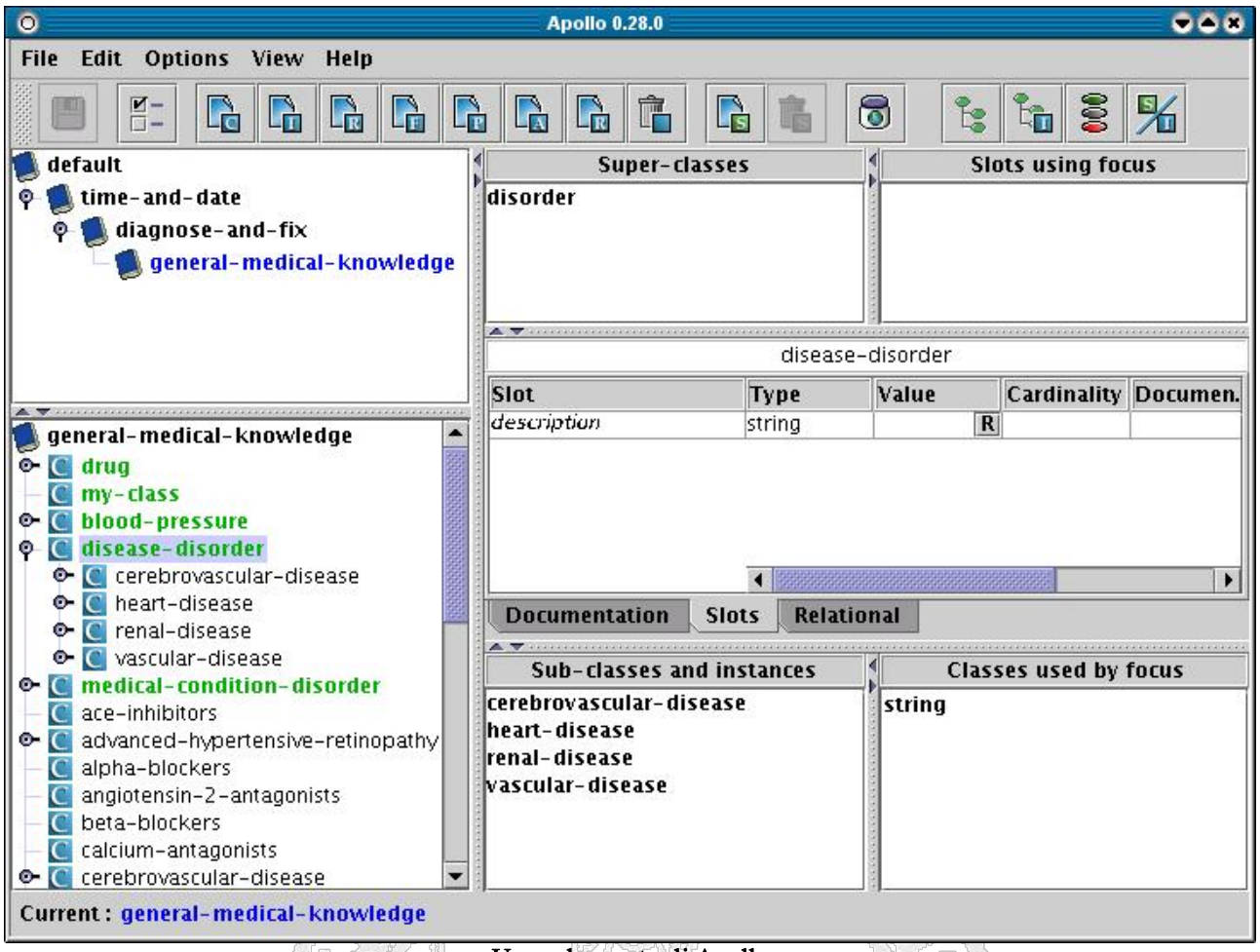
Usabilità. Si verificherà l'esistenza di editor grafici per la realizzazione di relazioni e tassonomie di concetti, la capacità di potare questi grafici e la possibilità di ingrandire, zoomare parti di essi. Inoltre si analizzerà se il tool consente di lavorare in modo collaborativo e se è corredato di librerie di ontologie.

4.3 Tool per lo sviluppo di ontologie

In questa sezione, si cercherà di fornire una vasta descrizione di alcuni dei tool e ambienti disponibili per la costruzione di ontologie a partire da zero o riutilizzando ontologie esistenti. Si fornirà una breve descrizione di ogni tool, presentando il gruppo che lo ha sviluppato, le principali caratteristiche e funzionalità, le relazioni relative con i formalismi del KR, (Knowledge Representation), ecc. Inoltre si fornirà l'URL (Universal Resource Locator) per ogni prodotto e riferimenti bibliografici (se disponibili) per consentire al lettore interessato di approfondire gli argomenti trattati.

4.3.1 Apollo

Apollo è un'applicazione user friendly per lo sviluppo di ontologie sviluppato dal Knowledge Media Institute (Inghilterra). La semplice interfaccia utente nasce dalle richieste da parte di organizzazioni aziendali che vogliono usare tecniche di modellazione della conoscenza, ma hanno bisogno di un ambiente e di una sintassi facili da utilizzare e da comprendere. Una schermata di Apollo é mostrata nella seguente figura. Nel pannello di sinistra, in alto, è visibile una rappresentazione gerarchica delle ontologie. La gerarchia delle classi e delle istanze di queste si trova immediatamente sotto. Una volta selezionata una classe o un'istanza, verranno mostrati i dettagli nel pannello alla destra dello schermo. Gli slot(descrittori di proprietà) e i valori di una classe o di un'istanza possono essere aggiunti utilizzando un'interfaccia in stile foglio elettronico. Apollo supporta tutte le primitive di base per la modellizzazione delle informazioni: classi, istanze, funzioni e relazioni. Un pieno controllo di consistenza è realizzato durante la fase di editing, ad esempio è in grado di riconoscere l'uso di una classe non ancora definita. Apollo possiede un proprio linguaggio per la memorizzazione delle ontologie, ma può anche esportare l'ontologia realizzata in differenti linguaggi di rappresentazione, a seconda delle scelte dell'utente. Apollo è implementato in Java.



Una schermata di Apollo

4.3.2 LinkFactory®

LinKFactory® è un sistema formale di gestione delle ontologie sviluppato dalla Language & Computing, progettato per costruire e gestire ontologie di grandi dimensioni e complessità, indipendentemente dal linguaggio usato per rappresentarle.

Il sistema di LinkFactory® si compone di due componenti principali: il Server LinKFactory®, e il LinKFactory® Workbench(banco di lavoro) (componente del livello client), entrambi sviluppati in Java. Dal lato server, LinKFactory® immagazzina i dati in un database relazionale. L'accesso al database è del tutto trasparente attraverso l'utilizzo di un gruppo di funzioni che si possono definire "naturali" quando si ha a che fare con le ontologie: *get-children*(ricava figlio), *find-path*(trova percorso), *join concepts*(unisci concetti), *get terms for concept X*(ricava termine per il concetto X), ecc. Queste funzioni sono accessibili ai client attraverso API standard che permettono di costruire applicazioni ad alto livello semantico senza la necessità di conoscere la struttura interna del database. Questo componente è in grado di operare con molti utenti in concorrenza ed è indipendente della piattaforma (testato su Windows, Solaris, UNIX e Linux).

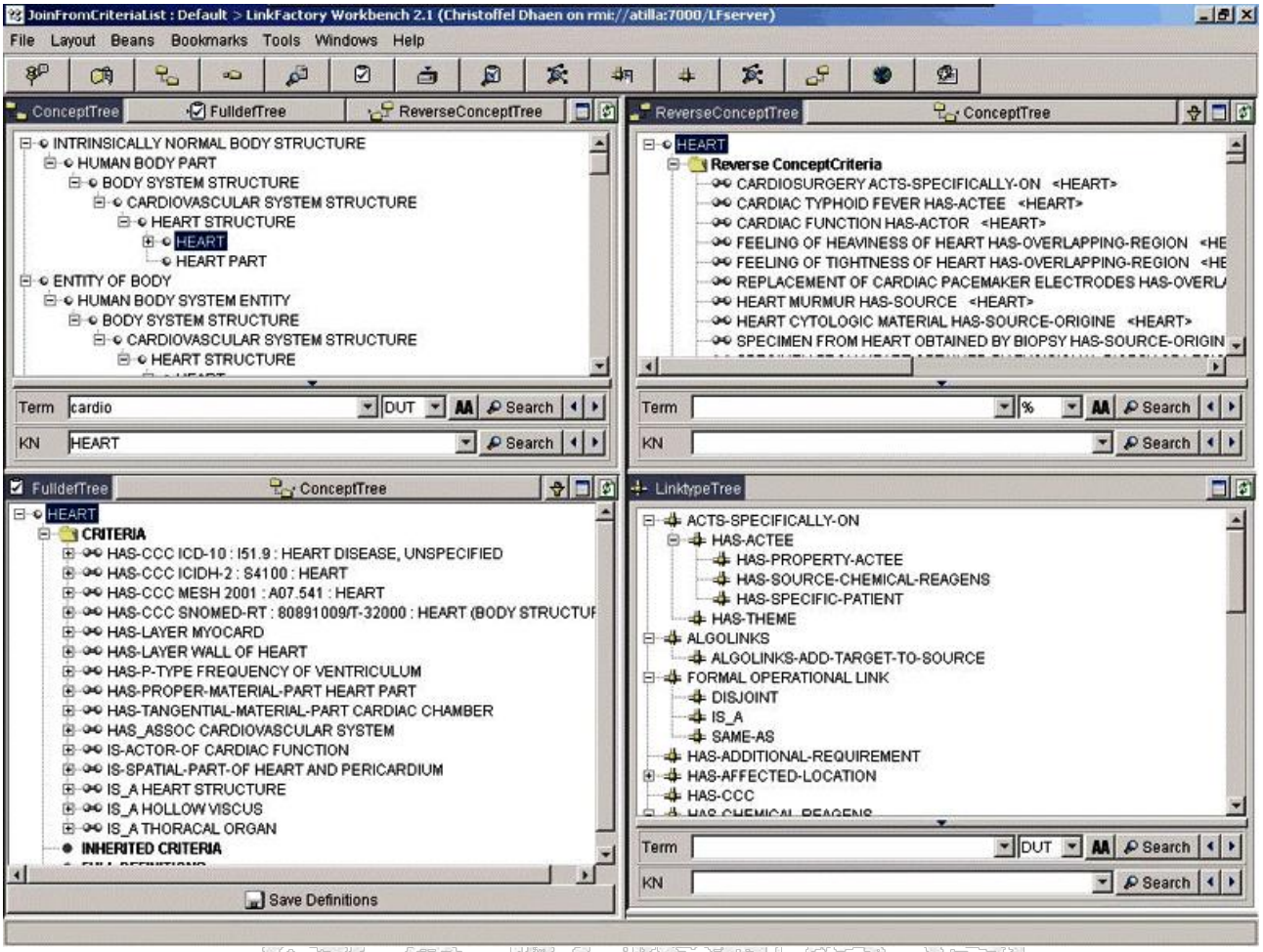
L'applicazione richiede un registro RMI (una specie di Domain Name Service per i server RMI) per poter funzionare, per essere in grado di registrarsi e di consentire ai clienti di connettersi al server RMI. Il LinkFactory® Workbench permette all'utente di sfogliare e modellare numerose ontologie e allinearle, come mostrato in figura.

Il workbench è un ambiente dinamico implementato attraverso Java beans. Ogni bean(che può essere inteso come un componente) provvede a realizzare specifiche funzionalità e ha una vista limitata sulla sottostante ontologia formale, ma combinando un gruppo di componenti bean nel workbench è possibile fornire all'utente un potente strumento per analizzare e gestire i dati. Esempi di Java beans possono essere: alberi di concetti, definizioni e verifiche di concetti, alberi LinkType, test di verifica, lista dei termini, pannelli di ricerca, pannelli di proprietà, relazioni inverse, e molti altri.

Ogni utente può creare molteplici viste utilizzando i bean disponibili. Tali viste sono dette 'Layout'. Ogni layout può essere composto di numerosi frame sui quali i bean possono essere visualizzati. Creare un nuovo layout o aggiungere nuove frame ad uno esistente sono semplici azioni che l'utente può selezionare da un menu. Ogni frame può essere suddiviso in diversi *bean -spaces*(alloggiamenti per i bean) in cui i bean possono essere piazzati. L'utente può selezionare qualunque bean disponibile e può semplicemente trascinarlo col mouse nell'area di lavoro desiderata. Una volta che l'utente ha posizionato i bean desiderati nel layout, può creare collegamenti tra questi, sempre trascinando e rilasciando il tasto sinistro del mouse. Ogni bean ha le sue specifiche proprietà, che possono essere impostate a tempo di esecuzione(*run-time properties*). Questo approccio garantisce l'esecuzione di diversi tipi di compiti adottando il miglior layout. Oltre all'accoppiamento dinamico che può essere applicato sul lato client di LinkFactory, i Java-beans possono anche essere usati al di fuori di questo workspace; in questo modo gli sviluppatori software possono integrarli come componenti (statici) dei loro programmi. Sono presenti inoltre numerosi meccanismi di controllo per la correttezza della modellazione di ontologie: controlli sulla versione, tracciamento d'utente, gerarchie d'utente, meccanismi di autorizzazione con possibilità di revoca, gerarchie tipizzate di collegamenti ecc. Dalla rappresentazione delle informazioni(della conoscenza) e da un punto di vista a basso livello, LinkFactory® ha le seguenti possibilità e caratteristiche:

- Ha già incluse le relazioni di tipo ISA(generalizzazione/specializzazione), DISJOINT(disgiunto), and SAME-AS(stesso tipo);
- Possibilità di definizione di gerarchie di relazioni (gerarchie multiple);
- Specificazione delle condizioni necessarie e sufficienti per le definizioni dei singoli concetti;
- Numerosi metodi di controllo basati su vincoli;
- Classificazione automatica di nuovi concetti a partire dal linguaggio naturale, così come da definizioni formali;

- Meccanismi per collegare e/o integrare ontologie;
- Capacità di analizzare testi in automatico e assegnare collegamenti alle ontologie (vedi figura).



L'area di lavoro di LinkFactory®

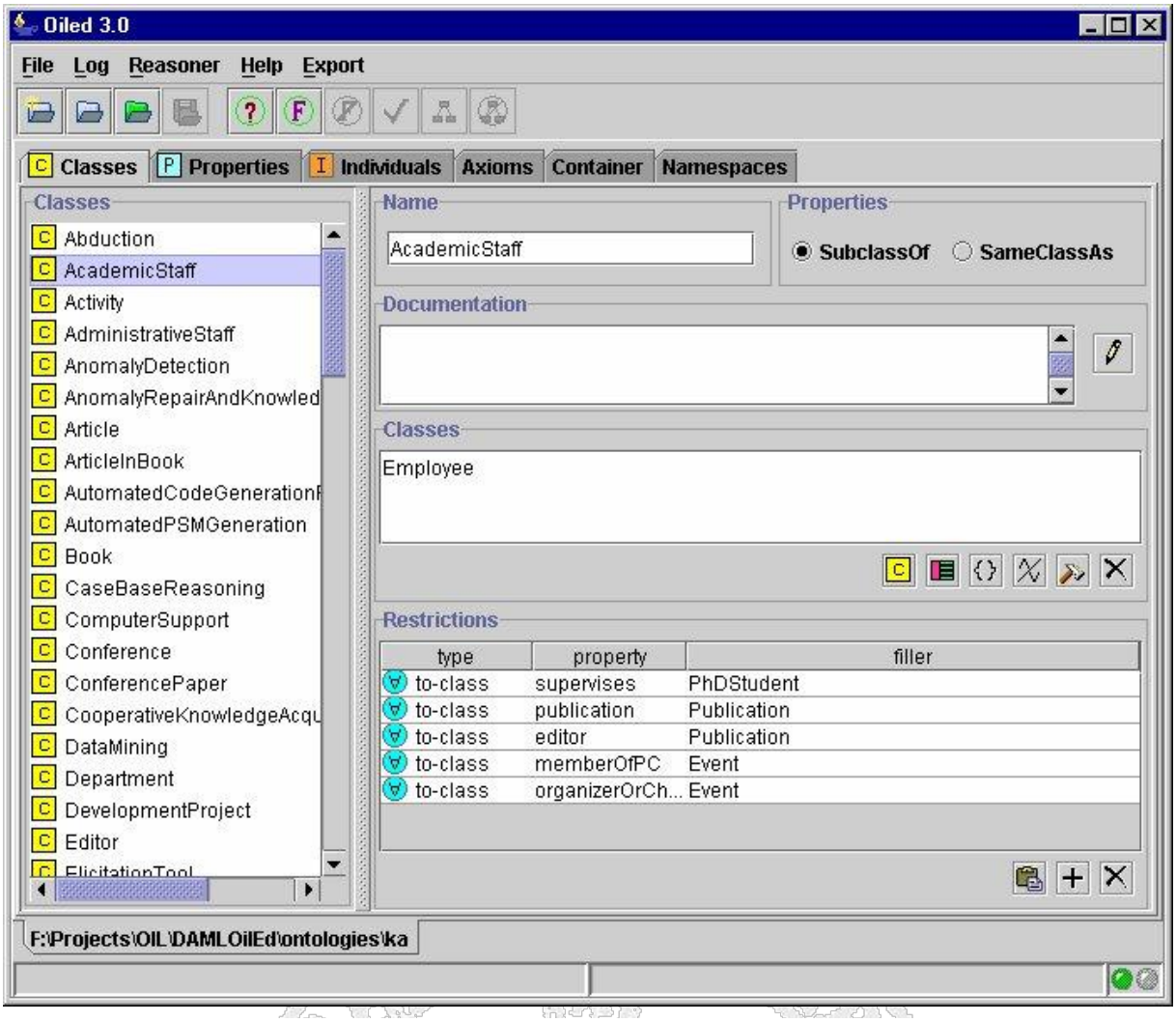
L'area di lavoro del LinkFactory® avvia la composizione di una data ontologia sulla base di un documento di testo. Il workspace presenta cinque bean che sono stati selezionati dalla barra degli strumenti. Sulla sinistra c'è il *VisualTeSSIbean* che contiene il testo originale nella metà superiore e il testo automaticamente elaborato nella metà inferiore. Il VisualTeSSIbean è connesso con un *ConceptTree* (albero dei concetti, nella parte alta sulla destra), il quale a sua volta è connesso a un *FullDefBean* (bean per una definizione

esauriente) e a un *TranslateBean* (bean di traduzione). Un clic su uno dei termini che appaiono nel testo annotato, provoca la visualizzazione delle informazioni nei bean collegati. In figura è inoltre visualizzato il LinkTypeTreeBean (bean per le gerarchie di tipi di collegamenti).

4.3.3 OILEd

OILEd è un editor grafico per ontologie, sviluppato dall'Università di Manchester che consente all'utente di costruire ontologie usando il linguaggio DAML+OIL.

Il modello di conoscenza di OILEd è basato su quello di DAML+OIL, sebbene esteso grazie all'uso di frame per la rappresentazione e la modellazione. In questo modo OILEd offre un paradigma più familiare per la rappresentazione, ma allo stesso tempo supporta la grande espressività del DAML+OIL, dove richiesto. Le *Classi* sono definite in termini delle proprie superclassi e proprietà, con la possibilità di aggiungere particolari assiomi che catturano ulteriori relazioni come quella di disgiunzione. L'espressività del linguaggio consente l'uso di descrizioni composite come quella cruciale di *ruolo*. Ciò è in contrasto con i molti editor esistenti basati su frame, dove questi tipi di "frame anonimi" devono essere etichettati prima di poter essere usati come modelli. Il compito principale per cui OILEd è stato progettato è quello dell'editing delle ontologie o di schemi, in opposizione all'acquisizione di dati e alla costruzione di numerose istanze. Sebbene sia possibile definire singole istanze, il programma è soprattutto proteso allo sviluppo di *nominals* che sono utilizzati nella costruzione dei tipi *uno-di* (one-of) di DAML+OIL.



Area di lavoro di OILed

Un aspetto chiave del funzionamento di OILEd è l'uso del FaCT reasoner [Horrocks et al, 99] per classificare le ontologie e verificarne la consistenza.

Ciò consente agli utenti di descrivere le proprie classi di ontologie e lasciare che il ragionatore determini il punto appropriato della gerarchia in cui inserirle sulla base della definizione. Nel caso in cui la definizione di un concetto non sia classificabile, appare un'icona rossa che segnala il problema.

Secondo gli sviluppatori il tool sarà in grado, in un prossimo futuro, di leggere/scrivere gerarchie di concetti in RDF puro e convertire le definizioni delle ontologie in HTML.

Questa versione di OILEd è implementata in Java ed è liberamente scaricabile solo dagli

utenti registrati. Ulteriori informazioni sono disponibili presso il sito web.

4.3.4 OntoEdit versioni Free e Professional

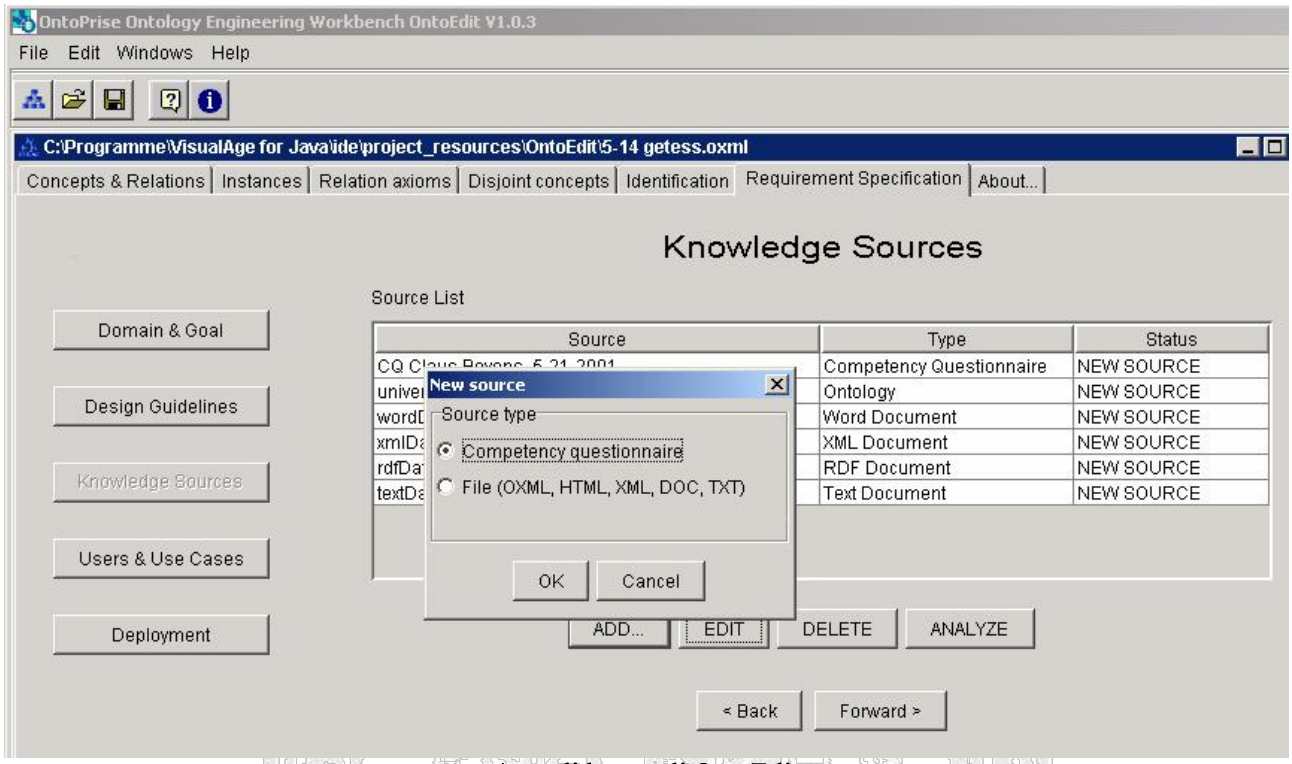
OntoEdit è un software di ingegneria delle ontologie che supporta lo sviluppo e la manutenzione di ontologie grazie all'uso di *grafici,* che visualizzano in maniera chiara i concetti. OntoEdit si fonda su un potente modello interno di ontologie. Questo paradigma supporta una modellazione neutrale al linguaggio di rappresentazione, per quanto possibile, per i concetti, le relazioni, gli assiomi. Sono presenti numerose viste grafiche che supportano le diverse fasi della modellazione delle ontologie.

