UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA
Facoltà di Ingegneria - Sede di Modena
Corso di Laurea in Ingegneria Informatica

DINAMICA DELLE ONTOLOGIE:
INSERIMENTO DI UNA NUOVA SORGENTE NEL SISTEMA MOMIS

Stefania Bruschi

Tesi di Laurea
Anno Accademico 2003/2004

Relatore: Chiar.mo Prof. Sonia Bergamaschi
Controrelatore: Chiar.mo Prof. Federica Mandreoli
Correlatore: Dott. Ing. Francesco Guerra

Ai miei genitori
e ad Ilaria

"Non esistono condizioni ideali in cui scrivere, studiare, lavorare o riflettere, ma è solo la volontà, la passione e la testardaggine a spingere un uomo a perseguire il proprio progetto."

Konrad Lorentz

Parole Chiave:
Ontology
Dynamics
Semantic Web
MOMIS
SEWASIE

INDICE

INDICE DELLE FIGURE

INDICE DELLE TABELLE

INTRODUZIONE

La gestione di grandi quantità di informazioni, è di crescente importanza nella realtà odierna, si pensi ad esempio al Web o alla sempre più massiccia presenza di grandi organizzazioni dotate dei loro sistemi informativi in cui sempre più si sente la necessità di ottenere risultati più mirati e meno ridondanti come esito di ricerche.

Questo problema e' stato affrontato con l'introduzione del concetto di "Web semantico" da Tim Berners Lee, in cui i documenti vengono richiamati utilizzando anche la semantica dell'informazione contenuta, questo ha comportato lo sviluppo di nuovi strumenti, tra i quali le ontologie. Queste propongono una base per ovviare alle difficoltà riguardati sia l'accesso ai dati che la loro gestione, tuttavia il loro utilizzo richiede:

- ✓ un tempestivo adattamento ai cambiamenti che possono derivare dallo sviluppo nel dominio rappresentato, da errori durante la fase di modellazione, dall'insorgere di nuovi obiettivi o da altre motivazioni;
- ✓ una coerente gestione e diffusione di questi cambiamenti, poiché la modifica in una parte dell'ontologia può generare delle inconsistenze in altre parti della stessa e nelle istanze di base.

Uno degli scopi di questa tesi sarà, quindi, quello di affrontare queste problematiche che concernono la "dinamica di una ontologia", saranno quindi presentati alcuni approcci studiati in letteratura.

Nell'ambito del progetto **SEWASIE**, all'interno del sistema **MOMIS**, framework sviluppato dal DBGROUP del Dipartimento di Ingegneria

dell'Informazione dell'Università di Modena e Reggio Emilia riguardante l'estrazione e l'integrazione di informazioni per sorgenti dati strutturate e semistrutturate, verrà in modo più specifico affrontata una delle dinamiche ancora relativa all'integrazione di una nuova sorgente in una Global Virtual View già formata con l'obiettivo di sviluppare un algoritmo che aggiorni la struttura dell'ontologia rispettandone i requisiti.

La tesi sarà strutturata nel seguente modo:

Nel primo capitolo verrà descritto l'ambiente di sviluppo cominciando con un accenno a come sia nata l'idea del Semantic Web, successivamente si evidenzieranno gli obiettivi e le caratteristiche più importanti del progetto SEWASIE e del sistema MOMIS.

Nel secondo capitolo verrà introdotto il concetto di ontologia, l'origine del termine, e come questa sia stata assorbita nella comunità di ricerca dei sistemi informativi. Verranno poi proposte diverse tipologie di classificazione delle stesse ontologie. Il capitolo si concluderà con una più dettagliata panoramica su quale sia il processo di costruzione di uno schema ontologico in MOMIS.

Nel terzo capitolo si articolerà uno studio su quali siano gli scenari possibili in letteratura quando si parla di "dinamica di una ontologia". Partendo dai due possibili approcci, già introdotti nella tesi di laurea svolta da Alein Fergnani [32], verranno come prima cosa esaminate le diverse tipologie di evoluzione di una ontologia, successivamente verranno esposti progetti tratti dalla letteratura, riguardanti alcune tematiche principali legate all'evoluzione come l'eliminazione di concetti ritenuti irrilevanti o obsoleti e l'integrazione di più ontologie.