Il programma fornisce agli utenti la possibilità di costruire gerarchie di concetti e classi (vedi figura). Questi concetti possono essere concreti o astratti; ciò indica se di questi concetti è possibile costruire istanze. Un concetto può avere molti nomi, che in pratica è un modo per definire sinonimi del concetto stesso. Una caratteristica da sottolineare è la possibilità di utilizzare le funzioni di "copia e incolla" per (ri)organizzare i concetti all'interno delle gerarchie. Si possono aggiungere funzionalità al tool grazie all'uso di plugin. In primo luogo, ciò semplifica l'estensibilità del software in maniera modularizzata. L'interfaccia per l'aggiunta di plugin è aperta a terze parti: gli utenti possono quindi estendere OntoEdit aggiungendo le funzionalità di cui necessitano. In secondo luogo, la presenza di un gruppo di plugin disponibili, come un dizionario di dominio, un plugin d'inferenza, e altri numerosi plugin da importare ed esportare, facilita la personalizzazione per adattare il tool ai diversi scenari di utilizzo.

Tutte le versioni di OntoEdit sono disponibili in versione free o professional. Le versioni professional, tipicamente, includono un numero più grande di plugin (vedi più avanti nel paragrafo), come ambienti collaborativi e capacità inferenziali.

La **versione Professional di OntoEdit** contiene numerosi plugin aggiuntivi rispetto alla

versione free. Tra gli altri moduli aggiuntivi, le funzionalità sono estese attraverso (1) un plugin di inferenza per effettuare controlli di consistenza, di classificazione e di regole di esecuzione, (2) una tecnica collaborativa di ontologie e (3) un server di ontologie per gestire le librerie di ontologie, la condivisione collaborativa delle ontologie e una funzione di memoria persistente per le ontologie disponibili. Le versioni Professional 2.0 e 2.5 includono i moduli (1) e la versione 3.0 punta sulle funzionalità (2) e (3).

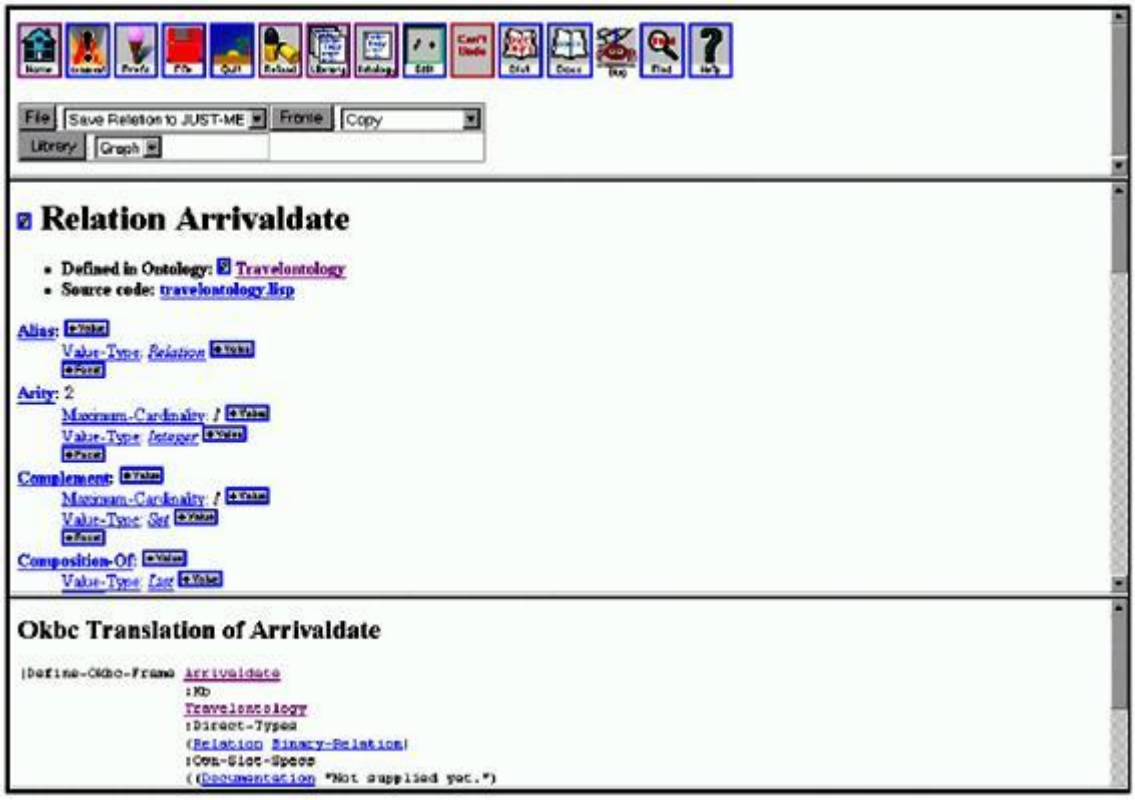


Area di lavoro di OntoEdit

4.3.5 Ontolingua Server

Ontolingua Server è un set di tool e servizi che supportano la costruzione di ontologie in maniera condivisa tra gruppi in ambiente distribuito, ed è stato sviluppato dal Knowledge Systems Laboratory (KSL) presso l'Università di Stanford. L'architettura server fornisce l'accesso a una libreria di ontologie, traduttori verso diversi linguaggi (Prolog, CORBA IDL, ecc.) e un editor per creare e sfogliare le ontologie (vedi figura). Gli editor remoti

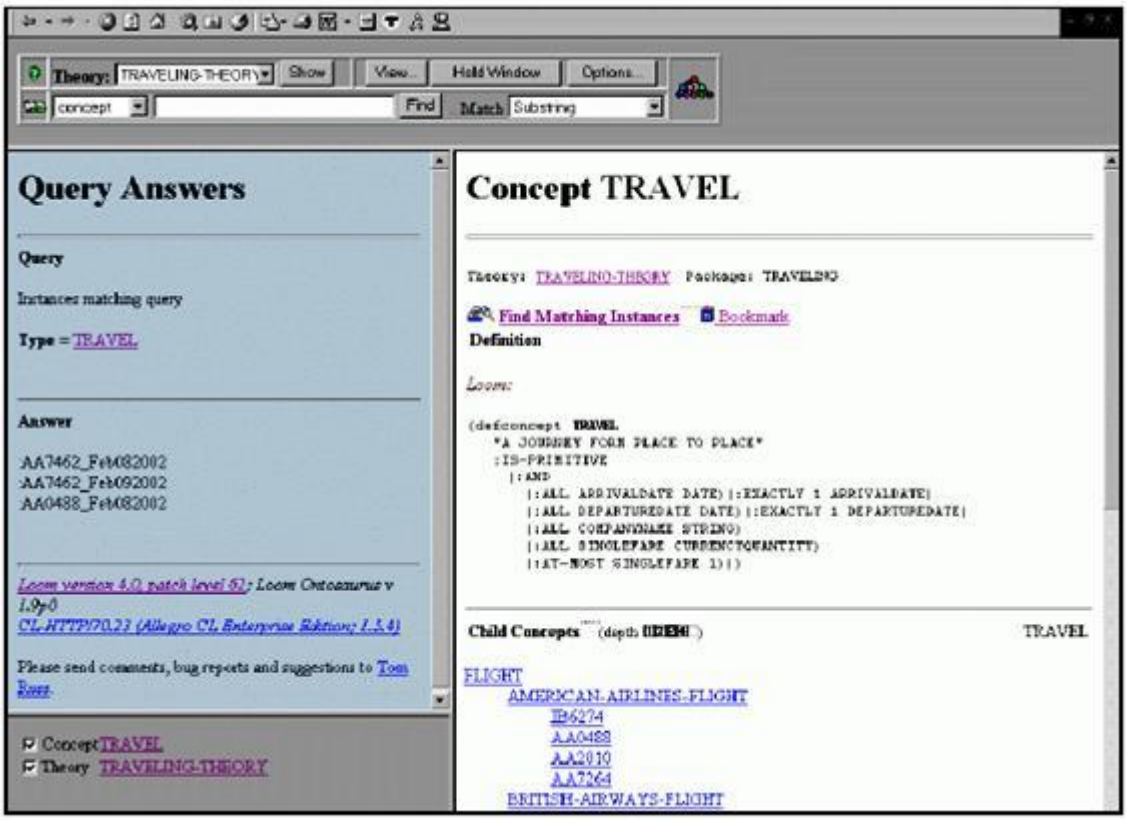
possono anch'essi sfogliare e costruire ontologie e le applicazioni locali o remote possono accedere alle librerie di ontologie usando il protocollo OKBC (Open Knowledge Based Connectivity).



Area di lavoro di Ontolingua Server

4.3.6 OntoSaurus

Ontosaurus è stato sviluppato dall'Istituto di scienze dell'informazione dell'università del sud California. Si compone di due moduli: un server di ontologie, e un server browser, che crea dinamicamente pagine HTML (includendo immagini e documentazione testuale), e che visualizza la gerarchia di ontologie(vedi figura). L'ontologia può essere modificata attraverso form HTML ed esistono traduttori verso Ontolingua e C++.



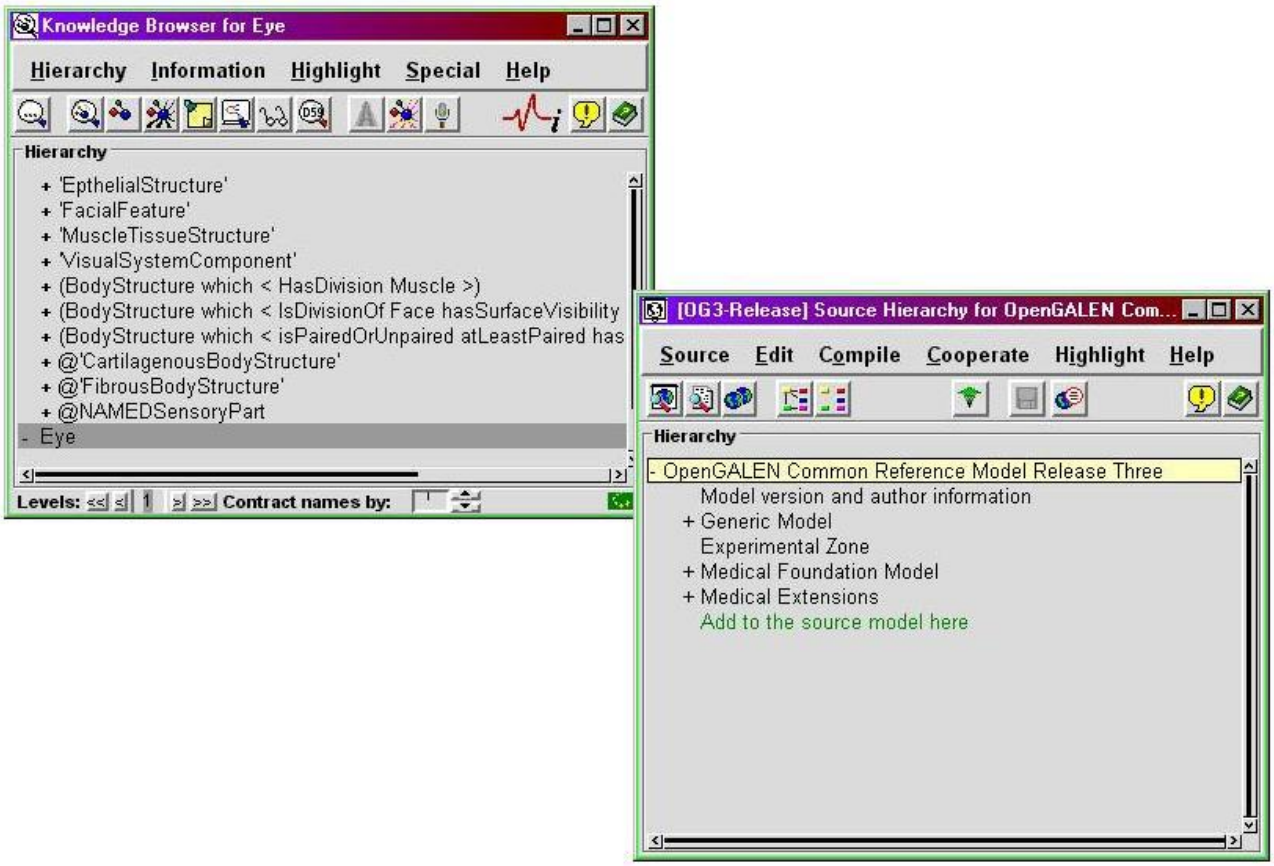
Una schermata di OntoSaurus

4.3.7 OpenKnoME

Il KnoME è una suite di tool per lo sviluppo collaborativo di ontologie basato sul linguaggio di modellazione GRAIL [Rector et al, 97]. Tigger è una parte importante di questa suite, sviluppato per la rapida acquisizione di conoscenza dagli esperti del dominio dell'applicazione non abituati all'*ingegneria ontologica*. Questi tool, così come OpenKnoMe sono disponibili gratuitamente. Sono stati prodotti dall'Università di Manchester a partire da numerosi programmi ontologici in ambito medico e farmaceutico, tra cui GALEN (1992 – 1995), GALEN-IN-USE (1996 – 1999), e PRODIGY (1999 – 2001). KnoME è utilizzato per modellare in GRAIL, un diverso linguaggio sviluppato a Manchester per l'uso dei programmi in GALEN. Si basa su grafi logici e concettuali. Il modello di rappresentazione delle informazioni di KnoME è fortemente influenzato da

GRAIL.

GRAIL è usato per modellare ontologie concettuali : KnoMe dunque, non è studiato per raccogliere istanze. Tra le caratteristiche principali di GRAIL ricordiamo:



Il centro di lancio di KnoME, knowledge browser e sanctions browser

- **Raffinamento** – la coordinazione di relazioni inclusive(all'interno di gerarchie);
- **Sanzionamento** – vincoli che descrivono come le categorie possono essere collegate attraverso attributi per costituire nuove definizioni, e in questo modo specificare ciò che deve essere comunicato per "trasferire" conoscenza;
- **Extrinsics** – collegamento di una informazione non relativa alla definizione alla struttura di ogni concetto, in modo da consentire l'indicizzazione delle informazioni specifiche di un'applicazione;

- Generazione di rappresentazione in *linguaggi naturali* di GRAIL.

KnoME non è un sistema 'stand-alone': esso comunica con un Server di Terminologia GALEN (TeS) per mezzo di specifiche API (Application programming Interface). I codici sorgenti GRAIL sono convertiti in un modello concettuale compilato. Il TeS memorizza il modello concettuale creato, utilizzando diversi moduli per fornire servizi di diverso tipo: concettuale, linguistico, di codifica e altri. Le API forniscono una netta distinzione tra l'ontologia e i clienti che la utilizzano. L'ontologia è pertanto vista come un servizio, piuttosto che come una struttura dati. Attraverso questo servizio, KnoME è in grado di visualizzare, esplorare e controllare in maniera efficace l'ontologia. Essendo l'ontologia intesa come un servizio, essa non viene solitamente esportata in "forma statica", ma piuttosto essa viene usata dai clienti che ne fanno richiesta al TeS. In ogni caso è presente un tool di esportazione pienamente configurabile che supporta la possibilità di esportare i dati in formato HTML. E' stato utilizzato principalmente, come già ricordato, nello sviluppo di progetti in ambito medico.

Esso incorpora anche una suite di tool per la gestione dei codici sorgenti: verifica, controllo, bloccaggio e separazione dei codici. Questi sorgenti vengono poi compilati per essere memorizzati.

Tigger consente di acquisire la maggior parte di conoscenza del dominio. Gli esperti di tale dominio sono abituati all'uso di una semplice *Rappresentazione Intermedia* (IR). Essi sviluppano i concetti in IR usando un tool grafico o un semplice programma per il trattamento di testi. A questo punto l'IR è coordinato, integrato e tradotto in codice GRAIL usando Tigger. Ciò supera le incomprensioni che spesso si verificano durante il processo di acquisizione della conoscenza.

OpenKnoME è implementato in VisualWorks Smalltalk della Cincom ed è liberamente disponibile dal sito internet http://www.topthing.com anche se è richiesta la registrazione di un account di posta elettronica per effettuare il download. Il download fornisce manuali,

tutorial, una versione compilata del modello medico OpenGALEN e il codice sorgente di SmallTalk.

4.3.8 Protégé-2000

Protégé-2000 è l'ultimo nato tra tutti i tool sviluppati dall'Università di Stanford per l'acquisizione delle informazioni. Protégé-2000 ha migliaia di utilizzatori in tutto il mondo per la realizzazione di progetti che coprono vari domini: dal campo medico(per modellare la diffusione del cancro) al campo militare(per la gestione delle centrali nucleari). Protégé-2000 è scaricabile gratuitamente, poiché fa parte della licenza open-source Mozilla.

Protégé-2000 fornisce un ambiente grafico e interattivo per la progettazione delle ontologie e un ambiente di sviluppo concettuale. Questo aiuta gli ingegneri e gli esperti del dominio a realizzare applicazioni per la gestione delle informazioni. Gli sviluppatori di ontologie possono accedere ad informazioni rilevanti in maniera semplice e veloce ogni volta che ne hanno bisogno, e possono usare strumenti di manipolazione diretta per navigare tra le ontologie. I comandi utilizzabili per le gerarchie (Tree controls, alberi di ontologie) consentono una veloce e semplice navigazione tra le gerarchie di classi. Protégé utilizza dei form come interfaccia per il riempimento dei valori degli slot (vedi figura). Il modello di rappresentazione delle informazioni (knowledge model) di Protégé-2000 è compatibile con OKBC. Fornisce supporto per le classi e le gerarchie di classi con molti legami di ereditarietà; offre svariati template slot pronti per l'uso; specifiche degli attributi degli slot, che includono valori consentiti, restrizioni sulla cardinalità, valori predefiniti; metaclassi (classi per gestire le classi dei domini) e gerarchie di metaclassi.

Oltre alla presenza di una semplice interfaccia, altre due caratteristihe distinguono Protégé 2000 dai molti ambienti di sviluppo per le ontologie: la **scalabilità** e l'**estendibilità.**

Gli sviluppatori hanno corredato, con successo, Protégé-2000 con oltre 150.000 frame.

Il supporto di tanti frame comporta l'utilizzo di due componenti: (1) un database sottostante per memorizzare e recuperare i dati e (2) un meccanismo di caching per consentire il caricamento di nuovi frame se il numero di questi presente in memoria, ha superato il limite consentito.

Uno dei principali vantaggi dell'architettura di Protégé-2000 è che il sistema è costituito in maniera modulare. La sua architettura basata su componenti semplifica l'aggiunta di nuove funzionalità, creando plugin appropriati. La *Protégé Plugin Library 3* contiene plugin creati da sviluppatori di tutto il mondo.

Molti dei plugin ricadono in una delle seguenti tre categorie:

backends plugins, che consentono agli utenti la memorizzazione e l'importazione delle informazioni in vari formati;

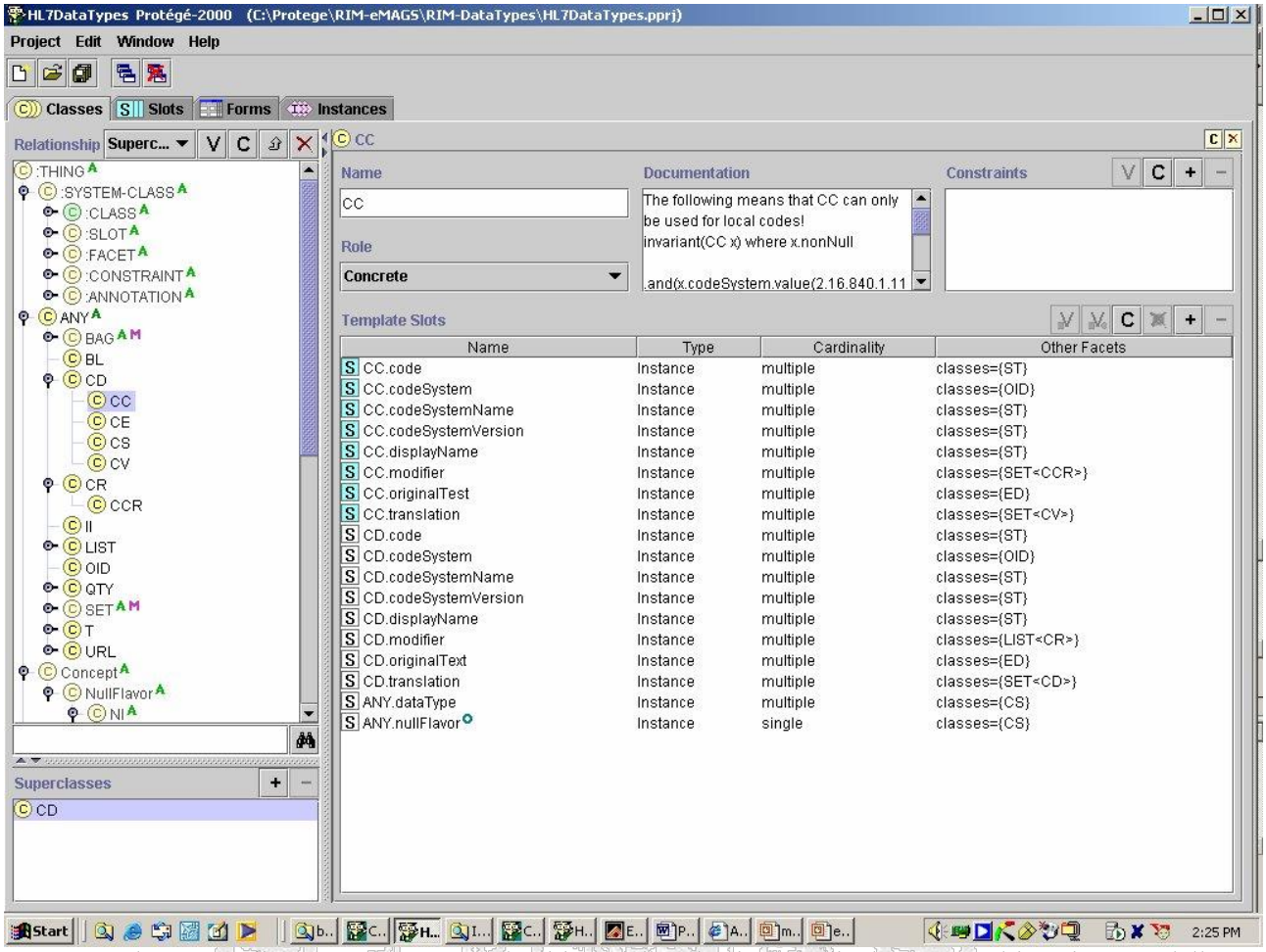
slot widgets plugins, che sono utilizzati per visualizzare e modificare i valori degli slot o le loro relazioni semantiche in domini e applicazioni specifici;

tab plugins, che sono applicazioni basate sui dati, strettamente collegate con le basi di conoscenza di Protégé.

I plugin backend attuali, (e quelli backend standard) includono supporto per la memorizzazione e l'importazione di ontologie in schemi RDF, in file XML, e in schemi XML. Il supporto per traduzione in OIL e DAML+OIL è ancora in fase di sperimentazione, ma presumibilmente sarà disponibile nelle prossime versioni.

Gli slot widgets disponibili al giorno d'oggi includono interfacce utente per visualizzare immagini GIF, video e audio. Un diagramma widget (vedi figura) consente agli sviluppatori di costruire elementi di una knowledge base tracciando un diagramma nel quale i nodi e i margini sono essi stessi dei frame particolari (che si distinguono per forma e colore). I plugin più usati sono quelli di tipo *tab* che forniscono capacità di visualizzazione avanzata, integrazione di ontologie e controllo di versione, inferenza e così via. I tab OntoViz e Jambalaya, ad esempio, presentano differenti **viste grafiche** di una

stessa rappresentazione; Jambalaya permette una navigazione interattiva della struttura, la possibilità di zoomare su particolari elementi della struttura, e differenti layout di nodi in un grafico per evidenziare le connessioni tra gruppi di dati.

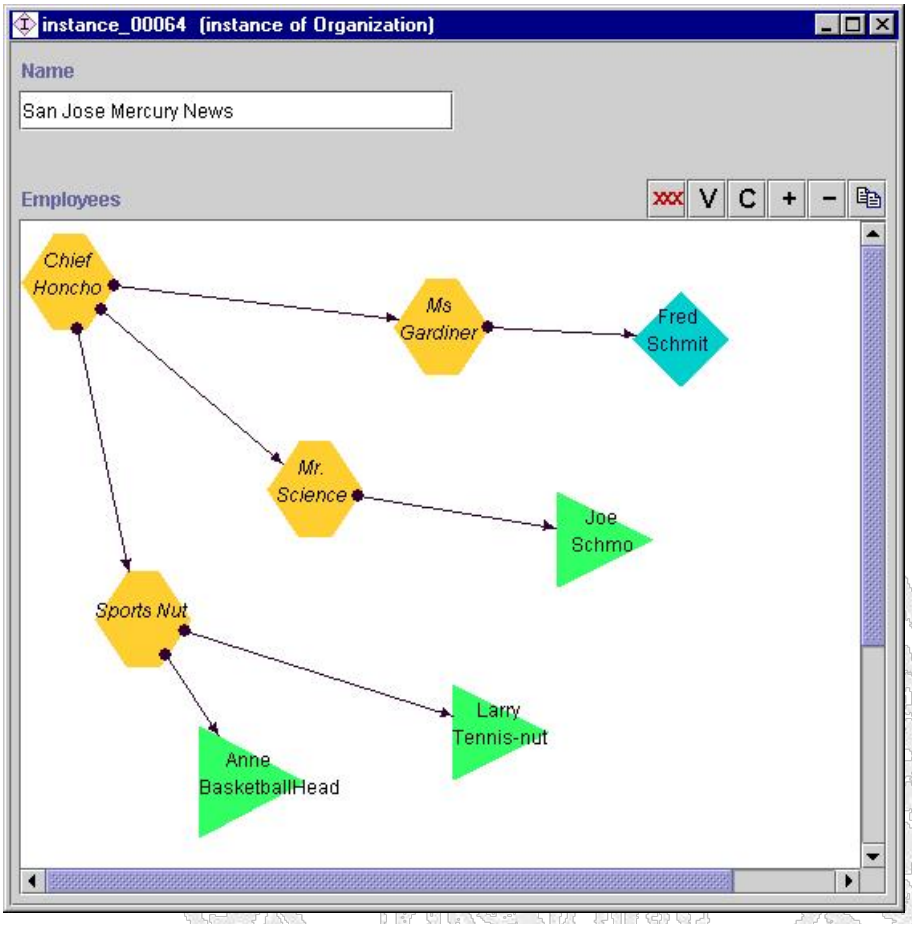


Una schermata di Protégé-2000 per l'editing delle classi e degli slot e per l'inserimento delle informazioni delle istanze. La gerarchia di classi con legami di ereditarietà è mostrata nel pannello a sinistra. Gli utenti possono riorganizzare la gerarchia con il semplice metodo del drag and drop, trascinando le classi con il mouse. Il pannello di destra mostra informazioni dettagliate per la classe selezionata. Esso include anche gli slot che descrivono le istanza della classe.

Il tab PAL (plugin) fornisce il supporto per il linguaggio di assiomi per Protégé (*Protégé Axiom Language*). PAL permette agli utenti di aggiungere **vincoli** sui dati per i quali il formalismo del frame stesso non è sufficientemente espressivo. Inoltre il motore di inferenza PAL analizza i dati per indicare agli utilizzatori quali vincoli hanno violato e in che modo. Il tab Flora e il tab Jess forniscono accesso a "motori di ragionamento"

sviluppati da terze parti. Il tab PROMPT fornisce un ambiente per **gestire ontologie multiple**. Sono presenti tool per la fusione di ontologie che aiutano l'utilizzatore a trovare similitudini tra le ontologie; per il controllo di versione delle ontologie, i quali automaticamente individuano differenze strutturali tra versioni diverse di un'ontologia; tool per l'estrapolazione di sottoparti semanticamente complete di un'ontologia e il riordino dei frame in ontologie differentemente collegate.

I tab UMLS e WordNet consentono agli utenti di importare e integrare anche elementi presenti on-line nelle proprie ontologie.



Un diagramma widget in Protégé. Gli utenti possono trascinare gli elementi presenti in una lista, per creare un diagramma che mostri le relazioni tra le istanze di un'ontologia.

4.3.9 SymOntoX

SymOntoX (Symbolic Ontology Manager XML savvy), è un prototipo software per la gestione di ontologie di dominio, che forniscono gli oggetti specifici di una particolare applicazione. E' stato sviluppato dal LEKS (Laboratory for Enterprise Knowledge and Systems), nell'ambito del CNR-IASI. In SymOntoX, I concetti del dominio e le relazioni tra essi, sono modellati in accordo all'OPAL (Object, Process, and Actor modelling Language), un modello di rappresentazione ideato dallo stesso laboratorio di ricerca. SymOntoX, successore di SymOntos, è al giorno d'oggi sviluppato e sperimentato nel contesto del progetto europeo *Harmonise*, che riguarda il settore del turismo (perciò, saranno forniti esempi concernenti questo dominio applicativo).

In accordo al linguaggio OPAL, i concetti sono suddivisi in tre tipi di modelli:

Attori, Processi, e Oggetti. Più precisamente, si ha:

- *Attore* – una qualunque entità rilevante del dominio che può attivare o compiere un processo (ad esempio, un *Turista, un'Agenzia di Viaggi*);
- *Oggetto* – un'entità passiva sulla quale opera un processo (ad esempio, un *Hotel,* un *Volo aereo*);
- *Processo* – un'attività mirata all'assolvimento di un obiettivo (ad esempio, *Effettuare_una_prenotazione*).

Oltre ai tipi precedenti l'OPAL propone i seguenti tipi di modelli di tipo complementare:

- Componente di informazione – un gruppo di informazioni riguardanti un Attore o un Oggetto. (ad esempio, Le informazioni di un volo aereo, l'indirizzo di un

Hotel);

- *Elemento di informazione* – un elemento *atomico* di informazione che fa parte di un componente di informazione (ad esempio, *il prezzo di un volo aereo*, *il numero di camere di un hotel*);

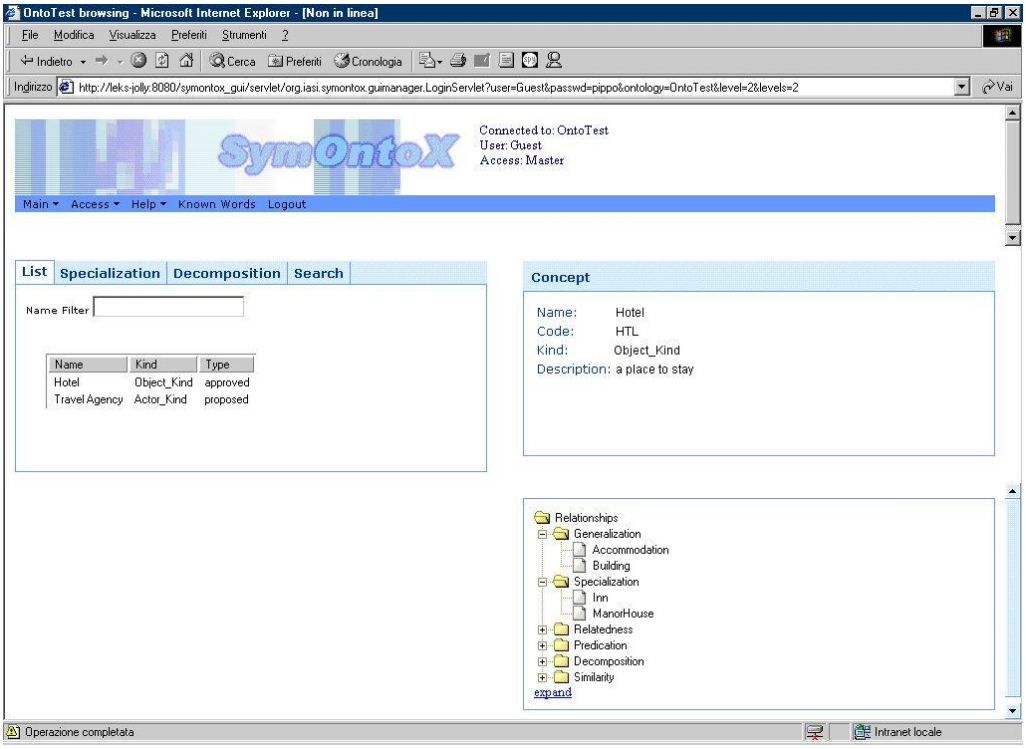
- *Azione* – un'attività che *rappresenta* una componente di un processo, che può essere decomposta (ad esempio, *la richiesta di una camera in un albergo*);

- *Azione elementare* – un'attività che *rappresenta* una componente di un processo che non può essere più decomposta (ad esempio, *La cancellazione di una prenotazione*).

- *Goal* – è uno stato che rappresenta un obiettivo che un attore cerca di raggiungere(ad esempio, *Andare in vacanza*);

- *Stato* – è un modello tipico di valori che le istanze di una entità possono assumere (ad esempio, *volo completo*, che può significare che tutti i posti sono stati assegnati);

- *Regola* – è un'espressione che mira a restringere i possibili valori di un'istanza di un concetto (vincolo) o che consente di ricavare nuove informazioni (regola di elaborazione). (ad esempio, *L'acquisto di un biglietto trenta giorni prima della partenza*);



Una schermata di SymOntoX

I modelli presentati sono necessari per la definizione (univoca) dei concetti. In accordo all'OPAL, I concetti sono tra loro collegati attraverso tipiche relazioni:

- Specializzazione;
- Decomposizione;
- Asserzione;
- Analogia;
- Relazione.

SymOntoX è stato pensato per essere un servizio fruibile attraverso Internet tramite un comune browser di navigazione web.

E' principalmente basato su XML (tutti i dati sono memorizzati in un database XML) e costruito su tecnologia Java, per garantire la massima flessibilità, interoperabilità,

portabilità e indipendenza dalla piattaforma. Da sottolineare è anche la sua architettura a tre livelli.

SymOntoX è in grado di gestire differenti ontologie, diversi tipi di utilizzatori e differenti modalità d'uso. Un utente può essere registrato come **User** (con permessi di sola lettura), come **SuperUser** (col permesso di aggiungere nuovi concetti, ma solo come *proposte,* cioè solo come idee) oppure come **Ontology Master** (il responsabile dell'ontologia. Può anche avere l'incarico di accettare o rifiutare le proposte dei SuperUser).

Il sistema può essere usato come un **glossario, (**sono mostrati solo il nome e la descrizione in linguaggio naturale dei concetti), come un **thesaurus** (sono mostrati anche le gerarchie di specializzazione e le relazioni di analogia/similitudine), come un **ontology system** (sono visualizzate tutte le relazioni), oppure come una **knowledge base** (che aggiunge ai precedenti anche il concetto di istanza).

SymOntoX presenta un'interfaccia utente grafica per l'editing e la visualizzazione e una visualizzazione a diagrammi per sfogliare il contenuto dell'ontologia. Inoltre un set di API Java fornisce l'interoperabilità con altri sistemi. Inoltre, un *validatore di ontologie* ne assicura la consistenza rispetto al modello OPAL. Tutti i dati (ontologie, istanze dei concetti, e informazioni di riepilogo) sono memorizzati in un database XML.

4.3.10 WebODE

WebODE è uno strumento per l'ingegneria ontologica che fornisce vari servizi relativi alle ontologie e fornisce supporto per molte delle attività necessarie al processo di sviluppo e utilizzo delle ontologie. Presenta un'architettura simile ad un application server, che garantisce una alta estendibilità e usabilità, consentendo una facile aggiunta di nuovi servizi e l'utilizzo di altri già esistenti. Le ontologie, in WebODE sono rappresentate utilizzando un modello molto espressivo, basato sulla metodologia METHONTOLOGY, che modella l'ontologia con componenti come i concetti (con istanze, e attributi di classe),

partizioni, relazioni binarie specifiche, relazioni predefinite (tassonomie, parte di), istanze, assiomi, regole, costanti e riferimenti bibliografici. Inoltre consente l'importazione di termini da altre ontologie; le ontologie in WebODE sono memorizzate in un database relazionale. In più WebODE fornisce una serie di API ben definite e orientate ai servizi per l'accesso alle ontologie che rendono semplice l'integrazione con altri sistemi. Le ontologie realizzate con WebODE sono facilmente integrabili con altri sistemi e ciò si ottiene utilizzando la funzione automatica di esportazione/importazione di servizi da e in XML.

Vi sono anche servizi per la traduzione delle ontologie in specifici linguaggi ontologici (RDF(S), OIL, DAML+OIL) e strumenti per la traduzione di servizi in altri linguaggi e sistemi come Java e Jess. L'editing delle ontologie è assistito da un'interfaccia utente basata su form, da un gestore di viste definite dall'utente, da un verificatore di consistenza, da un motore di inferenza, da un costruttore di assiomi e dalla documentazione. Due interessanti caratteristiche di WebODE che lo differenziano dagli altri prodotti sono: gruppi di istanze(*istance sets*) che consente di istanziare lo stesso modello concettuale per diversi scenari, e viste concettuali (*conceptual views*) che consente di creare e memorizzare differenti parti dell'ontologia, evidenziando e/o personalizzando la visualizzazione dell'ontologia per ciascun utente.

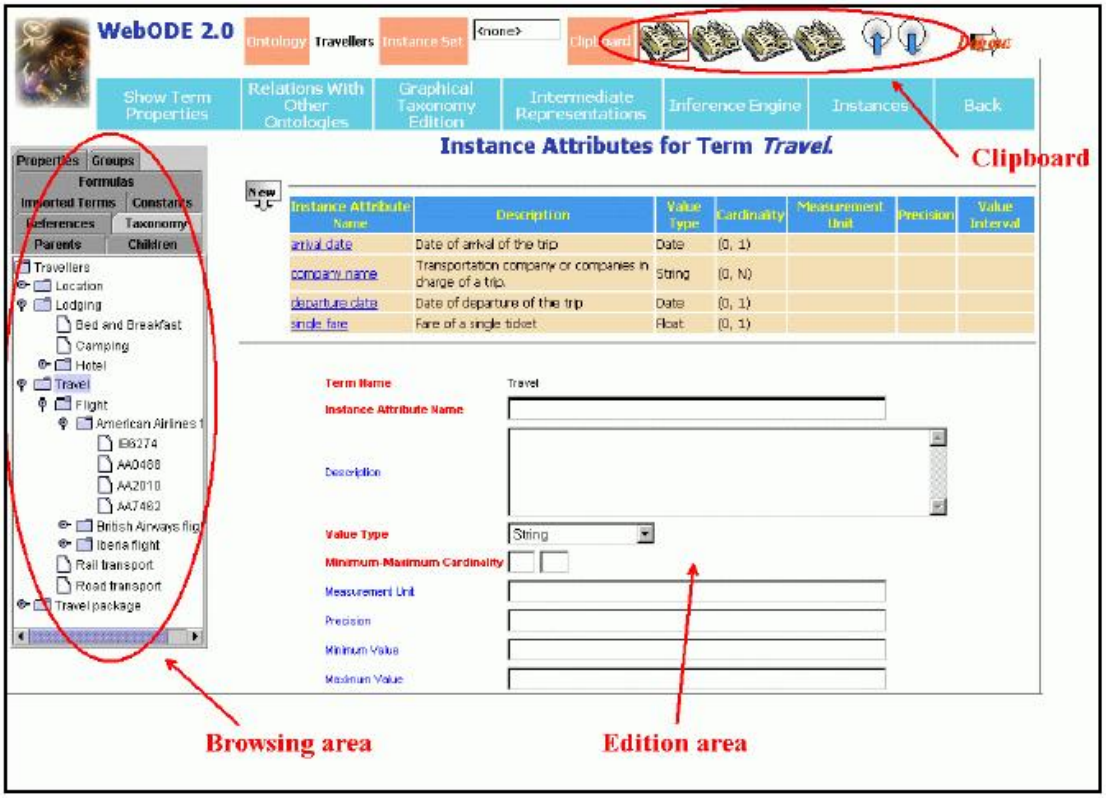
L'interfaccia grafica consente di passare in rassegna tutte le relazioni definite sull'ontologia, ma anche di visualizzare solo relazioni di un certo tipo.

Proprietà matematiche quali la riflessiva, la simmetrica, e altre proprietà definite dall'utente possono essere aggiunte alle relazioni "ad-hoc".

La collaborazione è assicurata da un meccanismo che permette agli utenti di stabilire il tipo di accesso alle ontologie sviluppate (sola lettura, aggiunta), attraverso la nozione di gruppo di utenti. Esistono anche meccanismi di sincronizzazione che permettono a numerosi utilizzatori di editare la medesima ontologia senza errori. Sono inoltre presenti capacità di controllo, di vincoli per i tipi di dato, vincoli sul numero di valori attribuibili,

sulla cardinalità e di verifica di consistenza delle tassonomie (ad esempio, istanze comuni di classi disgiunte, cicli, ecc.).

Infine, il servizio di inferenza di WebODE è stato sviluppato in Prolog (un sottoinsieme di primitive OKBC sono state definite in Prolog per il loro uso in questo motore inferenziale). Per di più l'*Axiom Builder* di WebODE trasforma gli assiomi logici e le regole iniziali in Prolog (cfr. Appendice A), se possible, cosicché essi possano essere usati al meglio.

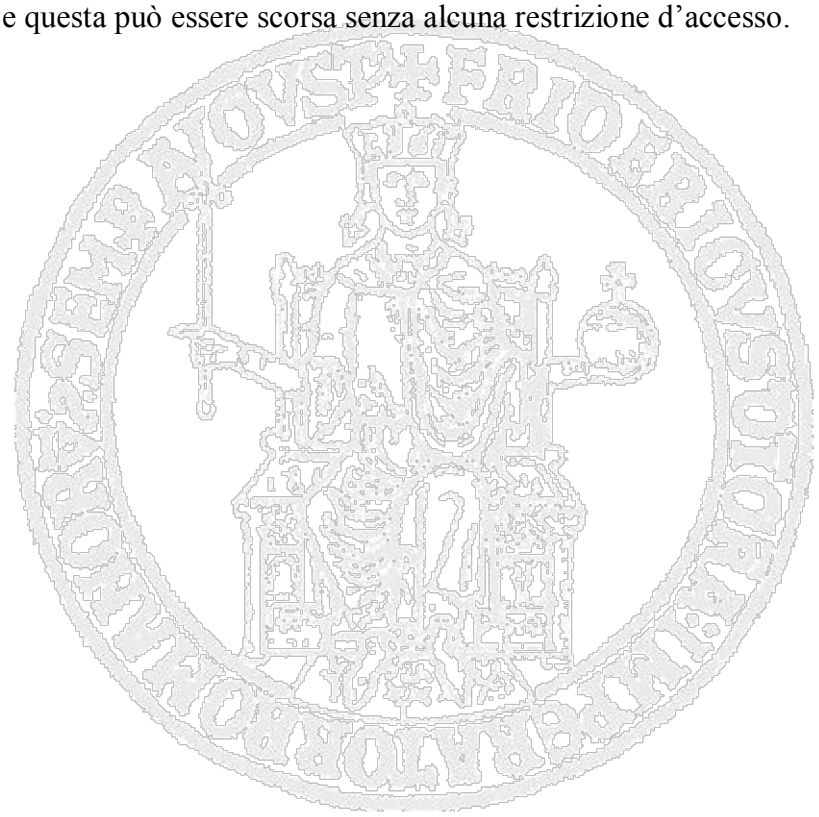


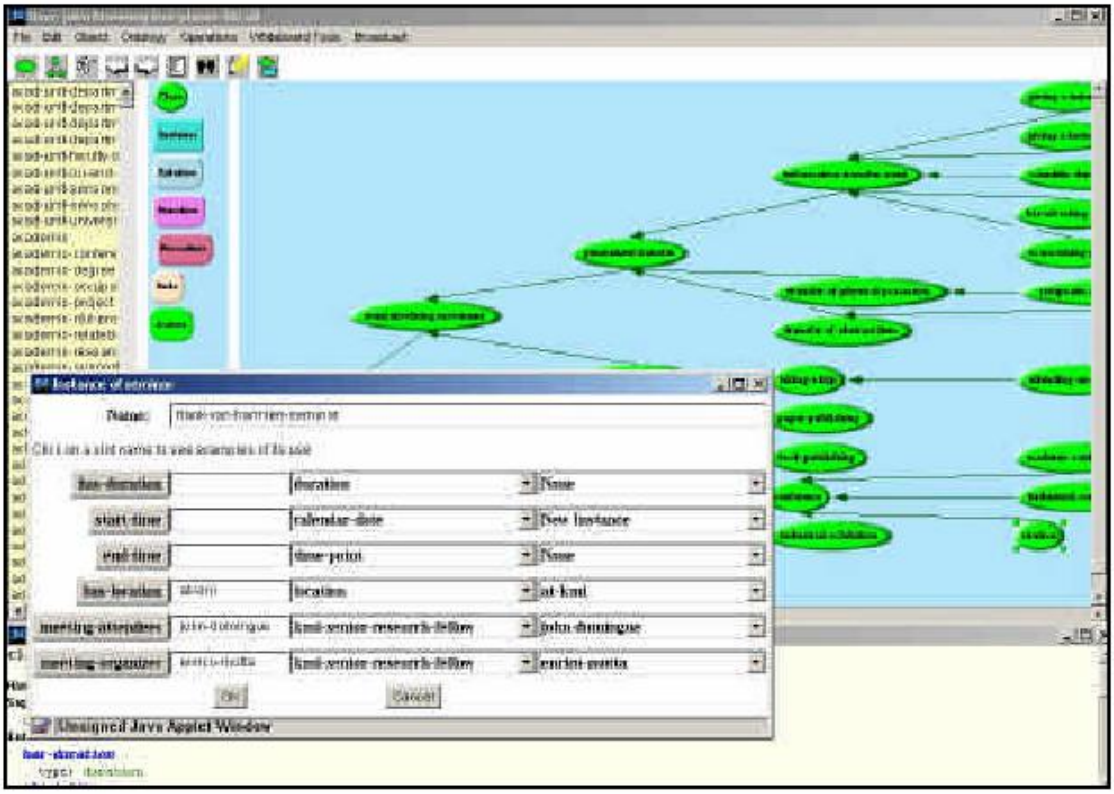
Una schermata di WebODE

4.3.11 WebOnto

WebOnto è un tool sviluppato dal Knowledge Media Institute (KMi) della Open University (Inghilterra). Supporta il browsing, la creazione e l'editing collaborativi di ontologie. Le sue principali caratteristiche sono: la gestione di ontologie attraverso una interfaccia grafica (vedi figura); la generazione automatica di form per l'editing delle istanze a partire dalle definizioni di classe, il supporto per la modellazione di attività; il controllo di elementi (tenendo conto del fatto che le proprietà sono ereditate) e il controllo di consistenza; una completa interfaccia *tell&ask* (informare e chiedere) di documentazione, e un supporto per la collaborazione, attraverso l'utilizzo di annotazioni di tipo broadcast(che vengono cioè inviate a tutti i collaboratori).

WebOnto server è un servizio liberamente fruibile, fornito alla comunità di ingegneri delle ontologie. Attraverso WebOnto è possibile accedere ad una libreria di oltre cento ontologie e questa può essere scorsa senza alcuna restrizione d'accesso.





Una schermata di WebOnto

4.3.12 OntoMaker

OntoMaker è un tool di sviluppo di ontologie progettato in Italia presso l'Università di Milano adatto sia agli esperti del dominio, sia agli utenti che non ne hanno conoscenza. I requisiti principali che il prodotto vuole soddisfare sono il supporto al ciclo di vita iterativo delle ontologie e la collaborazione con altri tool di sviluppo. Un'altra caratteristica, che lo rende diverso dagli altri tool esistenti, è la possibilità di sviluppare un'ontologia a partire da un template (cioè ontologie già definite, modelli preesistenti pronti da usare); OntoMaker fornisce un modello per guidare gli utenti nella creazione dell'ontologia seguendo le tipiche regole di buona progettazione.

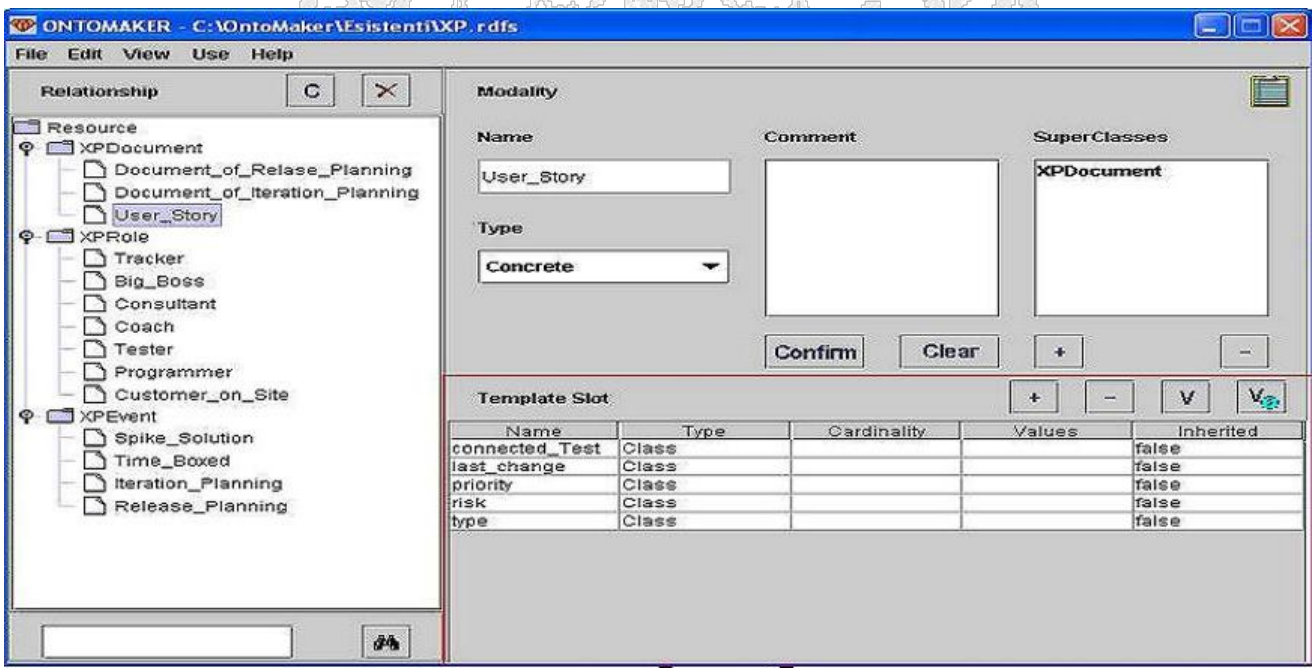
Il ciclo di vita iterativo si basa sulla concezione che le ontologie possono essere definite da gruppi indipendenti. Un'ontologia già usata per l'accesso alle risorse e alle informazioni può modificare la sua struttura, pertanto se un concetto cambia, anche le asserzioni su di

esso devono essere modificate. Invece, un'ontologia può essere utilizzata come un *indice,* cosicché quando il contenuto cambia, bisogna solo garantire l'integrità delle istanze che essa contiene. L'interfaccia grafica e le funzionalità sono simili a quelle di Protégé-2000.