Infine il quarto capitolo sarà dedicato all'applicazione realizzata in ambiente MOMIS, che implementa l'algoritmo per l'integrazione di una o più sorgenti sfruttando quello cioè che viene definito una comparazione o confronto di due ontologie.

Capitolo 1

IL PROGETTO SEWASIE E IL SISTEMA MOMIS

1.1 Web Semantico

1.1.1 I primi passi

Se dovessimo dare una definizione semplice, ad un non addetto al settore, di che cos'è Internet, comunemente chiamato Web, la prima che ci verrebbe in mente è "Internet: un insieme di testi, un insieme molto vasto di documenti che descrivono dei informazioni". Per la verità questa non è una grossa novità, collezioni di testi sono esistiti fin dall'antichità (biblioteche). La novità di Internet sta nel fatto che questi testi possono richiamarsi l'uno con l'altro, in modo rapido e semplice. Il link è l'elemento nuovo che l'HTML ha saputo proporre.

Ai suoi esordi Internet era costituito unicamente di testi e indici ipertestuali a questi, successivamente, le cose si sono evolute con l'avvento dei motori di ricerca per accedere direttamene ai contenuti dei testi e la strutturazione multilivello dei siti: database, script cgi, fogli di stile. Ciò che è rimasto sempre presente è un alto grado di trasparenza nei confronti dell'utente, questo significa l'architettura interna rimane quasi del tutto nascosta all'utente. Il vantaggio del Web infatti è che può essere utilizzato da tutti, l'utente si orienta durante la ricerca grazie a due cose: la sua esperienza di navigazione e la capacità di utilizzare al meglio parole o espressioni chiave. L'esperienza è un aspetto molto importante di cui tutti ci serviamo, impariamo che determinati contenuti si possono reperire sotto determinate tipologie

di siti, ad esempio portali, impariamo che l'aspetto grafico di un sito può dirci qualche cosa sul genere (formale o informale) delle informazioni. Tuttavia l'esperienza è un attributo che si crea da solo non è legato ad aspetti tecnici, al codice e alle applicazioni che costituiscono un sito. L'altro aspetto, invece, quello delle parole chiave, è più legato al codice.

1.1.2 Legami semantici

La capacità espressiva di un link dipende dalla applicazione che lo gestisce. Un primo approccio è quello di inserire in un motore di ricerca una certa espressione nella convinzione che questa sia in grado di individuare nel modo più efficace possibile il contenuto cercato, ma l'efficacia dell'operazione dipende in larga misura dagli algoritmi che il motore di ricerca utilizza per estrarre i contenuti. Un altro approccio si può essere di scegliere tra le voci di una barra di navigazione, in questo caso l'utente dovrà stabilire quale espressione si adatti meglio a individuare, come titolo generico, un contenuto, l'efficacia dipende da chi ha progettato i contenuti del sito e da quanto si prestino ad essere indicizzati secondo una gerarchia ad albero.

In entrambe le situazioni l'utente si affida ad una espressione unica che ha un rapporto molto generico col contenuto effettivamente ricercato, infatti nel caso del motore di ricerca qualsiasi query attivata è sempre soggetta al rischio della ambiguità, ad esempio la parola "albero" potrei trovare contenuti legati all'informatica alla botanica alla nautica. Allo stesso modo nel caso di una barra di navigazione, la genericità è data dal tipico meccanismo dell'indice, che sotto un unico titolo deve raccogliere un gruppo spesso vario di contenuti.

La morale che si può trarre è che Internet è sì un insieme di testi collegati tra loro, ma soprattutto quelli legati alla capacità di descriverne il significato sono deboli, nel senso che sono troppo generici e vaghi. Per ovviare a questo un collegamento oltre a portare in un determinato luogo dovrebbe descrivere il luogo verso cui porta; la parola giusta per definire questa funzione è "capacità semantica" (semantico è un meccanismo che sa predire, il valore della sua azione).