Ciò che distingue davvero OntoMaker dagli altri tool è la possibilità di creare "istanze reali" e non di definirle come *Stringhe* di una classe. Questo è possibile grazie alla presenza di un database per la memorizzazione delle informazioni; il programma fornisce un modo semplice per descrivere e immagazzinare la struttura delle ontologie. Il database è composto da 15 tabelle di cui 8 sono utilizzate per rappresentare le ontologie, 5 per conservare le informazioni generate quando le ontologie sono usate come indici e le ultime 2 per la manutenzione delle informazioni. Nelle prossime versioni si prevede l'utilizzo di MySQL e di Oracle per la realizzazione del database, che per il momento è costruito avvalendosi di Microsoft Access.

Slot

Gli slot rappresentano gli attributi di una classe o le relazioni con altre classi dell'ontologia. La figura seguente mostra come per ogni slot vi siano facets (proprietà degli slot) predefinite.



Una schermata di OntoMaker. I template slots

Uno slot rappresenta un'entità, pertanto non può avere duplicati. A tale scopo sono previsti controlli sull'integrità nominale. Il controllo è effettuato sulla base del nome e del tipo. Gli utenti possono ottenere una lista degli slot definiti, la quale consente di visualizzarli o di aggiungerne di nuovi.

OntoMaker è in grado di importare ed esportare ontologie espresse in linguaggio RDF o RDF(S); in questo modo può operare in maniera collaborativa con altri tool che utilizzano questo tipo di linguaggio.

4.3.13 Confronto e valutazione dei tool

In questa sezione saranno forniti commenti sui tool presentati in precedenza. In ordine alfabetico: Apollo, LinkFactory, OpenKnoME, OILEd, OntoMaker, OntoEdit Free Version, OntoEdit Professional Version, Ontolingua, Ontosaurus, Protégé2000, SymOntoX, WebODE e WebOnto.

Un aspetto importante da analizzare in un tool è quello della sua architettura e della sua evoluzione. Per architettura si intende se il tool è standalone, di tipo client/server, oppure se è ha una struttura a strati. Altri parametri da valutare sono di certo l'estendibilità, il tipo di memorizzazione delle ontologie (database, file ASCII, ecc.) e la presenza di sistemi di backup.

Molti tool si stanno orientando verso piattaforme Java e verso architetture estendibili.

La memorizzazione su database resta ancora un punto debole, dato che solo pochi di questi tool utilizzano basi di dati per immagazzinare ontologie: LinkFactory, OntoEdit versione Professional, OntoMaker, Protégé2000 e WebODE.

Stesso dicasi per le funzionalità di backup che sono fornite solo da OpenKnoME, SymOntoX, WebODE e WebOnto.

Un altro fattore da considerare è l'interoperabilità con altri tool per lo sviluppo ed il merging delle ontologie, la capacità di interagire con altri sistemi di informazione e

database; la possibilità di tradurre da e verso altri linguaggi per la costruzione di ontologie è un'altra caratteristica rilevante allo scopo di integrare le ontologie nelle applicazioni.

Molti dei nuovi tool importano ed esportano in formato HTML e altri linguaggi specifici basati su tag. In ogni caso, a tutt'oggi, non esiste uno studio comparativo sulla qualità di questi traduttori. Inoltre non ci sono prove che lo scambio di ontologie tra tool diversi e i processi di traduzione da un linguaggio ad un altro non possa comportare una perdita di informazioni.

Dal punto di vista del paradigma di rappresentazione delle informazioni, si possono individuare due famiglie di tool: tool basati su una descrizione logica, quali OILEd, OntoSaurus e OpenKnoMe; e altri tool, che invece consentono di rappresentare le informazioni seguendo un approccio ibrido basato su frame. Oltre a ciò, Protégé-2000 offre un metodo di rappresentazione molto flessibile che fa uso di componenti come le *metaclassi*(cioè classi per definire le classi dell'ontologia).

Per quanto riguarda le metodologie supportate, entrambe le versioni di OntoEdit supportano la metodologia OntoKnowledge, OpenKnoMe supporta la metodologia GALEN, SymOntoX supporta invece quella OPAL, infine WebODE supporta la METHONTOLOGY. In ogni caso, nessuno dei tool analizzati include applicazioni per la gestione dei progetti creati e servizi per la manutenzione delle ontologie, ma solo un esiguo supporto per la valutazione.

Prima di scegliere un tool, è anche importante conoscere quali servizi di inferenza esso è in grado di offrire. Ci si può concentrare sui motori di inferenza che sono disponibili, sui meccanismi di controllo dei vincoli e della consistenza, sulle capacità di classificazione automatica e gestione delle eccezioni. LinkFactory ha un motore di inferenza proprietario, WebODE e Ontosaurus forniscono servizi per la valutazione delle ontologie. LinkFactory, OILed, OntoSaurus e OpenKnome sono i soli a realizzare classificazioni automatiche perché basati su linguaggi di descrizione logica.

Da sottolineare il fatto che nessuno dei tool analizzati fornisce meccanismi di gestione

delle eccezioni.

Per quanto riguarda l'usabilità, WebOnto possiede le caratteristiche più avanzate relative alla costruzione di ontologie in maniera cooperativa. In generale un numero maggiore di caratteristiche sono richieste ai tool esistenti per assicurare una costruzione collaborativa *di successo* di ontologie. Inoltre l'usabilità si può valutare anche in base alla documentazione d'uso fornita a corredo, al tipo di visualizzazione ecc., caratteristiche che di certo saranno migliorate nelle prossime versioni.

4.3.14 Considerazioni generali

Riassumendo, esistono molti tool "simili" per lo sviluppo di ontologie, ma nessuno di essi è in grado di operare con tutti gli altri, né di coprire tutte le attività che caratterizzano il ciclo di vita delle ontologie (solo progettazione e implementazione).

La mancanza di interoperabilità tra tutti questi tool genera gravi problemi quando si tenta di integrare un'ontologia nella libreria di ontologie di un altro tool, oppure quando si vuole integrare o fondere il contenuto di ontologie realizzate con tool e/o linguaggi differenti.

Di conseguenza, la tendenza del futuro dovrebbe essere diretta alla creazione di un' area di lavoro comune per lo sviluppo delle ontologie che semplifichi: l'intero ciclo di vita di sviluppo delle ontologie, la creazione di tecniche avanzate per la gestione delle ontologie e per visualizzarne il contenuto, ecc.

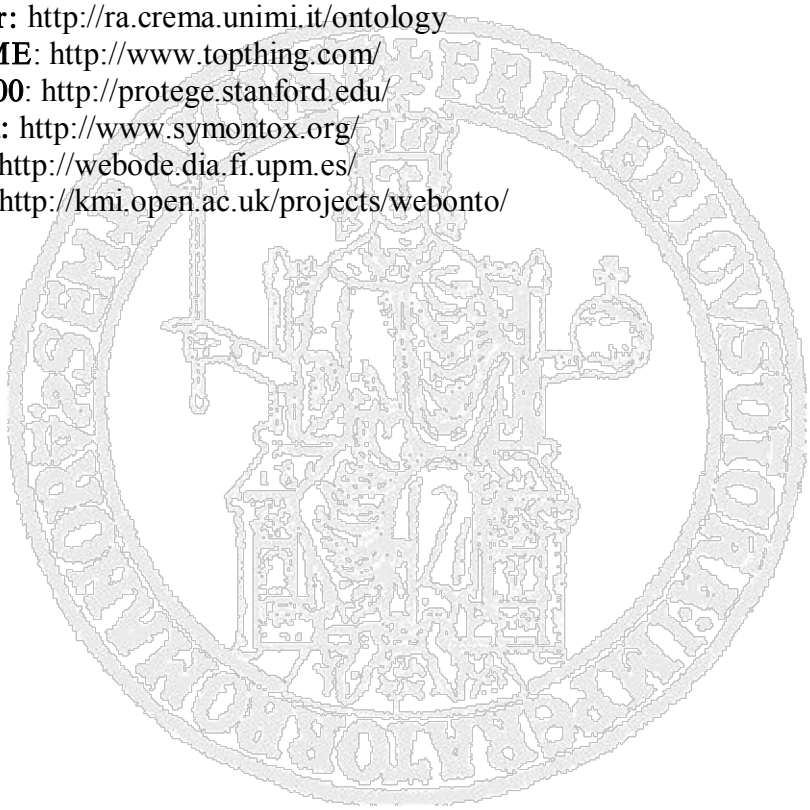
Questa area di lavoro comune dovrebbe anche prevedere un gruppo di servizi middleware per le ontologie che supportino l'uso di ontologie in altri tool. Alcuni di questi servizi possono essere: software che aiutino a individuare l'ontologia più appropriata per una data applicazione, metriche formali che valutino le analogie semantiche e le differenze tra termini della stessa o di differenti ontologie, software che consentano aggiornamenti incrementali, consistenti e selettivi di un'ontologia, accesso remoto ad un insieme di librerie, software che facilitino l'integrazione dell'ontologia con sistemi e database legacy,

ecc.

Infine, un largo uso di questa tecnologia nelle società, con il conseguente sviluppo di un gran numero di applicazioni basate sulle ontologie nel contesto del web semantico, sarà raggiunto grazie alla creazione di suite per lo sviluppo di applicazioni che fanno uso di ontologie, che consentiranno il rapido sviluppo e l'integrazione delle applicazioni esistenti e future su una architettura basata su componenti. La creazione di aree comuni porterebbe anche un vantaggio nella qualità delle ontologie che sarebbero analizzate e validate da più esperti del settore.

4.3.15 URL

Apollo: http://apollo.open.ac.uk/
LinkFactory: http://www.landc.be/
OntoEdit Free and Professional: http://www.ontoprise.de/com/start_downlo.htm
OILEd: http://oiled.man.ac.uk/
Ontolingua: http://www-ksl.stanford.edu/
Ontosaurus: http://www.isi.edu/isd/ontosaurus.html
OntoMaker: http://ra.crema.unimi.it/ontology
OpenKnoME: http://www.topthing.com/
Protégé 2000: http://protege.stanford.edu/
SymOntoX: http://www.symontox.org/
WebODE: http://webode.dia.fi.upm.es/
WebOnto: http://kmi.open.ac.uk/projects/webonto/



Feature	Apollo	Link Factory	OILEd	OntoEdit Free	OntoEdit Pro	Onto Lingua	Onto Saurus	Open KnoME	Protégé	SymOntoX	WebODE	WebOnto	OntoMaker
Architettura Sw	Standalone	3-tier	Standalone	Standalone	Standalone & client/server	Client/ Server	Client/ Server	Client/Server	Standalone	3-tier	3-tier	Client/Server	Standalone
Extensibility	Plugin	Sì	No	Plugin	Plugin	No	No	No	Plugin	No	Plugin	No	Plugin
Memorizzazione	File	DBMS	File	File	File e DBMS	File	File	File	File e DBMS(JDBC)	XML	DBMS(JDBC)	File	DBMS
Backup	No	No	No	No	No	No	No	Sì (log)	No	Sì	Sì	Sì	Sì

Architettura dei tool

Feature	Apollo	Link Factory	OILEd	OntoEdit Free	OntoEdit Pro	Onto Lingua	Onto Saurus	Open KnoME	Protégé	Sym OntoX	WebODE	WebOnto	OntoMaker
Importa i linguaggi	Metalinguaggio Apollo	XML RDF(S) DAML+OIL	RDF(S) OIL DAML+OIL	XML RDF Flogic DAML+OIL	XML RDF Flogic DAML+OIL	Ontolingua IDL	IDL Onto C++	GRAIL GALEN	XML RDF(S) XML Schema	---	XML RDF(S)	---	XML RDF(S) XML Schema
Esporta nei linguaggi	----	XML RDF(S) DAML+OIL HTML	XML RDF(S) DAML+OIL HTML	XML RDF(S) Flogic DAML+OIL	XML RDF(S) Flogic DAML+OIL	IDL OKBC syntax PROLOG syntax	IDL ONTO C++	GRAIL HTML GALEN	XML, RDF(S), XML Schema, FLogic, Java HTML	RDF(S)	XML, RDF(S) OIL DAML+OIL FLogic Prolog Java HTML	Ontolingua RDF(S) OIL	XML, RDF(S), XML Schema, Java HTML

Interoperabilità

Feature	Apollo	Link Factory	OILEd	OntoEdit Free	OntoEdit Pro	Onto Lingua	Onto Saurus	Open KnoME	Protégé	Sym OntoX	WebODE	WebOnto	OntoMaker
Modello di rappresentazione	Frame (OKBC)	Frame	DAML+OIL	Frame	Frame	Frame	---	GRAIL	Frame e Metaclassi.	OPAL	Frame	Frame	Frame
Supporto alla metodologia	No	Sì	No	Sì (Onto-Knowledge)	Sì (Onto-Knowledge)	No	No	Sì (GALEN)	No	Sì (OPAL)	Sì	Sì (Methontology)	No
Linguaggio assiomatico	---	proprietario	Sì (DAML+OIL)	Sì (Flogic)	Sì (Flogic)	Sì	Sì	Sì (GRAIL)	Sì	OPAL	Sì	Sì	No

Metodo di rappresentazione e supporto alla metodologia

Feature	Apollo	Link Factory	OILEd	OntoEdit Free	OntoEdit Pro	Onto Lingua	Onto Saurus	Open KnoME	Protégé	Sym OntoX	WebODE	WebOnto	OntoMaker
Motore di inferenza built-in	No	Sì	Sì	No	Sì (OntoBroker)	No	Sì	Sì	Sì	Sì	Sì	Sì (Prolog)	No
Controllo di consistenza	Sì	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	Sì	Sì	Sì	Sì
Classificazione autometica	No	Sì	Sì	No	No	No	Sì	Sì	No	No	No	No	No
Gestione delle eccezioni	No	No	No	No	No	No	No	No	No	No	No	No	No

Servizi di inferenza

Feature	Apollo	Link Factory	OILEd	OntoEdit Free	OntoEdit Pro	Onto Lingua	Onto Saurus	Open KnoME	Protégé	Sym OntoX	WebODE	WebOnto	OntoMaker
Tassonomia grafica	Si	Sì	No	No	No	Sì	No	No	Sì	Sì	Sì	Sì	Sì
Potature dei grafi(viste)	Sì	Sì	No	No	No	No	No	No	Sì	No	No	No	Sì
Zoom	No	Sì	No	No	No	No	No	No	Sì	No	No	No	Sì
Collaborazione	No	Sì	No	No	Sì	Sì	Sì	Sì	No	Sì	Sì	Sì	No
Librerie di Ontologie	Sì	Sì	Sì	No	Sì	Sì	No	Sì	Sì	Sì	No	Sì	Sì

Usabilità dei tool

4.4 Tool per l'integrazione e la fusione di ontologie

4.4.1 Introduzione

La fusione di ontologie è diventato un punto chiave negli ultimi anni. Da un lato, la fusione delle ontologie durante la fase di progettazione è molto importante, poiché la fusione di due società o organizzazioni comporta spesso la fusione delle proprie ontologie. A volte si ricorre alla fusione di numerose ontologie per ottenerne una di migliore qualità. D'altra parte, anche il merging a tempo di esecuzione può essere cruciale. Infatti, le ontologie sono diventate estremamente comuni nel World-Wide-Web dove forniscono semantica alle annotazioni presenti nelle pagine di internet. Inoltre, l'eterogeneità delle informazioni sul web possono portare a lavorare con differenti ontologie in domini molto simili. Queste diversità devono coesistere con le interazioni tra i sistemi. Una scelta possibile per rendere compatibili le diversità è quella di stabilire regole di mappatura tra diverse ontologie e riunirle tutte a tempo di esecuzione (Mena et al.; 2001).

Come conseguenza delle motivazioni appena espresse, sono nati numerosi tool per effettuare il merging delle ontologie.

Si presenteranno, come prima cosa, i criteri principali usati per comparare differenti tool per il merging. Successivamente saranno presentati brevemente alcuni tool. Questi saranno poi oggetto di una valutazione comparativa che porterà alla conclusione dell'analisi.

4.4.2 Principi di valutazione usati per comparare i tool

Questa sezione presenta un set di criteri che possono essere utilizzati per comparare I tool per il merging di ontologie.

Le caratteristiche principali da valutare sono:

- **Descrizione generale**. Questa descrizione è importante per conoscere il contesto del tool. Questa include il nome, gli sviluppatori, e la politica dei prezzi(open source, shareware, license, ecc.).

- **Architettura ed evoluzione**. E' importante conoscere quali sono i requisiti tecnici, e come gli utilizzatori hanno accesso alle nuove versioni e aggiornamenti. Sono incluse le seguenti caratteristiche:

  - Il tool di merging è integrato in un tool per lo sviluppo di ontologie? Quale?
  - Le piattaforme Hardware e Software. I punti di riferimento sono: il sistema operativo, il tipo di host(PC, Macintosh, Sun, ecc.)
  - Il tool deve essere installato presso l'utilizzatore?

- **Tolleranza ai guasti.** I possibili valori sono bassa, media, alta. Per giudicare questa caratteristica, bisogna tener conto delle seguenti considerazioni:

  1) Questo criterio entrerà in gioco solo in caso di comparazioni.
  2) I valori attribuiti dovranno essere giustificati da malfunzionamenti riscontrati.

- ***Sistema di gestione di backup?*** Se il tool di merging è integrato in un altro tool, un valore per questo criterio potrebbe essere: "si, eseguito dal tool host"
- ***Stabilità.*** I possibili valori sono: bassa, media, alta. Valgono le stesse considerazioni fatte per la tolleranza ai guasti.

- ***Efficienza***. Possibili valori: bassa, media, alta. Valgono le stesse considerazioni fatte per la tolleranza ai guasti.
- ***Politica di aggiornamento del tool***. Vale a dire, qual è la periodicità d'uscita di una nuova versione del tool?, quanto costa l'aggiornamento? In che modo vengono avvisati i clienti della presenza di nuove versioni?

- ***Informazioni usate durante il processo di fusione*** (quali informazioni usa il tool per i suggerimenti e le fusioni automatizzate). Il processo di merging può essere molto più efficiente se sono disponibili informazioni addizionali come alcune risorse linguistiche. Le caratteristiche da considerare sono le seguenti:

  - *Dizionari elettronici, dizionari dei sinonimi, ecc*. I valori possibili sono: "Si, ma non è necessario per il processo", "si, ed è necessario per il processo", o "no".

  - *Vocabolari*. I valori possibili sono: "Si, ma non è necessario per il processo", "si, ed è necessario per il processo", o "no".

  - *Definizione di concetti e valori degli slot*. I possibili valori sono: "Si, ma non è necessario per il processo", "si, ed è necessario per il processo", o "no".

  - *Struttura a grafi*. I valori possibili sono: "Si, ma non è necessario per il processo", "si, ed è necessario per il processo", o "no".

  - *Istanze di concetti*. I valori possibili sono: "Si, ma non è necessario per il processo", "si, ed è necessario per il processo", oppure "no".
  - *Input dall'utente*. I valori possibili sono: "Si, ma non è necessario per il processo", "si, ed è necessario per il processo", o "no".

- ***Interoperabilità.*** Questa caratteristica è importante perché le operazioni chiave possono essere eseguite da altri tool, che non consentono il merging: cambiamento di formati, valutazione, ecc. Le principali domande da porsi sono le seguenti:

  - E' consentita l'interoperabilità con altri tool o altri sistemi di informazione? Ad esempio, il tool può recuperare ontologie da un tool di sviluppo remoto?

  - Il tool è in grado di fondere ontologie espresse in linguaggi diversi? Quali?

  - *Modalità di esecuzione.* E' importante valutare il ruolo dell'utente nel processo di fusione. La caratteristica da considerare è la seguente:

  - *In che modo lavora il tool?* Completamente interattivo, completamente automatico, oppure automatico con una successiva approvazione da parte dell'utilizzatore.

- ***Gestione di differenti versioni di ontologie***. Un cambiamento nello schema iniziale di un'ontologia, cambia il risultato della fusione. Le caratteristiche da considerare sono:

  - *Il tool si avvantaggia della fusione di versioni precedenti delle ontologie.* Ad esempio, se l'ontologia O, è la fusione di O1-v.1 e O2-v.1 e sviluppiamo O1-v.2: è necessario ripetere completamente il processo di fusione di O1 e O2, o ci si può basare sul risultato precedente?

  - *Avvertimenti sui cambiamenti delle ontologie di partenza?* Il sistema è in grado di avvertirci che O non è il risultato della fusione delle versioni aggiornate di O1 e O2?

- ***Componenti che il tool può integrare.*** Vale a dire, quali parti delle ontologie possono essere fuse? Quali vengono perse? Ciò è molto importante. I componenti da considerare

sono:

- *Concetti,* considerando i propri slot, i modelli di slot, le tassonomie, i concetti stessi, le partizioni, le relazioni e funzioni, e il numero totale di entità coinvolte in una particolare relazione.

- *Assiomi*. Si possono integrare gli assiomi delle ontologie da fondere? e le regole?

- *Istanze*. Istanze di concetti, istanze di relazioni.

❖ **Suggerimenti forniti dal tool** (il tool fornisce suggerimenti per realizzare la fusione?). Quando il processo di fusione è interattivo, il sistema può suggerire all'utente quali componenti possono essere miscelati al prossimo passo. I componenti da considerare sono:

- *Concetti,* considerando i propri slot, i modelli di slot, le tassonomie, i concetti stessi, le partizioni, le relazioni e funzioni, e il numero totale di entità coinvolte un una particolare relazione.

- *Gruppi di assiomi*. Il sistema può garantire che un gruppo di assiomi può essere integrato.

- *Gruppi di regole*. Il sistema può garantire che un gruppo di regole può essere integrato.

- *Istanze*. Istanze di concetti, istanze di relazioni

- ***Conflitti rilevati dal tool***. Questa caratteristica analizza se il sistema può rilevare nomi globali ripetuti, oppure strutture ridondanti nell'ontologia risultante.

- ***Supporto di tecniche e metodologie***. E' importante sapere se il tool possiede un supporto metodologico, oppure no; è anche importante sapere quali tecniche sono supportate dal tool. In particolare ci si chiede:

  - *Il tool supporta qualche metodologia?* ("No" oppure "Si" e il nome della metodologia)

  - *Altre tecniche:* Statistica (Sì/No), capacità di apprendimento(Sì/No).

- ***Sistema di supporto***. Questo è un punto cruciale di ogni tool, soprattutto per i meno esperti. Gli elementi da tenere in conto sono:

  - Documentazione fornita a corredo.

  - Tutorial sulle metodologie. Ha senso se il tool supporta metodologie

  - Guida attraverso l'interfaccia. Guida generale, tutorial, esercitazioni guidate, eccetera.

  - Guida specifica.

- ***Editazione e visualizzazione***. Molto importante per migliorare l'usabilità del tool.

  - Visualizzazione step-by-step del processo. (grafiche, non grafiche, nessuna).

- Visualizzazione simultanea delle ontologie da fondere.

- Visualizzazione di parti delle ontologie da fondere (viste).

- Zoom.

❖ *Esperienza nell'uso del tool.* Un aspetto importante è quello di avere un'idea delle capacità del tool

- Domini.

- Progetti in cui il sistema di merging è stato utilizzato.

- Applicazioni in cui le ontologie unificate sono state utilizzate.

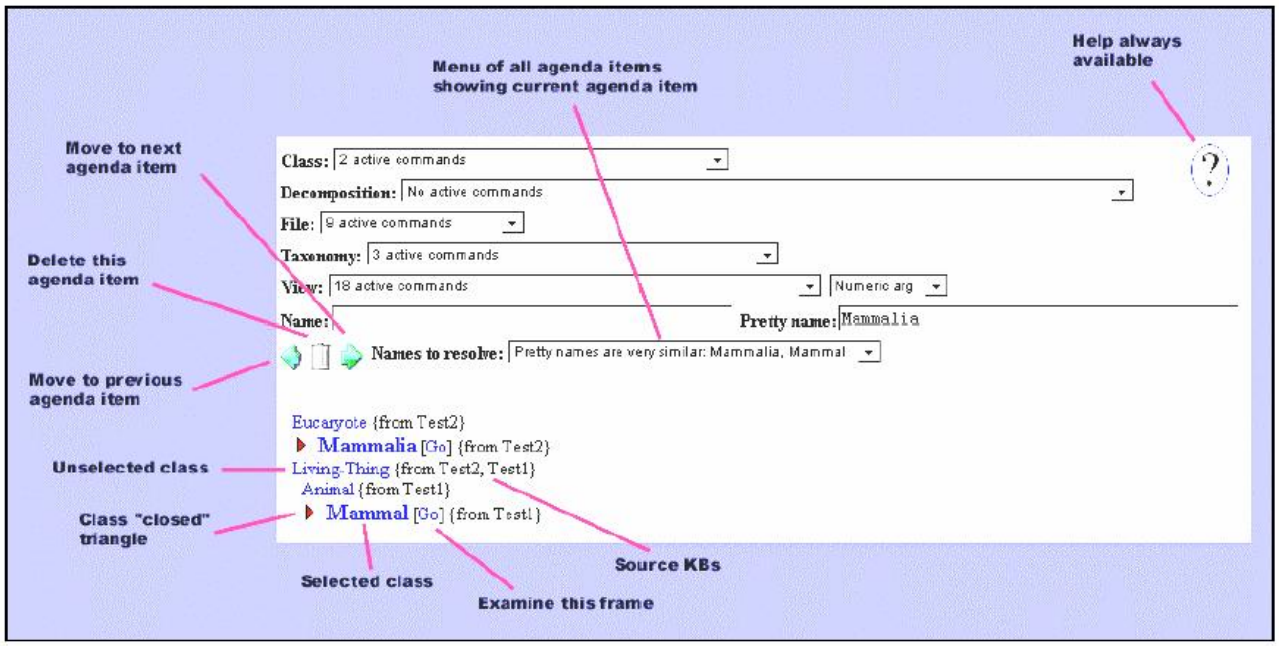
4.4.3 Chimaera

Chimaera è un ambiente per la fusione di ontologie e per la navigazione tra queste, orientato al web. Chimaera accetta oltre quindici tipi di formati di input (quali Ontolingua, Protégé, ecc.). Nelle prossime versioni sarà compatibile anche con standard emergenti quali l'RDF e DAML.

Chimaera possiede un semplice editor e permette anche di utilizzare l'editor e il browser di Ontolingua per una editazione facilitata. Ontolingua non è comunque un requisito per l'utilizzo; al suo posto possono essere usati altri editor. Chimera facilita il merging consentendo all'utente di trasferire (*upload*) ontologie esistenti in un nuovo workspace (oppure in un'ontologia esistente). La figura mostra il risultato del trasferimento di due ontologie (Test1 and Test2) e la modalità di risoluzione dei nomi (*name resolution mode*). Chimaera suggerisce i candidati di una possibile fusione sulla base di alcune proprietà. Il

tool genera una lista di risoluzione dei nomi che può essere usata come guida per il processo di fusione. L'opzione visualizzata in tale lista, in figura, suggerisce di fondere "Mammal" e "Mammalia" (poiché hanno nomi simili). Viene visualizzata la partizione di gerarchia coinvolta e l'utente può sfogliare la gerarchia in dettaglio espandendo le sottoclassi (attraverso un clic sul triangolo al fianco del nome della classe). L'utente può anche vedere le definizioni dei termini; con Ontolingua, si possono anche ottenere le analogie e le differenze strutturali tra questi. A questo punto l'utente può scegliere di fondere i termini scegliendo la voce corrispondente nel menu delle classi.

Le impostazioni di Chimaera consentono anche di definire *il livello di intensità dei suggerimenti* relativi al merging dei candidati. Chimaera è fornito anche di una modalità di risoluzione delle tassonomie (*taxonomy resolution mode*) che ricerca relazioni sintattiche fra i termini. Quando si dispone di un classificatore, ricerca anche le relazioni semantiche di inclusione.



La modalità di risoluzione dei nomi di Chimæra che suggerisce di fondere Mammal e Mammalia.

Chimaera possiede anche capacità di analisi che consentono agli utenti di eseguire una

serie di test con la possibilità di scegliere se eseguirli tutti o meno. Il risultato è un log interattivo che visualizza l'esito di test effettuati. Sono disponibili test d'incompletezza, controlli sintattici, analisi di tassonomie, e controlli semantici. Il log riporta ad esempio termini che sono stati usati, ma che non sono stati definiti, termini che hanno intervalli di valori contraddittori, l'esistenza di cicli.

Chimaera ha trovato uso nel progetto High Performance Knowledge Base per l'analisi delle ontologie del sistema. E' stato utilizzato/valutato anche da società quali VerticalNet e Cisco. L'utilizzo del tool è sottoposto ad una licenza del produttore.

4.4.4 PROMPT

PROMPT [Noy and Musen, 2000] è un tool semi-automatico per il merging delle ontologie. In realtà esso è un plugin per Protégé-2000. PROMPT guida l'utente durante il processo di merging delle ontologie, identificando i possibili *punti di integrazione*, e fornendo suggerimenti sulle prossime operazioni che dovrebbero essere eseguite, quali conflitti devono essere risolti, e come fare per risolverli. Il processo di merging delle ontologie di PROMPT è interattivo. Spetta all'utente la maggior parte delle scelte, e sulla base di queste, PROMPT esegue automaticamente compiti addizionali sulle ontologie.

Per la ricerca dei conflitti e per dare suggerimenti il tool tiene conto di alcune caratteristiche delle ontologie iniziali. Alcune tra queste sono:

- Nomi delle classi e degli slot (ad esempio, se i frame hanno nomi simili e dello stesso tipo, essi sono buoni candidati per il merging).

- Gerarchia di classi (ad esempio, se l'utente fonde due classi e PROMPT "si accorge" che le loro superclassi sono simili, allora "sarà più sicuro" di consigliare all'utente di effettuare merging anche delle superclassi, poiché queste giocano lo stesso ruolo rispetto alle classi che l'utente ha fuso).

- Collegamento degli slot alle classi (ad esempio, se due slot di differenti ontologie sono riuniti in una classe intergrata, e i loro nomi, il loro aspetto (inteso come insieme di frame), e i loro valori sono simili, allora questi slot sono candidati per il merging).

- Aspetto e valori degli slot (ad esempio, se un utente fonde due slot, allora gli intervalli di restrizione sono candidati per il merging).

Oltre a fornire suggerimenti, PROMPT identifica i conflitti. Alcuni dei conflitti che PROMPT identifica sono di seguito riportati:

- Conflitti sui nomi (più di un frame con lo stesso nome);

- Riferimenti pendenti (un frame riferisce un altro frame che non esiste);

- Ridondanze nella gerarchia delle classi (presenza di più di un percorso da una classe a una "classe parente" oltre alla root);

- Restrizioni sui valori degli slot che violano l'ereditarietà tra classi.

Le caratteristiche su elencate utilizzano la struttura a grafi delle ontologie in un ambito limitato: si attraversano i nodi che sono soltanto a uno o due gradi (della gerarchia) di separazione. AnchorPROMPT [Noy and Musen, 2001] è un'estensione di PROMPT che compara la struttura a grafi su larga scala. Accetta come input un gruppo di *anchors*, coppie di termini collegati, definiti dall'utente o identificati automaticamente da un confronto lessicale. Anchor-PROMPT tratta un'ontologia come se fosse un grafo in cui le classi rappresentano i nodi e gli slot rappresentano i collegamenti (archi). L'algoritmo

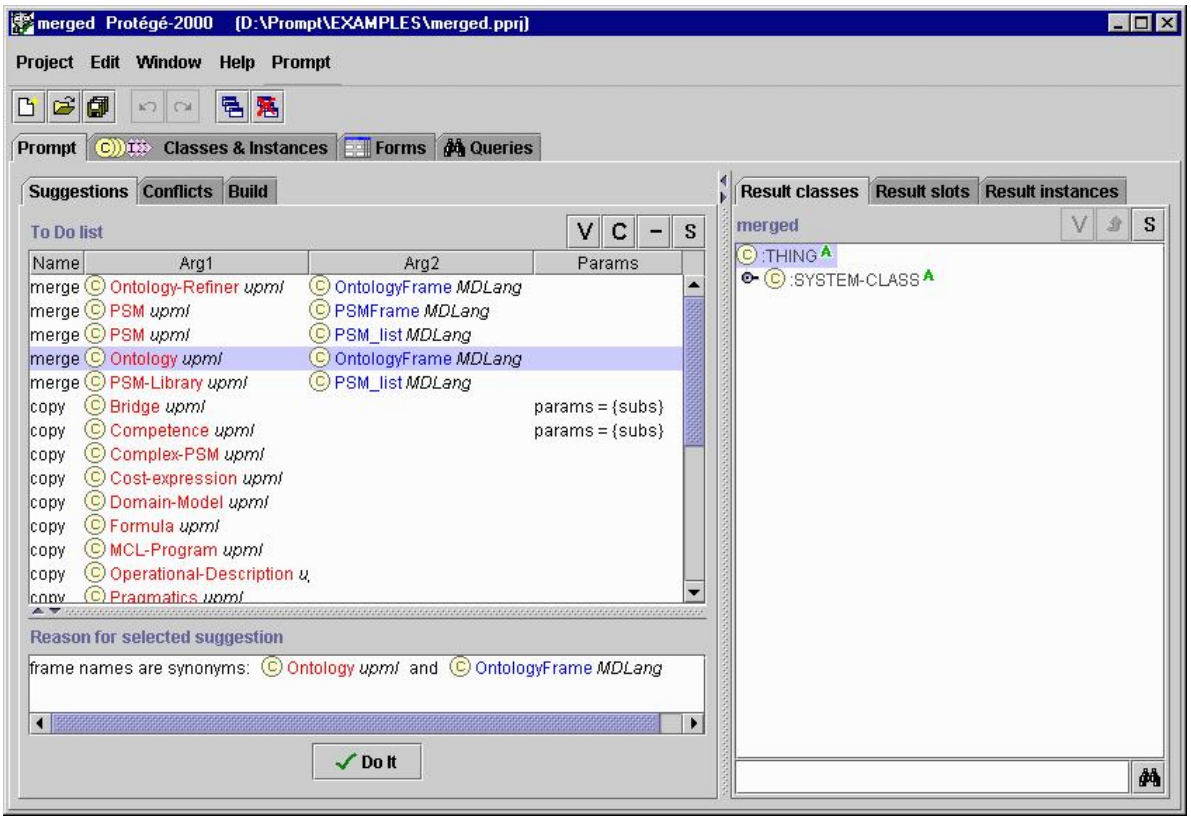
analizza i percorsi nel sottografo limitato dagli anchors ed effettua una stima statistica; in particolare determina quali classi appaiono di frequente in posizioni simili in percorsi analoghi. Queste classi probabilmente rappresentano semanticamente concetti simili. In termini di supporto all'utente, il tool ha le seguenti caratteristiche:

- **Impostare l'ontologia preferita.** Accade spesso che le ontologie di partenza non sono importanti o stabili allo stesso modo, e l'utilizzatore vorrebbe risolvere tutti i conflitti per una di esse. Questa caratteristica consente di designare una ontologia come quella preferita. Quando c'è un conflitto tra valori, invece di avvisare l'utente di tale conflitto allo scopo di risolverlo, il sistema risolve il conflitto automaticamente.

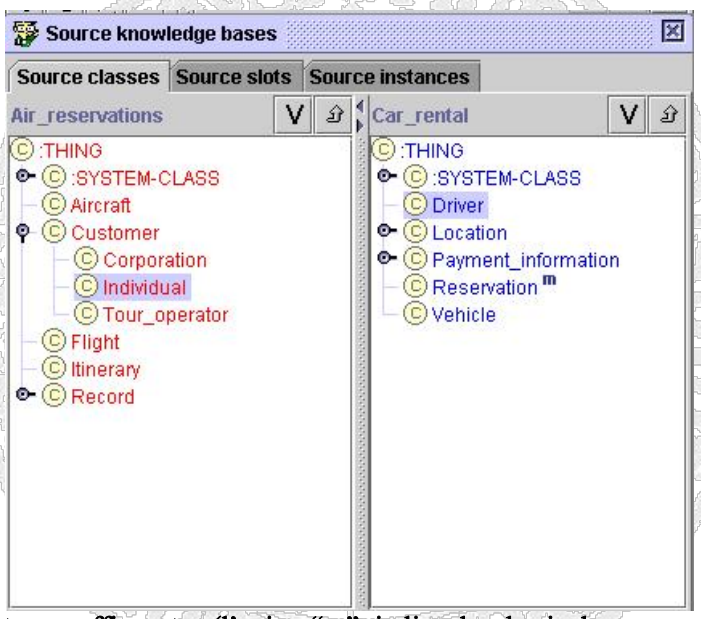
- **Conservare il focus dell'utente.** Supponiamo che un utente stia fondendo due ontologie e stia lavorando sul contenuto di una di queste. PROMPT conserva il focus d'utente, riorganizzando la propria lista di suggerimenti e conflitti e presentando per primi gli elementi che includono frame relativi agli argomenti delle ultime operazioni.

- **Fornire riscontri all'utente.** Per ogni suggerimento, PROMPT fornisce una serie di spiegazioni, a partire dal perché ha suggerito proprio l'operazione in quella posizione (viene infatti mostrata una lista). Se in seguito PROMPT cambia la posizione dell'operazione nella lista dei suggerimenti, allora spiega anche perché lo ha fatto.

Alcuni esperimenti effettuati su PROMPT e AnchorPROMPT hanno dimostrato che gli esperti seguono per l'88% i suggerimenti di PROMPT e che esso ha suggerito il 75% delle operazioni che l'utente ha infine eseguito.



Una schermata di PROMPT. La finestra principale mostra una lista di suggerimenti nel pannello in alto a sinistra e le spiegazioni per quello selezionato più in basso. Il pannello di destra della finestra mostra l'evolversi della nuova ontologia che si sta costruendo.



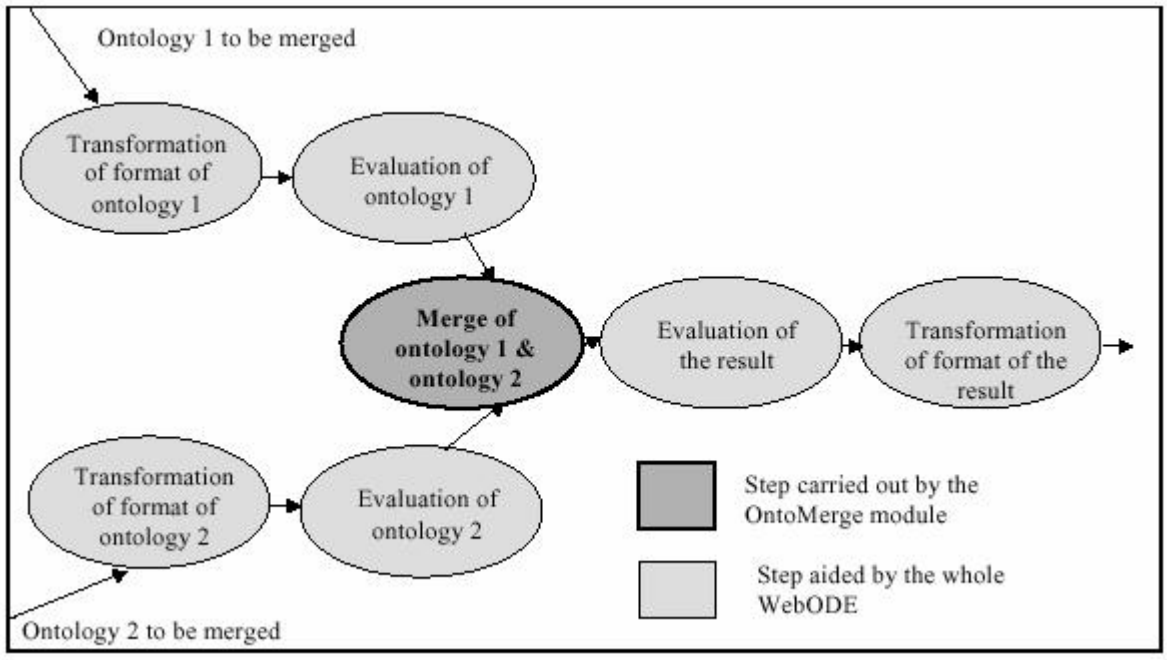
Le due ontologie di partenza affiancate. (l'apice "m" indica le classi che sono state integrate o spostate nell'ontologia fusa che si sta formando).

4.4.5 ODEMerge

ODEMerge (Ramos, 2001) è un tool per la fusione di ontologie ed è integrato in WebODE, la piattaforma software per costruire ontologie, sviluppata dall'Ontology Group della Università tecnica di Madrid. Per tale motivo, ODEMerge è un tool con architettura client-server che è impiegato nel Web.

Questo tool utilizza la **metodologia e**laborata da de Diego (de Diego, 2001) per il merging delle ontologie. Questa metodologia propone i seguenti passi (v. figura sottostante):

1) Trasformazione dei formati delle ontologie da integrare;
2) Valutazione delle ontologie;
3) Fusione delle ontologie;
4) Valutazione del risultato; e
5) Trasformazione dell'ontologia risultante in un formato adatto all'applicazione in cui dovrà essere usata.



La metodologia di ODEMerge e il suo supporto software

La metodologia stabilisce in modo molto dettagliato quali operazioni bisogna effettuare per fondere due ontologie, quando queste operazioni vanno eseguite, chi deve svolgere il singolo compito, come deve farlo, e quali sono gli output di ogni operazione. Sono proposte regole dettagliate anche per la valutazione e per il merging delle ontologie. La metodologia è basata sull'esperienza di fusione delle ontologie per il commercio elettronico.

WebODE è utile nei passi (1), (2), (4) e (5) della metodologia, e ODEMerge esegue la fusione delle tassonomie dei concetti al passo (3). Inoltre, ODEMerge è utile nel merging di attributi e relazioni, e incorpora molte delle regole dettate dalla metodologia.

ODEMerge accetta i seguenti input (v. figura in basso):

- l'ontologia 1 da integrare;
- l'ontologia 2 da integrare;
- la *tavola dei sinonimi,* che contiene le relazioni di sinonimia dei termini tra le ontologie 1 e 2.
- La *tavola degli iperonimi*[2], che contiente le relazioni d'iperonimia tra i termini delle ontologie 1 e 2

ODEMerge elabora le ontologie sulla base delle informazioni contenute nelle tavole dei sinonimi e degli iperonimi, e genera una nuova ontologia, che è la fusione delle ontologie 1 e 2 di partenza;

ovvero, il tool compara l'ontologia 1 con l'ontologia 2 considerando le tavole di sinonimia e di iperonimia e fonde le ontologie. Si prevede che nelle prossime versioni del tool, saranno disponibili anche dizionari elettronici e altre risorse linguistiche.

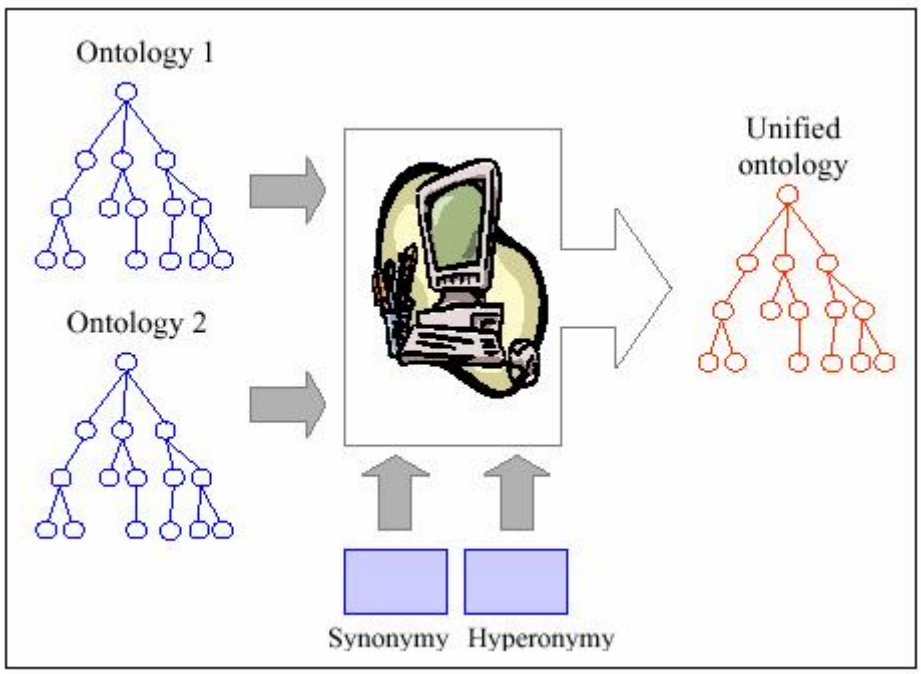
Questo tool non è interessante soltanto per le sue funzionalità attuali, ma anche perché

[2] Per *Iperonimia* s'intende il rapporto semantico tra un vocabolo di significato più generico (detto *iperonimo*) e uno o più vocaboli di significato più specifico e ristretto (detti *iponimi*).

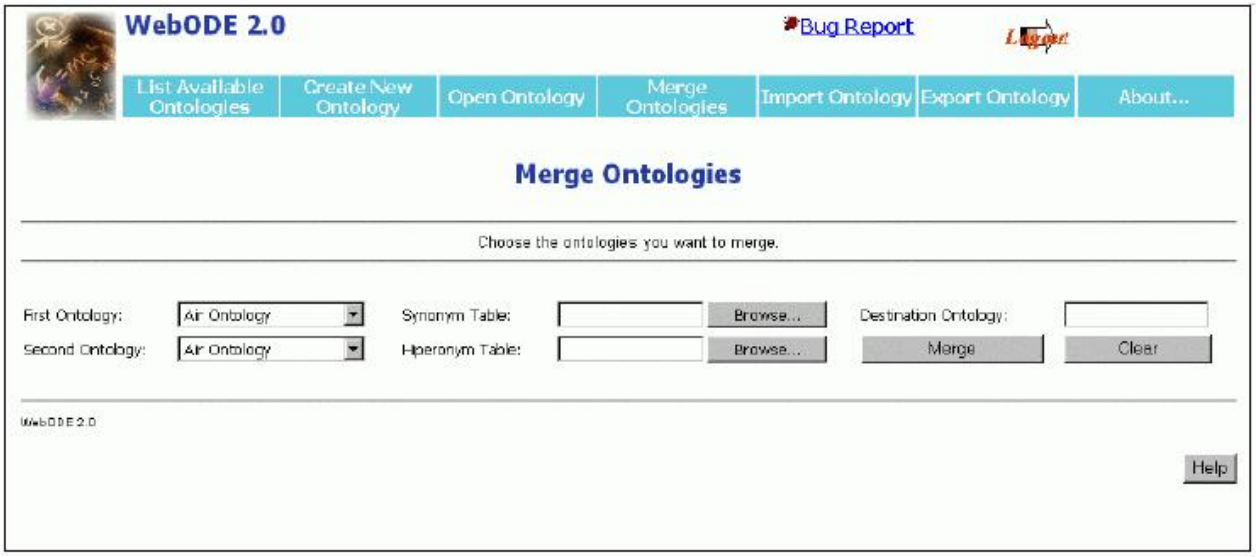
esso è facilmente estendibile: possono essere aggiunte nuove regole di merging a quelle esistenti.

Inoltre l'aggiunta di dizionari elettronici e altri tool linguistici, potrebbero sostituire le tavole di sinonimia e iperonimia.

Un'altra caratteristica importante di ODEMerge consite nella possibilità di fondere le ontologie in tutti i linguaggi di implementazione supportati da WebODE, poiché questo è l'host della piattaforma ODEMerge. Il modulo di importazione di WebODE consente di importare ontologie scritte in XML, RDF(S), OIL, DAML+OIL, Java and HTML.



Input e Output di ODEMerge



ODE Merge in WebODE

4.4.6 Confronto e valutazione dei tool

Queste osservazioni sono relative ai tool analizzati in precedenza (eccezion fatta per Chimaera): PROMPT e ODEMerge.

Architettura software

Come prima cosa si valuteranno le piattaforme di sviluppo in cui i tool sono integrati.
PROMPT and ODEMerge sono integrati rispettivamente in Protégé-2000 e WebODE.
PROMPT and ODEMerge ereditano molte delle proprie caratteristiche dai rispettivi host.
PROMPT deve essere installato in locale mentre ODEMerge è integrato perfettamente nell'area di lavoro di WebODE, la quale può essere acceduta dal Web.
Questi tool hanno la necessità di usare delle informazioni (dizionari elettronici, vocabolari, ecc.) durante il processo di fusione. Più informazioni usa un tool durante questo processo, minore sarà la necessità di intervento da parte dell'utente.