Se ci si pensa bene però, si usano abitualmente strutture con un valore semantico, anche se minimo, infatti, in fin dei conti un codice è una serie regole

sintattiche, non ha altra capacità che seguire i passi di una regola descritta dal programmatore.

Si pensi ad esempio ad un archivio bibliografico online, organizzato con un database con tre classi diverse: autori, articoli e libri; posso permettere all'utente di cercare tra gli autori o solo quelli che hanno pubblicato articoli, oppure solo quelli che hanno pubblicato libri. In questo ultimo caso l'utente non cerca informazioni utilizzando come chiave un unico concetto (autore) e nemmeno due concetti affiancati (es. autore + libro), cadendo così nelle imprecisioni del motore di ricerca, l'utente può, quindi, formulare una precisa relazione tra un autore e il tipo della sua pubblicazione. Questo è dovuto al fatto che chi ha progettato il sito ha stabilito una struttura per organizzarla: da una parte gli autori, da una parte le loro opere, dividendole in due categorie, i libri e gli articoli. Nel contesto di quella applicazione succede perciò che un autore possa essere individuato esprimendo il vincolo che abbia o meno pubblicato articoli, si conosce inoltre che se non ha pubblicato articoli allora ha pubblicato libri. L'inserimento di uno schema per archiviare quelle informazioni produce quindi un notevole vantaggio diminuendo la genericità dei collegamenti. Si pensi ad uno schema XML, esso è formato da un insieme di regole che stabiliscono come debbano essere organizzate le informazioni in esso raccolte, ma anche definisce le relazioni tra i dati, esso, quindi, è anche in grado di esprimere vincoli che legano o oppongono due tipi di dati.

1.1.3 La proposta del Web Semantico

Il web semantico[1] nasce quindi dall'idea di utilizzare schemi per descrivere domini di informazione, si comincia a parlare, quindi, di "metadato"[1], una struttura in grado di descrivere e automatizzare i collegamenti esistenti fra i dati. Il web semantico è quindi composto da tre livelli fondamentali: al livello più basso appaiono i dati, poi i metadati che riportano questi ai concetti di uno schema, infine lo schema,

[1] Metadato, dal greco "metà" (oltre, al di là) e dal latino "datum", dato su un altro dato, riportano una descrizione standardizzata di risorse documentali o informative, in ordine al loro reperimento e utilizzo. In ambito Web servono a descrivere pagine, documenti, immagini, allo scopo di migliorare la navigazione e il recupero dei dati.

o ontologia, in cui si esprimono le relazioni fra concetti, che diventano classi. Quando si parla di web semantico si intende proporre 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 Berners Lee, da allora è stato associato all'idea di un web nel quale agiscano agenti intelligenti: applicazioni in grado di comprendere il significato dei testi presenti sulla rete e perciò in grado di guidare l'utente direttamente verso l'informazione ricercata, oppure di sostituirsi a lui nello svolgimento di alcune operazioni. Un agente intelligente dovrebbe essere una applicazione in grado di svolgere operazioni non banali in modo autonomo, ad esempio cercare una prenotazione di un aereo per Parigi con arrivo in centro città prima delle 13.00, il tutto analizzando informazioni da siti che definiscono l'aeroporto di Parigi in modo diverso (Paris, Charles de Gaule, Orly) e deducendo, senza che sia specificato nella query, che un arrivo per le 13.00 in centro implichi un arrivo in aeroporto diverso a seconda dell'aeroporto effettivamente selezionato.

Questa proposta ha affascinato molto la comunità informatica; il W3C ha attivato immediatamente un gruppo di lavoro, le università hanno aperto numerosi programmi di ricerca legati a questi temi; si sono imposti subito degli standard, il più famoso dei quali è certamente RDFs (un linguaggio in sintassi XML per definire e esprimere ontologie); rientra tra questi programmi di ricerca il progetto SEWASIE.