Anche l'**interoperabilità** con altri tool è un aspetto importante ed è solitamente

determinato dalla piattaforma di sviluppo di ontologie in cui il tool di merging è integrato. Un altro aspetto chiave è se il tool è in grado di fondere ontologie espresse in differenti linguaggi; tutti i tool presentati sono capaci di integrare ontologie espresse in linguaggi diversi (XML, RDFS, OIL, ecc.).

I tool di merging possono lavorare in maniera **interattiva** oppure no. PROMPT richiede l'intervento dell'utente quando bisogna scegliere tra diverse alternative. ODEMerge, invece, esegue l'intero processo in maniera del tutto automatica.

Assodato che le ontologie solitamente non sono statiche ma evolvono, la **gestione di differenti versioni di ontologie** è molto importante. Nessuno di questi tool si avvantaggia di versioni precedenti delle ontologie per il processo di merging, e nessuno di essi avverte l'utente di cambi effettuati sulle ontologie di partenza.

Da considerare è anche il **tipo di componenti che un tool può integrare.** Tutti i tool presentati consentono il merging di concetti, tassonomie, relazioni e istanze. Tuttavia, nessun tool consente di integrare assiomi.

In modo simile, bisogna considerare **i suggerimenti forniti dal tool.** PROMPT presenta all'utente le alternative possibili e fra tutti è il tool che fornisce più suggerimenti agli utilizzatori.

Un altro aspetto chiave dell'analisi è verificare se il tool è in grado di **rilevare conflitti** durante il processo di fusione. Ad esempio, nascono conflitti di nomi quando lo stesso nome è utilizzato per identificare concetti diversi in entrambe le ontologie. I tool analizzati rilevano conflitti di nomi e di strutture.

Da notare è che solo ODEMerge (integrato in WebODE) supporta una dettagliata **metodologia** per il processo di fusione di ontologie. Gli altri utilizzano metodi non dettagliati. Per quanto riguarda il **sistema di supporto,** la documentazione su come usare il tool non è di solito acclusa al programma, ma si trova in un documento separato o in tutorial realizzati dai produttori.

Le caratteristiche di **editazione e visualizzazione** sono fortemente condizionate dalla

piattaforme di sviluppo in cui questi tool sono integrati.

Infine bisogna valutare l' **esperienza d'uso del tool**. PROMPT e ODEMerge sono stati usati per l'elaborazione di molte ontologie di diverso tipo e in differenti domini di conoscenza.

4.4.7 Considerazioni generali

Le considerazioni principali che si possono trarre dallo studio effettuato sono le seguenti:

1) Si seguono due approcci per integrare ontologie:

   - A partire dalle istanze delle ontologie. Si integrano le ontologie incrociando le rispettive istanze.
   - A partire dai concetti delle ontologie. PROMPT e ODEMerge integrano le ontologie ricercando le similitudini tra i concetti.

2) *Tutti i tool hanno bisogno dell'intervento dell'utente per giungere al merging definitvo*. PROMPT permette all'utente di guidare il processo. L'utente in ODEMerge può modificare il risultato del processo.

3) *Nessun tool si avvantaggia di precedenti versioni delle ontologie risultanti.* Ad esempio, se l'ontologia O è la fusione di O1-v.1 e O2-v.1, e si sviluppa la versione 2 di O1, il tool deve ripetere completamente il processo di fusione di O1-v.2 e O2-v.1.

4) *Tutti i tool consentono la manipolazione di ontologie implementate in differenti linguaggi.* Ciò è molto importante, poiché le ontologie possono avere differenti provenienze e possono essere usate in diverse applicazioni.

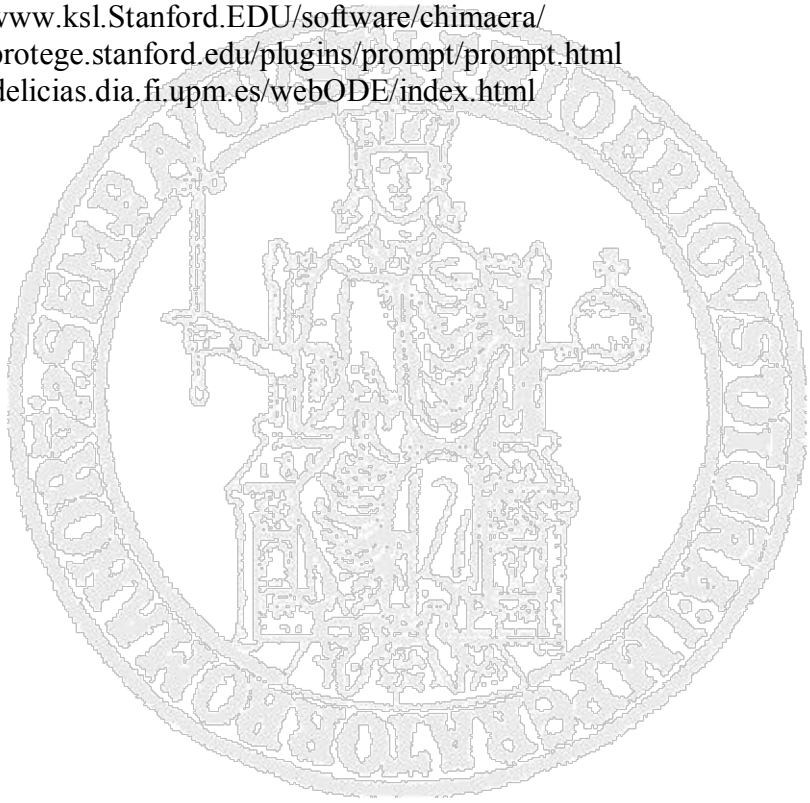
5) *Nessun tool consente l'integrazione di assiomi e regole.* Ciò vuol dire che i tool di oggi, non permettono il merging di ontologie di un certo calibro (*heavyweight ontologies*).

Riassumendo, si può dire che nello sviluppo di tool per l'integrazione di ontologie sono stati affrontati aspetti importanti (ad esempio, la diversità dei linguaggi). Tuttavia, altri punti chiave (quali l'integrazione di assiomi) rappresentano un campo non ancora ben esplorato.

D'altra parte, l'evoluzione naturale dei tool di integrazione, dovrebbe portare ad accrescere l'utilizzo di informazioni e diminuire l'intervento dell'utente durante il processo.

4.4.8 URL

Chimaera: http://www.ksl.Stanford.EDU/software/chimaera/
PROMPT: http://protege.stanford.edu/plugins/prompt/prompt.html
WebODE: http://delicias.dia.fi.upm.es/webODE/index.html



Caratteristica	Prompt	ODEMerge
Il tool di merging è integrato in un tool di sviluppo? Quale?	Sì, Protégé-2000	Sì, integrato in WebODE
Il tool deve essere installato localmente?	Sì	No
Tolleranza ai guasti	---	---
Gestione di backup	No	Sì, eseguita dal tool host.

Architettura dei tool di integrazione

Caratteristica	Prompt	ODEMerge
Dizionari elettronici, thesauri, ecc.	No (ma si possono integrare)	No (sebbene possano essere incorporati)
Definizione dei concetti e valori degli slot	Sì (ed è necessario per il processo)	Sì (ma non è necessario per il processo)
Struttura a grafo	Sì (ma non è necessario per il processo)	Sì
Istanze di concetti	Sì (ma non è necessario per il processo)	No
Input dall'utente	Sì (ed è necessario per il processo)	Sì (ed è necessario per il processo)

Informazioni usate durante il processo di integrazione

Caratteristica	Prompt	ODEMerge
Il tool può interoperare con altri tool o altri sistemi di informazione?	Sì, attraverso il meccanismo di importazione del tool host (Protégé-2000)	Sì. Le ontologie possono essere generate in differenti linguaggi (XML, RDF(S), OIL, DAML+OIL, FLogic, Prolog, Java) attraverso WebODE.
Può il metodo di fusione delle ontologie essere espresso in diversi linguaggi? Quali?	Sì, attraverso il meccanismo di importazione del tool host (Protégé-2000) - RDFS, XML Schema, OIL.	Il tool può utilizzare il modulo di importazione di WebODE.

Interoperabilità

Caratteristica	Prompt	ODEMerge
In che modo lavora il tool?	Interattivo + automatico	Completamente automatico

Modalità di lavoro

Caratteristica	Prompt	ODEMerge
Il tool si avvantaggia di versioni precedenti delle untologie?	No	No
Avvisi sul cambiamento delle ontologie di partenza?	No	No

Gestione di differenti versioni delle ontologie

Caratteristica	Prompt	ODEMerge
Concetti?	Sì	Sì
Slot?	Sì	Sì
Template Slot?	Sì	Sì
Tassonomie	Sì	Sì
Concetti?	Sì	Sì
Relazioni ?	Sì	Sì
Partizioni e/o Decomposizioni	No	Sì
Relazioni e Funzioni?	Relazioni – sì, funzioni - no	Sì
Merge di elementi coinvolti?	Per relazioni binarie	Per qualunque tipo di relazione
Integrazione di assiomi?	No	No
Integrazione di regole?	No	No
Istanze	Sì	No
Di concetti?	Sì	No
Di relazioni?	Sì	No
Asserzioni?	No	No

Componenti che possono essere integrati

Caratteristica	Prompt	ODEMerge
Il tool rileva conflitti di nomi?	Si	Sì (attraverso WebODE)
Il tool rileva conflitti di strutture?	Sì	Sì (attraverso WebODE)

Rilevazione di conflitti

Caratteristica	Prompt	ODEMerge
Visualizzazione passo-passo del processo?	Grafica-Tabulare- Gerarchica	Non Grafica
Visualizzazione simultanea delle ontologie da integrare?	Si	No
Potature dei grafici (viste) delle ontologie da integrare?	Sì attraverso il tool host	Sì attraverso il tool host
Zoom?	Sì attraverso il tool host	Sì attraverso il tool host

Editazione e visualizzazione

4.5 Tool per la valutazione di ontologie

4.5.1 Introduzione

Il semantic web attrae ricercatori da ogni parte del mondo. Numerosi tool e applicazioni per il web semantico sono già disponibili e il loro numero cresce velocemente.

Le ontologie giocano un ruolo importante per il web semantico come sorgenti di termini formalmente definiti utili ai fini della comunicazione. Il loro scopo è quello di catturare la conoscenza di un dominio in maniera generale e fornire una comune rappresentazione della conoscenza, che può essere riusata, condivisa tra gruppi e applicazioni differenti.

Tuttavia, a causa della loro dimensione, della loro complessità e della necessità di dover convergere verso un modello comune di *comprensibilità,* le ontologie sono lontane dall'essere convenienti.

Lo sviluppo di soluzioni per ontologie di larga scala induce a seguire numerose attività separate e richiede molteplici tool per consentirne l'uso. Pertanto elementi pragmatici come l'interoperabilità sono requisiti chiave per l'utilizzo di questa tecnologia.

La grande visibilità del web semantico, i suoi tool e le sue applicazioni attraggono molti partner industriali, dalle istituzioni accademiche alle società commerciali. Come è chiaramente intuibile, queste istituzioni hanno differenti esigenze. Tool differenti creati da diversi produttori, hanno la necessità di collaborare; essi pertanto sono tipicamente soluzioni non standalone, integrate in altri framework. Questi contesti devono essere aperti agli ambienti commerciali e fornire interfacce per standard industriali.

Una sistematica valutazione delle ontologie e delle relative tecnologie può portare a un consistente livello di qualità, tanto da poter essere impiegate dalle industrie. Per il futuro, questo sforzo potrebbe anche condurre a certificazioni e benchmark standardizzati.

Saranno di seguito analizzati alcuni tool per lo studio delle proprietà e delle tecnologie usate per lo sviluppo di ontologie.

4.5.2 Criteri di Valutazione

Vi sono due aspetti principali da considerare: (1) la valutazione delle proprietà delle ontologie generate da tool per lo sviluppo, (2) la valutazione della tecnologia, ad esempio alcuni tool e applicazioni che includono a loro volta tool per la valutazione di proprietà.

Per quanto riguarda le ontologie generate dai tool di sviluppo, possono essere controllati la conformità ai linguaggi e la consistenza.

Conformità ai liguaggi significa che la sintassi della rappresentazione dell'ontologia in uno specifico linguaggio è conforme allo standard. Questo può essere uno standard proprietario
ben-documentato, oppure si tratta di uno standard industriale perlopiù determinato da un'implementazione cui fare riferimento.

Valutare la consistenza implica verificare che l'estensione di un tool, assicuri che le ontologie risultanti siano consistenti rispetto alla loro semantica, ad esempio che parti diverse di una rappresentazione di un'ontologia non si contraddicano.

L'interoperabilità indica la semplicità con cui si possono scambiare ontologie tra tool differenti. Questa caratteristica include aspetti quali "il tool è capace di interpretare i risultati di un altro allo stesso modo?". Questo controllo è più informativo della semplice conformità al linguaggio perché esamina se i diversi tool interpretano le stesse cose allo stesso modo. Spesso, poi le informazioni possono essere rappresentate nello stesso linguaggio in modi diversi.

Turn around ability è la capacità per la quale il risultato di un tool viene rappresentato all'utente sempre allo stesso modo. Ad esempio una restrizione per un valore può essere

rappresentata come una restrizione di intervallo di valori o attraverso un vincolo. Se il tool la mostra come una restrizione di intervallo, non dovrebbe mostrarlo come un vincolo la prossima volta che l'utente apre la stessa ontologia.

Le Prestazioni indicano in particolare lo sforzo di esecuzione del tool, ad esempio quanto tempo occorre per risolvere una particolare operazione di inferenza, per memorizzare le ontologie, ecc. I test di valutazione automatica (benchmarks) devono essere sviluppati per valutare questi tipi di prestazioni.

Allocazione di memoria indica quanta memoria è richiesta dal tool per lavorare con le ontologie. Similmente alle prestazioni, i benchmark devono essere capaci di testare l'allocazione di memoria. E' necessario anche valutare chiaramente cosa si intende per dimensione o complessità di un'ontologia oppure la complessità di un'operazione e quali parametri influenzano questa dimensione, in che modo, ecc.

La **Scalabilità** valuta le prestazioni e le richieste di memoria del tool in risposta all'aumento del numero di ontologie e di operazioni. Esamina problemi del tipo "quanto la crescita lineare di ontologie, influisce sulla memoria allocata dal tool?".

L'integrazione in framework valuta la semplicità di passare tra un tool e un altro. Ad esempio non è molto conveniente che per passare da un tool ad un altro sia necessario salvare l'ontologia, trasformarla poi con un tool differente in un altro linguaggio per poterla "trasferire" al tool di destinazione. Ambienti completamente integrati simili ai ben noti ambienti di programmazione, devono essere l'obiettivo dei programmi per la costruzione di ontologie.

Ultima, ma non meno importante delle altre caratteristiche è la **connettività** verso altri tool. Da un lato consiste in connettori per istanze a database, a sistemi quali MS exchange, Lotus Notes, ecc e dall'altro lato in interfacce verso il tool stesso, per consentire l'uso delle proprie funzionalità da parte di programmi esterni.

4.5.3 OntoAnalyser

OntoAnalyser e OntoGenerator (Institute AIFB e ontoprise GmbH), sono entrambi esempi di tool di valutazione, realizzati come plugin per OntoEdit, che è un ambiente grafico per l'ingegneria delle ontologie

La struttura a plugin consente estensioni flessibili per le principali funzionalità di OntoEdit (anche da terze parti). Sebbene OntoEdit si occupi del ciclo di vita completo per lo sviluppo di un'ontologia , alcune operazioni sono supportate poco dalle funzionalità di base. Con l'utilizzo di plugin specializzati, si può ottenere un supporto migliore. Saranno oggetto di valutazione ontologie, (i) create con OntoEdit oppure (ii) importate da altri tool come Protégé e WebODE durante il processo di sviluppo.

Ognuno dei due plugin, copre differenti aspetti dei criteri di valutazione presentati. OntoAnalyser focalizza l'attenzione sulla valutazione delle proprietà delle ontologie, in particolare alla conformità al linguaggio e alla consistenza. OntoGenerator, invece, valuta i tool, in particolare analizza le prestazioni e la scalabilità di questi. Dall'esperienza ricavata dallo sviluppo delle ontologie, si nota che ontologie che hanno obiettivi diversi, devono anche possedere differenti proprietà. Queste proprietà possono anche essere differenti in diversi progetti o applicazioni. Ad esempio un tool che consente di visualizzare gerarchie di concetti potrebbe avere il bisogno di mostrare solo ereditarietà singole.

La definizione dei metodi di valutazione per queste proprietà deve dunque essere molto flessibile e manutenibile; perciò non è conveniente programmarla in un tool. La logica è un modo molto potente e comodo per esprimere vincoli o per esaminare proprietà di un'ontologia ad un livello astratto. Per questo motivo, il linguaggio per regole e vincoli deve essere in grado di avere accesso alla stessa ontologia, ad esempio per *fare commenti* sulle classi, sulle relazioni, sulle sottoclassi, ecc.

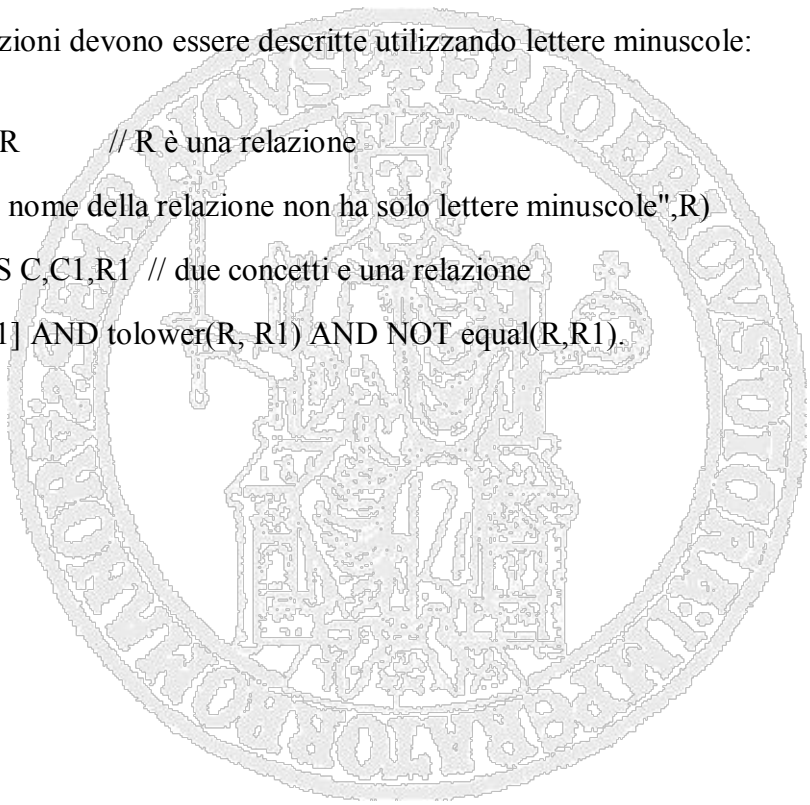
Il linguaggio F-logic consente di definire regole ed espressioni sull'ontologia (concetti, sottoconcetti, relazioni). Ad esempio, per verificare se in un'ontologia un concetto ha al massimo un *superconcetto* (un concetto più in alto nella gerarchia) si può ricorrere alla seguente regola:

```
FORALL C
check("Il concetto ha più di un superconcetto",C)
<- EXISTS S1,S2
      C::S1 AND C::S2 AND NOT equal(S1,S2).
```

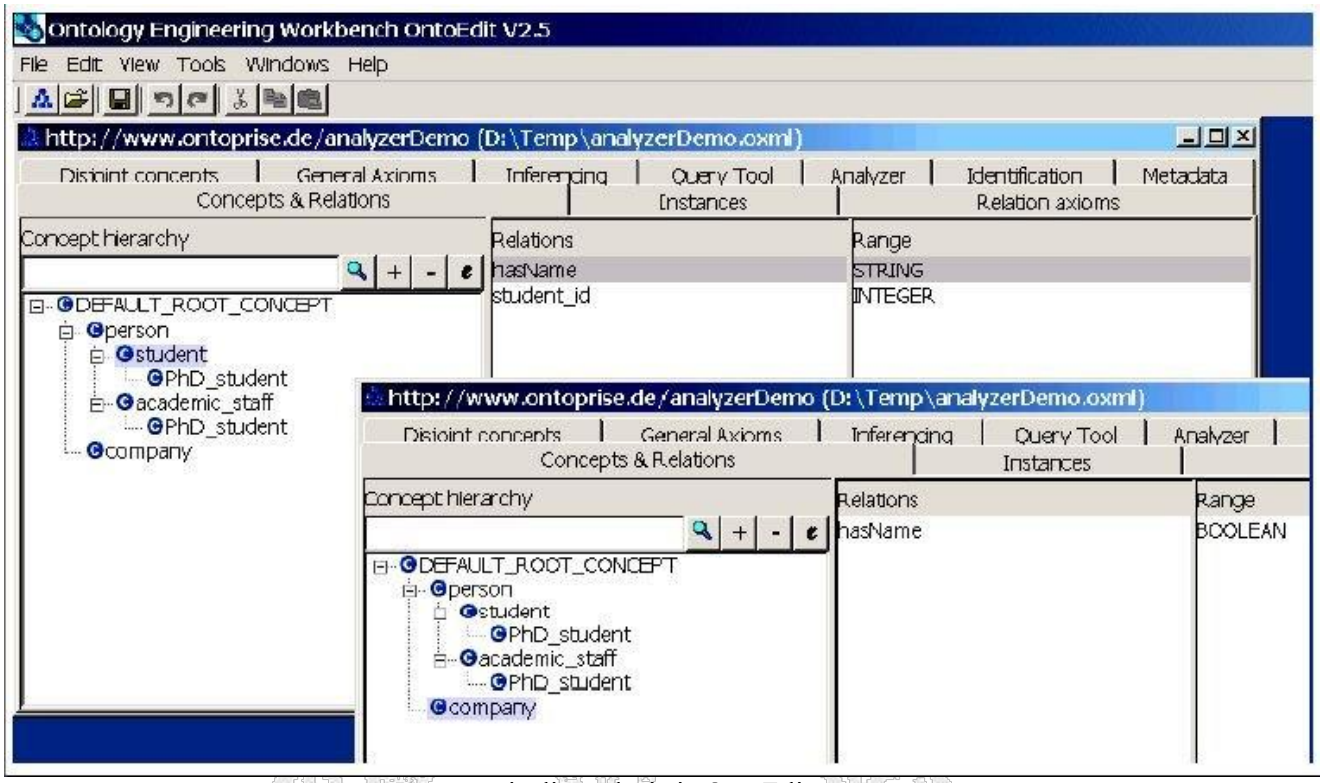
dove S1 e S2 sono superconcetti.

Allo stesso modo è semplice definire regole per verificare, ad esempio, la disgiunzione delle classi, convenzioni per i nomi delle relazioni, ecc. Le norme di un progetto dicono che le relazioni devono essere descritte utilizzando lettere minuscole:

```
FORALL R          // R è una relazione
  check("Il nome della relazione non ha solo lettere minuscole",R)
<- EXISTS C,C1,R1  // due concetti e una relazione
C[R=>>C1] AND tolower(R, R1) AND NOT equal(R,R1).
```



OntoAnalyser è un tool che prende in input regole come queste, fa partire il motore di inferenza Ontobroker passandogli queste regole e fornisce all'utente i risultati dell'analisi. OntoAnalyser è in grado di importare diversi package di regole, ognuno specifico per un progetto o un tool.



Un esempio di ontologia in OntoEdit

Si illustra un esempio di applicazione delle due regole definite con OntoAnalyser. La figura precedente mostra una schermata di OntoEdit con un esempio di ontologia. Da notare che (i) il concetto `PhD_student` è un sottoconcetto di `student` e lo stesso vale per `academic_staff` (ereditarietà multipla), e (ii) i due concetti `student` e `company` hanno una relazione[3] `hasName`, ma ognuna con un differente tipo, l'una `STRING`, l'altra `BOOLEAN`.

Le due regole, applicate all'ontologia considerata riportano i seguenti risultati di

[3] In alcuni casi le relazioni di tipi predefiniti come "STRING" e "BOOLEAN" sono detti "attributi".

valutazione, visualizzati in figura 2. Primo, l'ereditarietà multipla del concetto `PhD_student` ha generato il messaggio d'errore "Il concetto ha più di un superconcetto".

Secondo, l'esistenza della relazione `hasName` di diverso tipo, ha generato il messaggio di errore "esiste una stessa relazione che ha diversi tipi".

OntoAnalyser è realizzato come plugin per OntoEdit. Pertanto è ben integrato nell'ambiente di sviluppo delle ontologie. Un'ontologia può essere creata con OntoEdit e durante questo processo, può essere analizzata "al volo"(*on the fly*) utilizzando OntoAnalyser. In questo modo non sono

necessarie trasformazioni dell'ontologia, né tantomeno importazioni e/o esportazioni per eseguire la valutazione. Per importare ontologie, bisognerebbe controllare l'interoperabilità di questo tool con quello usato per lo sviluppo.

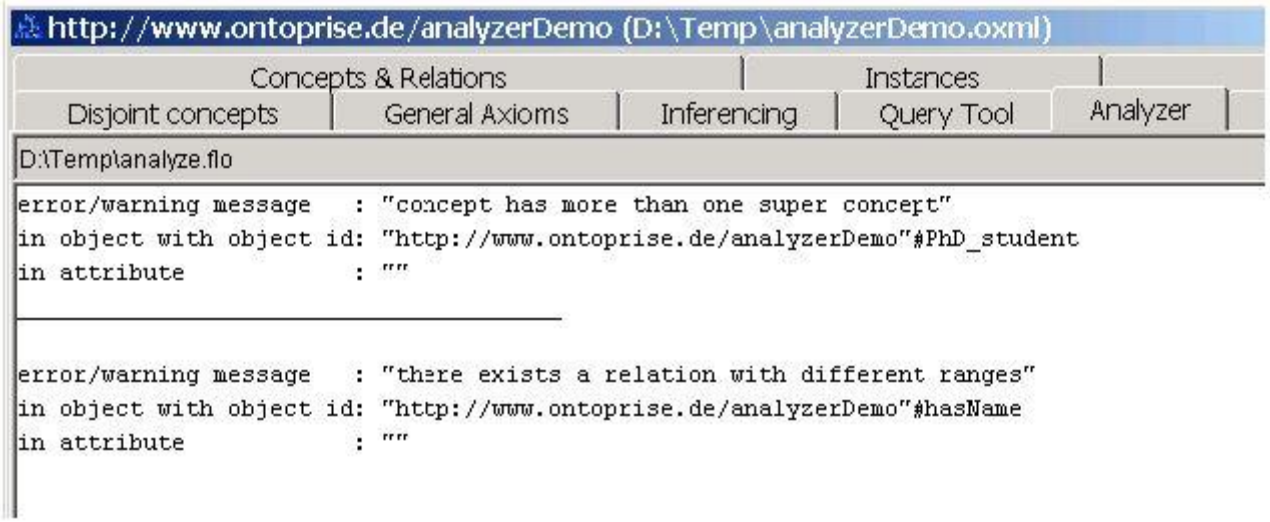


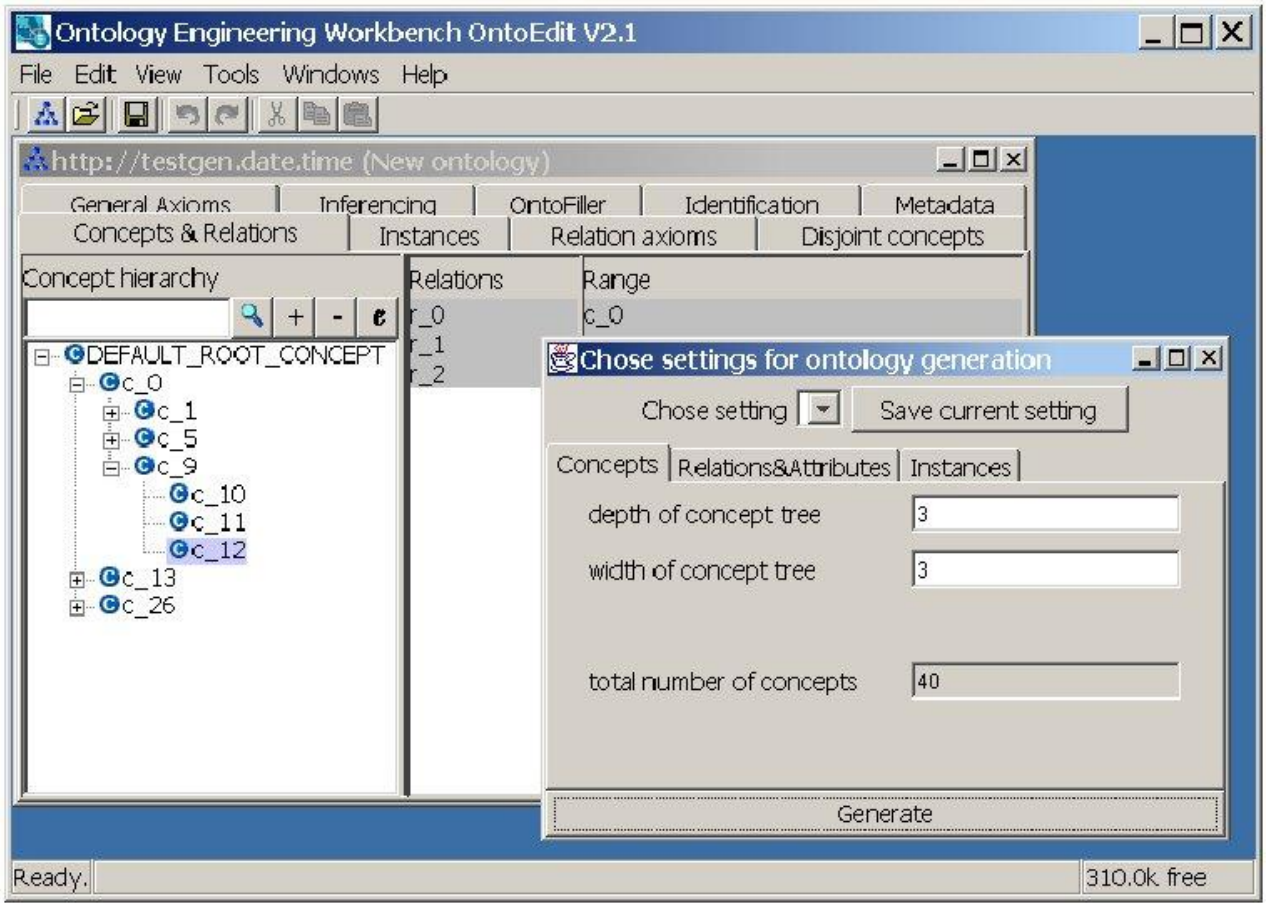
Figura 2. Risultati della valutazione di OntoAnalyzer relative all'esempio.

Nelle versioni future, saranno disponibili package di regole standard per la valutazione di ontologie di vari ambiti per supportare un'efficace ingegneria e per migliorare la qualità delle ontologie allo stesso tempo.

4.5.4 OntoGenerator

OntoGenerator è in grado di effettuare dei test di prestazione ("stress tests") dei tool per lo sviluppo di ontologie. Crea delle ontologie "sintetiche" che non hanno lo scopo di rappresentare un dominio di interesse, ma piuttosto di soddisfare specifici parametri tecnici quali ad esempio il numero di concetti e istanze, oppure certi tipi di regole.

Originariamente, OntoGenerator è stato progettato per supportare l'ottimizzazione delle regole di valutazione di Ontobroker realizzate attraverso F-Logic (come quelle presentate nel precedente paragrafo). Le prestazioni delle regole di valutazione dipendono fortemente dalla sequenza in cui "i corpi" delle regole sono valutati. Un esperimento di ottimizzazione realizzato dai professionisti del settore ha dimostrato che **un'ottima sequenza** produce, per un determinato esempio, 20.000 risultati intermedi di valutazione, mentre una sequenza errata ne produce ben 38.000.000. L'ottimizzatore ricerca la migliore sequenza utilizzando un algoritmo genetico (basandosi cioè sui tipi di gerarchie). Per valutare queste strategie di ottimizzazione, sono necessari test su numerose ontologie di diverso tipo e differenti gruppi di regole con particolari proprietà. Per questa ragione OntoGenerator produce "ontologie sintetiche" *on the fly* sulla base di parametri predefiniti. La figura mostra un'ontologia generata da OntoGenerator. In questa versione è possibile definire i seguenti parametri per la generazione delle ontologie: (i) l'estensione dell'albero dei concetti, (ii) l'ampiezza dell'albero dei concetti, (iii) il numero complessivo di relazioni, (iv) il numero totale degli attributi e (v) il numero complessivo delle istanze.



La generazione di ontologie sintetiche con OntoGenerator

In particolare i parametri producono i seguenti effetti nella generazione: (i) definisce quanto sarà esteso l'albero generato (nell'esempio: a 3 livelli), (ii) definisce il numero di figli per ogni concetto generato (3 figli), (iii) e (iv) definiscono il numero complessivo di relazioni e attributi che sono connessi agli oggetti in modo casuale (in questa versione) e (v) definisce il numero di istanze da creare; i concetti da istanziare sono scelti in maniera random.

Nelle future versioni si prevede l'uso di misure statistiche per la selezione dei concetti, utili ad esempio nella fase di collegamento di relazioni e attributi e durante la creazione di istanze, poiché è importante il livello (dell'albero) in cui vi è una relazione. Ciò è dovuto ai meccanismi di ereditarietà: una relazione presente ad un livello alto avrà come risultato una gran quantità di relazioni ai livelli bassi della gerarchia dei concetti. E' prevista anche

l'aggiunta di un generatore di regole, in modo da poter esaminare anche le seguenti caratteristiche:

- **Estensione dell'albero di regole:** la lunghezza della catena di regole, da cui una regola dipende.
- **Ciclicità delle regole:** la dipendenza tra le regole può essere ciclica.
- **Lunghezza dei cicli delle regole:** la lunghezza dei cicli di dipendenze.
- Complessità dei corpi delle regole: la complessità delle formule.
- **Transitività**: la quantità di regole transitive presenti.

4.5.5 OntoClean in WebODE

Il gruppo ontologico del CNR (Centro Nazionale di Ricerca) ha elaborato una serie di criteri di valutazione di un'ontologia, basati su nozioni filosofiche quali: rigidità, identità, unità, dipendenza, ecc. Questi criteri sono usati in OntoClean, la metodologia di valutazione di ontologie proposta da questo gruppo per "riordinare" ontologie complesse.
Su un altro fronte, il gruppo ontologico del laboratorio di intelligenza artificiale dell'Università Politecnica di Madrid (UPM) ha sviluppato una metodologia chiamata METHONTOLOGY per lo sviluppo, la re-ingegnerizzazione e la valutazione di ontologie. Il gruppo dell'UPM ha anche sviluppato WebODE, un tool che fornisce supporto alle attività principali (concettualizzazione, implementazione, reingegnerizzazione, valutazione) identificate nella metontologia.
Sebbene i lavori eseguiti dai due gruppi siano complementari, esiste una prima versione di una metodologia congiunta che integra la metodologia di OntoClean nella fase di concettualizzazione della metontologia quando è costruita la tassonomia. Inoltre esiste un plugin di WebODE chiamato proprio OntoClean che dà supporto alla metodologia di OntoClean e a quella integrata nel tool. Questo plugin fornisce le seguenti funzionalità:

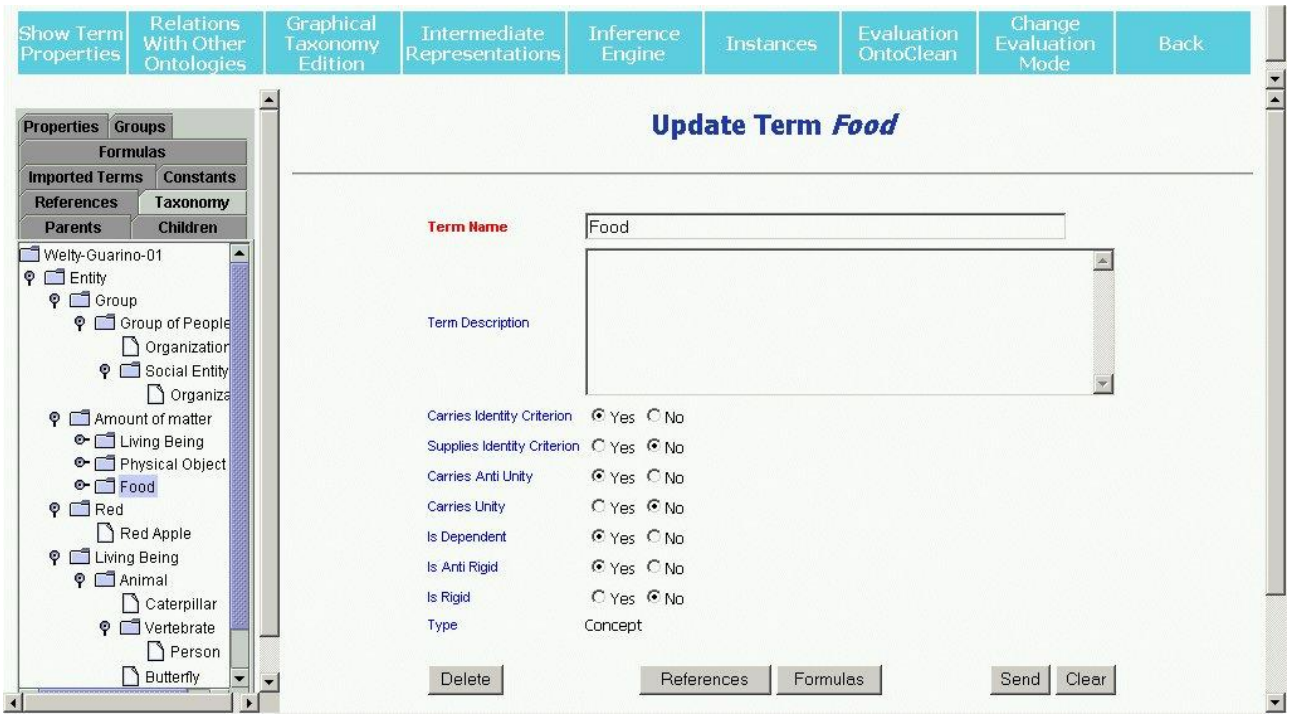
Stabilire la modalità di valutazione. L'utente può scegliere se il sistema dovrà mostrare gli errori ogni volta che venga rilevata una violazione degli assiomi di OntoClean, oppure se esso dovrà mostrarli solo in seguito ad una richiesta dell'utente.

Assegnare meta-proprietà ai concetti. L'utente può aggiungere meta-proprietà che riguardano l'identità, l'unità, le dipendenze, la rigidità.

Concentrarsi (o meno) su proprietà non rigide. L'utente può decidere che le proprietà non rigide siano mostrate in maniera meno evidente. Questo è importante perché OntoClean è basato su proprietà rigide.

Valutazioni in accordo ai vincoli tassonomici. Il sistema effettua una diagnosi delle ontologie in accordo ai principi del gruppo del CNR. L'utente può rilassare o "stressare" la valutazione selezionando un numero di criteri minore o maggiore.

Il principale vantaggio del plugin OntoClean è che i criteri usati per valutare le ontologie sono dichiarati nel modulo di concettualizzazione di WebODE.
OntoClean usa il motore di inferenza di WebODE per rilevare le inconsistenze sulle tassonomie, consentendo all'utente di riparare la tassonomia in accordo ai vincoli menzionati in precedenza.
Una volta che l'ontologia è corretta, può essere esportata in altri linguaggi di implementazione attraverso il traduttore di WebODE.



Una schermata del modulo OntoClean in WebODE

4.5.6 Considerazioni generali

Il crescente interesse legato al web semantico, attrae oltre alle accademie, molti partner industriali. Le ontologie e le tecnologie collegate come lo sviluppo di tool per il web semantico hanno la necessità di soddisfare forti requisiti per affrontare la domanda dei partner industriali. Il framework di valutazione presentato ambisce ad assicurare i necessari livelli di qualità. Si distinguono in esso due aree principali: (i) proprietà delle ontologie, ad esempio conformità al linguaggio e consistenza, e (ii) proprietà di tecnologia, ad esempio l'interoperabilità, la turn around ability, le prestazioni, l'allocazione di memoria, la scalabilità, l'integrazione in altri framework e la connettività e le interfacce. Sono stati presentati implementazioni di riferimento del framework di valutazione, OntoAnalyser e OntoGenerator. Ciascuno copre differenti aspetti dei criteri di valutazione. OntoAnalyser, e OntoClean focalizzano la loro attenzione sulla valutazione delle proprietà delle ontologie, in particolare sulla conformità al linguaggio e consistenza.

OntoGenerator, invece, si concentra sull'analisi delle prestazioni e della scalabilità.

Le implementazioni *reali* (quelle d'uso comune) richiedono banchmark e sforzi per la standardizzazione. Pertanto è necessario che le comunità che si occupano del web semantico

accrescano il loro impegno nella ricerca e nello sviluppo di nuovi criteri standard, di nuove certificazioni e dello sviluppo di tool che integrino questi criteri allo scopo di valutare le ontologie e le tecnologie ad esse collegate.

Caratteristica	OntoAnalyzer	OntoGenerator	OntoClean
Proprietà delle ontologie			
Conformità al linguaggio (sintassi)	X		X
Consistenza (sintassi)	X		X
Proprietà delle tecnologie			
Interoperabilità			
Turn around			
Prestazioni		X	
Allocazione di memoria		X	
Scalabilità		X	
Integrazione in framework			
Connettori e Interfacce			

Valutazione delle implemetazioni rispetto ai criteri di valutazione

Capitolo 5

L'evoluzione delle ontologie

5.1 L'evoluzione futura

A parere di molti, il web semantico è il simbolo della prossima generazione del world wide web. La sua capacità di rappresentare informazioni "ricche" (di significato) e di permettere alle macchine di interpretare queste informazioni, potrebbe voler dire che i servizi di ricerca e filtraggio delle informazioni renderanno le nostre vite migliori. Ma cos'è che rende i metodi di rappresentazione attuali così inadeguati e la semantica così vitale? Ciò che sembra rallentare l'adozione del web semantico non sono i dati; non sono '7' o 'gatto'. E' la mancanza di regole e significati dei dati, l'assenza di definizioni rigorose, che non consente alle macchine, e non agli esseri umani lenti e soggetti a errore, di poter correttamente interpretare e elaborare velocemente questi dati; si può definire conoscenza qualcosa del tipo: "gli anni sabbatici ricorrono ogni 7 anni" oppure "il gatto e il cane sono mammiferi". Le ontologie sono utilizzate per codificare questo tipo di conoscenza. Il futuro del web semantico sembra legato quindi con quello delle ontologie.
Quello a cui siamo abituati è che l'autore sia il responsabile della preparazione dei dati, mentre l'utilizzatore sia responsabile dell'interpretazione e dell'elaborazione di questi. Tuttavia, la capacità di elaborazione della mente umana è poca cosa in relazione ai numerosi problemi che richiedono di essere elaborati, per trovare soluzioni oggettive.

Simon chiama questa difficoltà *bounded rationality* (razionalità limitata). Questi problemi, spesso, si presentano molto complessi e ciò richiede che per essere analizzati si ricorra alla loro scomposizione. Ciò implica la necessità di identificare sistematicamente i componenti del sistema allo scopo di costruire soluzioni adeguate oppure di re-implementare progetti preesistenti.

Per lo scambio di queste informazioni, nel web, si adotta l'XML, un mezzo di esplicita rappresentazione della conoscenza, ma che non porta con sé alcuna semantica e "costringe" l'utente a *interpretare il significato dei dati condivisi,* per riuscire a comprendere in maniera chiara il dominio applicativo. Nella pratica, conoscere il dominio del problema aiuta a risolvere i problemi che si possono incontrare durante l'elaborazione. Ciò è di fondamentale importanza quando il problema è complesso e si ha necessità (come su ricordato) di *suddividere il lavoro*. La collaborazione è possibile solo tra coloro che conoscono il dominio e le sue caratteristiche. Pertanto si deve ricorrere ad una preventiva "spiegazione" del problema tra tutti i partecipanti al gruppo di lavoro.

Le ontologie rappresentano la soluzione per tali problemi, perché esse codificano anche la semantica dei termini e fanno sì che il significato associato a un concetto sia lo stesso per l'autore e per l'utente. Da tenere presente, però, che le ontologie sono recenti e non hanno ancora un livello di maturazione adeguato (di cui invece XML può vantarsi); per tale motivo le ontologie sono effettivamente adottate solo nelle situazioni in cui la capacità di rappresentare la semantica è tanto importante da scavalcare i vantaggi della maturità di XML.

Al giorno d'oggi, le ontologie sono utilizzate prevalentemente da esperti nel campo delle nuove tecnologie e ciò rende il loro uso quasi "proibitivo" per gli utenti medi. La complessità dell'utilizzo dei tool esistenti per la costruzione di ontologie, rallenta ulteriormente la loro diffusione. Questi programmi infatti sono ancora in una fase di transizione, pertanto non sono in grado di guidare il neofita durante tutto il ciclo di

sviluppo dell'ontologia, tanto più che spesso essi richiedono l'intervento dell'utilizzatore. Le soluzioni ai problemi, però possono essere scovate solo da chi ha una buona conoscenza delle tecniche e delle metodologie. Si prevede pertanto che i prossimi tool provvederanno ad una costruzione pressoché automatica delle ontologie a partire da una semplice descrizione del dominio del problema in linguaggio naturale.

Le ontologie sono largamente utilizzate. La mancanza di uno standard per la costruzione di ontologie non facilita certo lo scambio di informazioni che è alla base della conoscenza. Diverse società, aziende, università riescono a condividere dati solo al proprio interno. Ciò fa presumere che nell'immediato futuro si formi un gruppo di standardizzazione degli schemi di rappresentazione, delle relazioni, dei vincoli e degli assiomi utilizzabili. L'utilizzo delle ontologie garantisce un risparmio (logico e temporale) nella condivisione della conoscenza e una maggiore capacità di separazione dei compiti. Ciascun collaboratore può concentrarsi su un'area specifica del problema e in un momento successivo integrare le diverse ontologie per la creazione dell'ontologia globale. E' pertanto prevedibile che in futuro le ontologie saranno lo strumento principe per la rappresentazione e la descrizione della conoscenza.

Riassumendo nel prossimo futuro assisteremo allo:

Sviluppo di tool che siano pratici e semplici da usare da un qualunque utente, che non sia necessariamente un esperto di rappresentazione della conoscenza.

Sviluppo di ontologie decentrate e versatili che abbiano valore in se stesse, ma tali che il loro intero potenziale sia espresso quando sono utilizzate insieme ad altre ontologie per lo scambio di informazioni.

Restano però delle domande per le quali è necessario trovare una risposta: Come devono lavorare questi tool? Come si possono costruire ontologie decentralizzate e versatili? Come devono essere organizzate queste ontologie per consentire una futura condivisione dei dati? Queste e altre domande sono alla base della nascita del nuovo web, più intelligente e più ricco di servizi di quello che conosciamo.

Conclusioni

Le ontologie svolgono il ruolo di intermediari nella comunicazione in campo informatico; esse sono praticamente indispensabili quando si presenta la necessità di avere una forte semantica sui dati.

Le ontologie di scopo e di dominio assolvono egregiamente tutte le mansioni richieste dai bisogni di collaborazione tra varie comunità. Il web degli ultimi anni, orientato ai servizi e all'integrazione di funzionalità, sfrutta questi schemi di rappresentazione per rendere più semplice la condivisione delle informazioni e per migliorare l'interfaccia utente e farla diventare sempre più *friendly*, in modo da guidare l'utente verso ciò che gli interessa in maniera intelligente, *interpretando* cioè le sue richieste. L'insieme dei servizi che garantiscono, attraverso l'uso di ontologie, una "comunicazione semantica" tra l'uomo e il web, costituisce il web semantico. L'uso di linguaggi per la costruzione di ontologie, consente di specificare tutti i particolari di un dominio, ma questi sono indicati per i professionisti del settore, non per semplici utenti che hanno conoscenze informatiche ridotte. Ecco perché sono nati software in grado di rappresentare graficamente le ontologie e capaci di guidare in tutto o in parte il processo di costruzione di queste. Essi sono stati suddivisi in questo elaborato in diversi gruppi per tenere conto delle loro principali funzionalità: sviluppo, integrazione e valutazione di ontologie. Tutti i tool sono stati valutati sulla base di alcuni set di criteri specifici per ogni gruppo.

Per quanto riguarda i **tool di sviluppo**, la principale considerazione è che esistono molti tool simili ma incapaci di interoperare e che non coprono tutte le attività del ciclo di vita delle ontologie (proprie del design e dell'implementazione). La mancanza di

interoperabilità tra questi tool genera importanti problemi nell'integrare un'ontologia nella libreria di ontologie di tool differenti oppure semplicemente nell'integrare due ontologie sviluppate utilizzando due programmi diversi.

Il **merging di ontologie** è un ambito non ancora abbastanza esplorato. In breve, esistono due differenti approcci alla fusione di ontologie: partendo dalle istanze o dai concetti dell'ontologia. Gli attuali tool hanno bisogno dell'intervento dell'utente per giungere al risultato definitivo della fusione, e non tengono conto di versioni precedenti di ontologie integrate. Tutti gestiscono differenti linguaggi di implementazione, e non sono in grado di integrare assiomi e/o regole.

Per i **tool di valutazione,** sono state analizzate due dimensioni principali: proprietà che valutano la qualità dell'ontologia creata (conformità al linguaggio e consistenza) e proprietà che valutano la qualità della tecnologia del tool utilizzato (interoperabilità, prestazioni, scalabilità, integrazione in framework, ecc.). La principale considerazione è che le implementazioni reali delle nuove tecnologie richiedono sforzi di testing e standardizzazione. Pertanto le comunità che si occupano di ontologie e del web semantico devono dirigere i loro sforzi di ricerca verso benchmark, certificazioni e software che adottino questi ed altri criteri per valutare le ontologie e le tecniche correlate.

Probabilmente i problemi principali delle tecnologie relative alle ontologie rispetto alle applicazioni industriali, è la mancanza di buoni strumenti di test che possano comparare differenti ontologie e tool sulla base degli stessi criteri, e l'assenza di ambienti integrati per lo sviluppo di ontologie (eccetto alcuni, quali OntoEdit, Protégé2000 o WebODE); infatti molte volte i tool sembrano moduli isolati che risolvono un determinato tipo di problema, ma non sono pienamente integrati con altre attività del ciclo di vita delle ontologie.

Di conseguenza, il lavoro futuro, dovrebbe essere guidato verso la creazione di un comune **workbench per lo sviluppo delle ontologie**, che: semplifichi la costruzione delle ontologie durante tutto il ciclo di vita, migliori la gestione delle ontologie e faciliti la creazione di tecniche avanzate per visualizzarne il contenuto, che renda agevole il loro

utilizzo, ecc. Questa area comune di lavoro dovrebbe essere supportata da un set di servizi middleware dedicati all'uso di ontologie per consentire un facile e veloce trasferimento delle informazioni. Alcuni di questi servizi possono essere: software che aiutino a localizzare le ontologie più appropriate per una specifica applicazione, metriche formali per comparare le analogie e le differenze tra i termini della stessa o di diverse ontologie, software che consentano aggiornamenti incrementali, consistenti e selettivi, accesso a sistemi di librerie di ontologie remoti, software che semplifichino l'integrazione delle ontologie con sistemi e database già esistenti ecc. Infine, l'estensione di queste tecnologie in ambito lavorativo e industriale, con il conseguente sviluppo di un gran numero di applicazioni che fanno uso di ontologie nel contesto del web semantico, consentirà la creazione di suite per lo sviluppo di applicazioni *ontology-based*, e porterà alla rapida crescita e all'integrazione di applicazioni esistenti e future basate su un'architettura a componenti.

Bibliografia

[1]. Ceusters W. Formal terminology management for language based knowledge systems: resistance is futile. In Temmerman R. (ed) Trends in Special Language and Language Technology, 2001.

[2]. Ceusters W, Martens P, Dhaen C, and Terzic B. LinkFactory: an Advanced Formal OntologyManagement System. In Proceedings of Interactive Tools for Knowledge Capture, KCAP-2001, October 20, Victoria, (http://www.isi.edu/~blythe/ kcapinteraction/papers/LinKFactoryXWhiteXPaperXfinal.doc).

[3]. Jackson B, Ceusters W. A novel approach to semantic indexing combining ontology-based semantic weights and in-document concept co-occurrences. In Baud R, Ruch P. (eds) EFMI Workshop on Natural Language Processing in Biomedical Applications, 8-9 March, 2002, Cyprus, 75-80.

[4]. Sean Bechhofer, Ian Horrocks, Carole Goble, Robert Stevens.

[5]. OILEd: a Reason-able Ontology Editor for the Semantic Web. Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

[6]. Horrocks, U. Sattler, S. Tobies. Practical reasoning for expressive description logics. 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99) (LNAI, Springer-Verlag, 1999). 161-180.

[7]. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer and D. Wenke. OntoEdit: CollaborativeOntology Engineering for the Semantic Web. In Proceedings of the International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardegna, Italia.

[8]. Y. Sure, S. Staab, J. Angele, D. Wenke and A. Maedche. OntoEdit: Guiding Ontology Developmentby Methodology and Inferencing. Submitted 2002.

[9]. Siegfried Handschuh. Ontoplugins – a flexible component framework. Technical report, University of Karlsruhe, May 2001.

[10]. Farquhar, R. Fikes, J. Rice. The Ontolingua Server: A Tool for Collaborative Ontology Construction, Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop, (Banff, Alberta, Canada 1996) 44.1-44.19.

[11]. B. Swartout, P. Ramesh, K. Knight, T. Russ, Toward Distributed Use of Large-Scale Ontologies. Symposium on Ontological Engineering of AAAI. (Stanford, California, March, 1997).

[12]. Rogers J.E., Roberts A., Solomon W.D., van der Haring E, Wroe C.J., Zanstra P.E., Rector, A.L.(2001) GALEN Ten Years On: Tasks and Supporting tools Proceedings of MEDINFO2001, V. Patel et al. (Eds) IOS Press; 256-260

[13]. Rector AL, Bechhofer SK, Goble CA, Horrocks I, Nowlan WA, Solomon WD. The GRAIL Concept Modelling Language for Medical Terminology. Artificial Intelligence in Medicine, Volume 9, 1997

[14]. N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, & M. A. Musen. Creating SemanticWeb Contents with Protege-2000. IEEE Intelligent Systems 16(2):6071, 2001.

[15]. N. F. Noy, R. W. Fergerson, & M. A. Musen. The knowledge model of Protege-2000: Combininginteroperability and flexibility. 12th International Conference on Knowledge Engineering andKnowledge Management (EKAW'2000), Juan-les-Pins, France, 2000.

[16]. M. A. Musen, R. W. Fergerson, W. E. Grosso, N. F. Noy, M. Crubezy, & J. H. Gennari. Component-Based Support for Building Knowledge-Acquisition Systems. Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress WCC 2000), Beijing, 2000.

[17]. Grosso, W., Gennari, J.H., Fergerson, R. and Musen, M.A. (1998). When Knowledge Models Collide (How it Happens and What to Do). In: Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Bases Systems Workshop, Banff, Canada.

[18]. Missikoff M., Velardi P., Navigli R.: The Usable Ontology: An Environment for Building andAssessing a Domain Ontology, Proceedings of the International Semantic Web Conference 2002 (ISWC2002), June 2002, Sardegna Italia.

[19]. Arpírez, J.C.; Corcho, O.; Fernández-López, M.; Gómez-Pérez, A. WebODE: a Scalable Workbench for Ontological Engineering. First International Conference on Knowledge Capture (KCAP01). Victoria. Canada. October, 2001.

[20]. Fernández-López M, Gómez-Pérez A, Pazos A, Pazos J. Building a Chemical Ontology Using Methontology and the Ontology Design Environment. IEEE Intelligent Systems & their applications.January/February 1999.

[21]. Gómez-Pérez, A. A proposal of infrastructural needs on the framework of the semantic web for ontology construction and use. Programme Consultation Meeting (PCM-9) on Knowledge Technologies. European Commission. April, 2001.

[22]. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick; CLASSIC: A Structural Data Model for Objects, SIGMOD, Oregon, 1989

[23]. V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice; OKBC: A Programmatic Foundation for Knowledge Base Interoperability; AAAI-98.

[24]. Farquhar, R. Fikes, and J. Rice; The Ontolingua Server: a Tool for Collaborative Ontology Construction; Intl. Journal of Human-Computer Studies 46, 1997. Intraspect Knowledge Server, Intraspect Corp., 1999. (http://www.intraspect.com/product_info_solution.htm)

[25]. Y. Iwasaki, A. Farquhar, R. Fikes, & J. Rice; A Web-based Compositional Modeling System for Sharing of Physical Knowledge. Morgan Kaufmann, Nagoya, Japan, 1997.

[26]. D.L. McGuinness; Ontological Issues for Knowledge-Enhanced Search;

Proceedings of Formal Ontology in Information Systems, June 1998. Also in Frontiers in Artificial Intelligence and Applications, IOS-Press,Washington, DC, 1998.

[27]. D.L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An Environment for Merging and Testing Large Ontologies. Proceedings of Knowledge Representation 2000.

[28]. D.L. McGuinness, L.A. Resnick, and C. Isbell; Description Logic in Practice: A CLASSIC: Application; IJCAI, 1995.

[29]. United Nations Standard Product and Services Classification (UNSPSC) Code organization.http://www.unspsc.org/home.htm

[30]. Welty; An HTML Interface for CLASSIC; Proceedings of the 1996 International Workshop on Description Logics; AAAI Press; November, 1996.

[31]. Noy, N.F. and Musen, M.A. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Seventeenth National Conference on Artificial Intelligence (AAAI-2000). Austin, TX, 2000.

[32]. Noy, N.F. and Musen, M.A. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. In: Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001). Seattle, WA, 2001.

[33]. (de Diego, 2001) de Diego, R. Método de mezcla de catálogos electrónicos. Facoltà di Informatica dell'Università Politecnica di Madrid. Spagna. 2001.

[34]. (Ramos, 2001) Ramos, J.A. Mezcla automática de ontologías y catálogos electrónicos. Facoltà di Informatica dell'Università Politecnica di Madrid. Spagna. 2001.

[35]. T. Berners-Lee, J. Hendler and O. Lassila. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American, 2002, cf.

[36]. http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html.

[37]. EU IST-1999-10132 project "On-To-Knowledge: Content-driven knowledge management tools through evolving ontologies", cf. http://www.ontoknowledge.org.

[38]. EU IST-2000-29243 thematic network "OntoWeb: Ontology-based Information Exchange for Knowledge Management and Electronic Commerce", cf. http://www.ontoweb.org.

[39]. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer and D. Wenke. OntoEdit: Collaborative Ontology Engineering for the Semantic Web. In Proceedings of the International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardegna, Italia.

Appendice A

PROLOG

Il Prolog (PRO*gramming in* LOG*ic*) è stato sviluppato all'Università di Marsiglia da Alain Colmerauer all'inizio degli anni 1970, come strumento di programmazione basato sulla logica, o più precisamente su un sottoinsieme del calcolo dei predicati. Il Prolog è un linguaggio dichiarativo, in esso non vi è una sequenza di azioni, ma una raccolta di predicati (o fatti) con delle regole: in terminologia logica i predicati sono proposizioni, proposizioni con variabili ed enunciati (con valore di verità "vero") e le regole sono implicazioni; ad esempio la frase se piove la strada è bagnata è una regola che lega il predicato piove con quello di bagnato. Tradotto nella sintassi Prolog ciò diventa:

piove.
bagnato:-piove.

L'esecuzione del programma è una richiesta a Prolog di dedurre dei fatti dall'insieme di informazioni che Prolog conosce, cioè l'insieme di predicati e di regole comunicate al programma; nell'esempio precedente vi è un predicato "*piove*" ed una regola "*è bagnato se piove*"; una possibile richiesta è:
goal bagnato.
La risposta di Prolog è:
yes

Mentre nel calcolo proposizionale si può dedurre unicamente il valore di verità di una proposizione, come nell'esempio visto, Prolog , è in grado di dedurre, se la domanda è un predicato contenente variabili, i valori che soddisfano il predicato. Il cuore di Prolog è un motore di inferenza, cioè un (sotto)programma cha può "ragionare logicamente" sulle informazioni. Prolog cerca di inferire la verità di un'ipotesi (la domanda che è stata posta), interrogando la collezione di informazioni già note che sono dunque vere. La risposta comprende tutte le soluzioni possibili.

Fatti e Regole

I *predicati* possono esprimere sia proprietà degli oggetti che relazioni; in linguaggio naturale alcune proprietà possono essere "l'erba è verde" e "Maria è una ragazza", mentre sono relazioni: "a Mario piace guidare", "Parigi è la capitale della Francia".

L'assunzione basilare è che i *predicati* sono ciò che è conosciuto; implicitamente ciò che è conosciuto è vero.

Nella sintassi di Prolog un predicato si descrive con un nome di predicato seguito fra parentesi dall'oggetto o dagli oggetti su cui agisce; il tutto chiuso con un punto:

```
tipo(ferrari,sportiva).
tipo(maserati,sportiva).
tipo(fiat,familiare).
tipo(renault,utilitaria).
piace(marco,auto).
piace(elena,auto).
piace(anna,bici).
```

Le *regole* sono predicati in cui un fatto è "vero" se uno o più altri fatti sono veri. La regola che definisce quando "piace guidare a Mario" potrebbe essere la seguente: "a Mario piace guidare se l'automobile è sportiva". Quindi le *regole* sono formate da due parti, la prima

(*testa*) è un predicato che è reso vero dalla seconda (*corpo*), separata dalla prima tramite i simboli "**:-**" (due punti meno col significato di *se*):

```
piace(anna,auto):-tipo(Automobile,familiare);tipo(Automobile,utilitaria).
piace(mario,auto):-tipo(Automobile,sportiva),Automobile <> "ferrari".
```

La prima regola afferma che ad anna piace l'auto se e' familiare o utilitaria, l'alternativa è indicata in Prolog col segno "**;**" (l'AND con il segno ","). Per la seconda regola, a mario piacciono le auto sportive purché non siano ferrari. In minuscolo si indicano i fatti conosciuti, mentre le variabili hanno un nome che inizia con una lettera maiuscola, come la variabile Automobile che compare nelle regole. Le regole permettono la deduzione di fatti da altri fatti, ma esse possono essere pensate come procedure per chiedere a Prolog di compiere azioni diverse dal provare dei fatti, quali la scrittura di qualcosa o la creazione di un archivio, e in genere, ciò che un linguaggio di programmazione tipicamente può fare.

Domande

Una volta fornito a Prolog un insieme di predicati, gli si possono porre domande che riguardano tali predicati; dagli esempi del paragrafo precedente, alcune domande potrebbero essere:

Domanda in linguaggio naturale	Domanda Prolog	Risposta

Cosa piace ad Anna?	`piace(anna,Cosa)`	`Cosa=bici`
		`Cosa=auto`
		`Cosa=auto`
		`3 Solutions`

Cosa piace a Chi?	`piace(Chi,Cosa)`	`Chi=antonio, Cosa=auto` `Chi=carla, Cosa=auto`

		Chi=anna, Cosa=bici Chi=anna, Cosa=auto Chi=anna, Cosa=auto Chi=mario, Cosa=auto 6 Solutions
A Mario piacciono le auto?	piace(mario,auto)	Yes
A Mario piacciono le bici?	Piace(mario,bici)	No
C'è qualcuno a cui piace la bici?	piace(_,bici)	Yes
Esiste un catorcio di macchina?	tipo(Macchina,catorcio)	No Solution

Prolog per rispondere ad una domanda esamina i predicati che conosce: il caso più semplice è quello in cui la domanda non contiene variabili, come in `piace(mario,auto)`, in questo caso Prolog cerca se esiste il predicato `piace(mario,auto)` o una regola la cui testa sia il predicato. Se non trova nulla la risposta è `no`, altrimenti, se esiste il predicato la risposta è `yes`; se esiste una regola e la coda di essa risulta vera, la risposta è `yes`, altrimenti è `no`.

Se la domanda contiene variabili, Prolog sostituisce alle variabili, il valore che ricava dai fatti, la risposta invece di essere `yes` o `no` è l'elenco degli attributi che soddisfano le variabili oppure: `No Solution`. Alla domanda `piace(anna,Cosa)`, Prolog ha risposto due volte `Cosa=auto`, ciò deriva dalla caratteristica di Prolog di cercare tutte le soluzioni, infatti la regola:

piace(anna,auto):-tipo(Automobile,familiare);tipo(Automobile,utilitaria).

é soddisfatta due volte, in quanto tra i fatti ci sono una automobile *familiare* ed una automobile *utilitaria*.

Il segno "_" indica una variabile *anonima* per rispondere a domande generiche.

Per rispondere Prolog esamina tutti i fatti, tranne quando la domanda è senza variabili, per cui al primo predicato verificato è in grado di rispondere `yes`, e quindi può interrompere la ricerca.

In Prolog le variabili sono gestite dal motore di inferenza, esse nascono libere (*free*), e nel

corso dell'elaborazione Prolog assegna loro dei valori che ricava dai fatti (*bound*), per stabilire se soddisfano la richiesta; finita l'elaborazione esse ridiventano libere. In un predicato possono coesistere variabili *bound* o di input e variabili *free* o di output. L'elemento che più si avvicina al concetto di variabile dei linguaggi di programmazione tradizionali, è il *fatto*, ed i *fatti* si possono aggiungere alla collezione di fatti e regole tramite un apposito predicato (`assert`).

Un predicato può produrre diverse soluzioni, in funzione dello stato delle variabili con cui è richiamato: ad esempio il predicato `str_int(Intero,Stringa)` potrà convertire un numero in una stringa, convertire una stringa numerica in numero o verificare che il numero e la stringa numerica convertita in numero siano uguali, secondo la tabella seguente:

`Intero`	`Stringa`	
bound	bound	Confronto
bound	Free	converte in stringa numerica
free	bound	converte in numero
free	Free	Errore

In generale se tutte le variabili sono *bound*, il predicato è vero o falso, altrimenti risponde con il valore o i valori che lo rendono vero.

PROLOG come linguaggio procedurale

La caratteristica di Prolog di cercare regole o fatti che soddisfino la domanda, è la base del suo utilizzo procedurale, in particolare se la domanda è relativa ad una regola plurima, cioè stessa testa e corpi diversi, Prolog verificherà la prima regola (quindi l'ordine delle regole è talvolta essenziale) se essa non è verificata, procederà con la regola successiva, ad esempio:

```
ciclo(10):-!.                                   /*   variabile = 10 termina      */
ciclo(Counter):-NewCounter=Counter + 1,      /*  aggiorno contatore di ciclo  */
        random(Rnd),                            /* genero un numero a caso       */
        write(NewCounter,"\t",Rnd),nl,
        ciclo(NewCounter).                   /* riciclo                  */

find:-write("Inizio della generazione di numeri casuali\n"),fail.
find:-ciclo(0),fail.
find:-write("Fine della generazione di numeri casuali\n"),exit.

goal find.
```

La domanda è `find`, Prolog verifica la prima regola `find`, in questa il predicato `write` produce una scritta sul video e risulta non verificato, a causa del predicato `fail`. Il predicato `fail` ha lo scopo forzare Prolog a proseguire e verifica quindi la seconda regola `find` e poi finalmente la terza.
La seconda `find` verifica il predicato `ciclo` che genera 10 numeri casuali, infatti la prima regola `ciclo` termina il ciclo, la seconda genera un numero a caso, tramite il predicato `random`, incrementa il contatore del ciclo, scrive il numero casuale ed infine ripete il ciclo.

La regola `ciclo(10):-!.` termina il ciclo tramite il predicato `!` (*cut*), il cui scopo è di terminare la verifica della regola. Il predicato `!` è utilizzato per lo scopo appena visto e quando interessa conoscere se c'è una soluzione, evitando così la perdita di tempo per trovarle tutte.
Il predicato `exit` evita la visualizzazione della risposta di Prolog . Infine il predicato `nl` permette l'avanzamento di una riga (in alternativa ai caratteri "`\n`" nel predicato `write`).

PROLOG e la conoscenza implicita

Nelle regole e nei fatti talvolta è presente della "conoscenza implicita": essa è *nel significato del predicato*, ad esempio dati i fatti `prima(a,b)` e `prima(b,c)`, la conoscenza implicita è `prima(a,c)` perché `prima` è una relazione di ordine, o *nelle relazioni tra fatti diversi*, ad esempio dai fatti e dalle regole evidenziati nei paragrafi precedenti si capisce che a *Mario piacciono le maserati*, e a *Elena piacciono tutte le automobili*. Ma non c'è una regola o un fatto che dica qualcosa del genere: se a Tizio piacciono le Automobili, la Marca dell'automobile preferita non è specificata. Prolog permette di generare dei fatti, tramite il predicato `assert(...)`, quindi potrebbe generare il fatto `marca_preferita(persona,marca)`, e poi elencarlo.

Le regole relative ad Anna e Carlo fanno riferimento al tipo di auto, quindi se la regola è verificata (Prolog ha trovato la Marca dell'automobile) si possono modificare le due precedenti regole, come segue:

```
piace(anna,auto):-  tipo(Automobile,familiare),
                                assert(marca_preferita(anna,Automobile));
tipo(Automobile,utilitaria),
            assert(marca_preferita(anna,Automobile)).

piace(carlo,auto):- tipo(Automobile,sportiva),Automobile <> "ferrari",
            assert(marca_preferita(carlo,Automobile)).
```

Questo vale solamente per Anna e Carlo, per tutti gli altri occorre aggiungere (in coda) un'ulteriore regola:

```
piace(X,auto):-tipo(Auto,_),X <> "anna",X <> "carlo",
        assert(marca_preferita(X,Auto)).
```

Infine occorre scrivere le regole "procedurali" per eseguire il tutto, un esempio è il seguente:

```
elenco_marche(X):- piace(X,auto),fail.
elenco_marche(_):- marca_preferita(X,Y),
        write(X," gradisce la ",Y),nl,fail.
elenco_marche(_):- write("--------------"),nl,exit.
```

goal elenco_marche(anna).

La risposta di Prolog è:

anna gradisce la fiat

anna gradisce la renault

La prima regola `elenco_marche` genera tutte le preferenze (di Anna), e la seconda le stampa.

La potenza di Prolog è nella capacità di estrarre la "conoscenza implicita" che è presente nei fatti, mediante opportuni predicati; questi in sostanza formalizzano le relazioni esistenti fra i fatti, ad esempio le relazioni d'ordine che il predicato *prima* (A è prima di B) sottende. L'esempio che segue propone un programma che verifica l'esistenza di cicli in un insieme di attività, di cui sono date le precedenze tramite il predicato `prima()`. I fatti sono:

```
prima(ideazione,pianificazione).
prima(pianificazione,pubblicità).
prima(ideazione,"ricerca materiale").
prima("ricerca materiale",stesura).
prima(pubblicità,ideazione).     // introduce un ciclo
prima(pianificazione,stesura).
prima(stesura,correzione).
prima(correzione,stampa).
prima(pubblicità,distribuzione).
prima(stampa,distribuzione).
```

I predicati di ricerca dei cicli sono:

```
trovacicli(X):-attivita(X,Y,[]).
attivita(Prima,Dopo,ListaLavori):-prima(Prima,Dopo),
                    not(member(Dopo,ListaLavori)),
                    prima(Dopo,NextDopo),
                    attivita(Dopo,NextDopo,[Dopo|ListaLavori]).
attivita(_,Dopo,ListaLavori):-member(Dopo,ListaLavori),
                write("Ciclo: "),nl,
                writelist([Dopo|ListaLavori]),fail.
attivita(_,_,[]):-write("--- Fine ---"),nl,
          exit.
```

Il predicato `trovacicli` è utilizzato per la ricerca, il suo oggetto è una attività di partenza; il primo dei predicati `attivita` "scorre" la sequenza di attività; se una attività è già stata trovata (predicato `member`) allora il predicato fallisce, altrimenti prosegue la ricerca inserendo l'attività in una lista di attività.

Il secondo predicato `attivita` agisce dopo il primo, ed ha come oggetti le variabili `Dopo` e `ListaLavori` ereditate dal primo predicato `attivita`, in particolare se questo è fallito per il predicato `not(member(Dopo,ListaLavori))`, `dopo` conterrà l'attività che provoca il ciclo, e `ListaLavori` l'elenco delle attività che compongono il ciclo, e queste verranno stampate.

Alla domanda:

GOAL trovacicli(ideazione).

Prolog risponde:

Ciclo:

pianificazione

ideazione

pubblicità

pianificazione

--- Fine ---

Alla domanda:

GOAL trovacicli(correzione).

Prolog, non trovando cicli dopo l'attività "correzione", risponde:

--- Fine ---

In questo programma si è utilizzata una lista per elencare le attività. Le liste sono una struttura dati che può essere usata ricorsivamente sfruttando la definizione:

Lista = [Testa o primo elemento della lista | Coda o resto della lista]

L' operatore " | " permette sia la costruzione delle liste, sia il loro trattamento, un esempio del primo caso è la clausola `writelist([Dopo|ListaLavori])`: l'oggetto di `writelist` è la lista formata dall'elemento `Dopo`, seguita dalla lista `ListaLavori`. Sfortunatamente ci sono pochi predicati nativi per trattare le liste, è necessario scriverli, o utilizzare delle definizioni da inserire tramite il comando Prolog `include`, come è stato fatto nell'esempio precedente con i due predicati:

```
member(Testa, [Testa|_]).
member(Testa, [_|Coda]):-member(Testa,Coda).
writelist([]):-!.
writelist([Testa|Coda]):-write(Testa),nl,writelist(Coda).
```

Questi chiariscono come trattare ricorsivamente le liste, ad esempio il predicato `member` verifica la presenza di un elemento in una lista: se l'elemento è il primo, viene verificato il primo predicato `member`, altrimenti la ricerca prosegue, ricorsivamente, sul resto della lista.