1.2 SEWASIE

1.2.1 Presentazione

SEWASIE [2], acronimo di SEmantic Webs and AgentS in Integrated Economies, è un progetto di ricerca finanziato dalla comunità europea nell'azione IST (Information Society and Technology), iniziato nel maggio 2002 e che si concluderà nel aprile di questo anno (2005). Il progetto coordinato dall'Università degli Studi di Modena e Reggio Emilia si avvale della collaborazione, in qualità di partner, di alcune affermate realtà scientifiche ed accademiche: Confederazione Nazionale dell'Artigianato e della Piccola e Media Impresa, CNA Servizi Modena s.c.a.r.l. (Italia), Università degli Studi di Roma "La Sapienza" (Italia), Rhenisch Westfaelische Technische Hochschule

Aachen (Germania), The Victoria University of Manchester (Regno Unito), Thinking Networks AG (Germania), IBM Italia S.p.A (Italia), Fraunhofer-gesellschaft zur Forderung der angewandten Forschung eingetragener Verein (Germania).

> ***SEWASIE*** *progetta e sviluppa un motore di ricerca, basato sulla semantica, che fornisca un accesso intelligente a sorgenti di dati eterogenee residenti su web, fornendo le basi per una sicura struttura di comunicazione basata sul web.*

Gli obiettivi sono di sviluppare un'architettura distribuita per la ricerca e la comunicazione semantica su una ontologia multilinguaggio specifiche per l'applicazione, implementando il concetto di agente; fornire alle ontologie un livello di inferenza basato sugli standards W3C[3] (XML, XMLS, RDF, RDFS, OWL); creare un insieme di applicazioni e infrastrutture che siano stabili, efficaci e pronte all'uso da parte delle PMI (Piccole Medie Imprese) nel contesto dell'UE; infine ottenere una documentata esperienza reale.

Misurare i benefici derivanti dalla realizzazione di questo progetto solo in termini di minor perdita di tempo e costi di ricerca delle informazioni è limitativo, esso offre, infatti, nel vasto mondo competitivo delle imprese, il vantaggio di ottenere un'informazione, aggiornata, completa e comparativa, tali da supportare adeguatamente l'attività di negoziazione richieste. In particolare è destinato a promuovere l'utilizzo di nuove tecnologie nei crescenti sistemi delle piccole e medie imprese, accrescendone così il loro livello di competitività e di presenza sul mercato.

1.2.2 Applicazioni commerciali

Gran parte del tessuto industriale europeo è costituito da piccole e medie imprese (PMI) agricole, industriali, commerciali e di servizi. Per ragioni sia di ordine storico che sociali, si è diffusa la politica dell'aggregazioni industriale, con lo sviluppo di aree di settori (in letteratura economica chiama "distretti industriali"). Purtroppo con l'avvento della globalizzazione, questo tipo di organizzazione economica risulta essere vulnerabile ai forti cambiamenti, e ancor di più lo sarà in avvenire. Ciò che differenzia successo e fallimento è la capacità che ha un'azienda di reperire ciò di cui ha bisogno (un fornitore sufficiente, un metodo di lavoro innovatore, un nuovo

mercato ed e così via) ed essere facilmente trovata (dai clienti, da soci o dai garanti possibili).

Gli strumenti forniti da Internet (come visto nel paragrafo precedente) sono inadeguati poiché difficili da sviluppare al meglio (le piccole imprese non hanno spesso le infrastrutture tecnologiche, la conoscenza e, soprattutto, l'esperienza necessaria), ad esempio fornendo a semplici domande troppe pagine e collegamenti non necessari. Si supponga che una ditta debba acquisire maggiori informazioni riguardanti un oggetto, generalmente dovrà fare ripetute ricerche con l'impiego di maggior tempo e quindi aumentati costi. Una tipica ricerca potrebbe svolgersi così: lo scopo è quello di trovare un nuovo processo di tinteggiatura di tessuti, per prima cosa si effettuerà una ricerca di chi fornisce questo prodotto usando il termine "tintura tessuti" e il motore di ricerca elencherà tra i collegamenti (151 nel caso di www.google.it) non solo i fornitori di apparecchiatura per la tintura del tessuto, ma anche informazioni inutili come la storia e la tecnologia della tinteggiatura. Dopo aver individuato un fornitore possibile, guardando tra i molti risultati, l'azienda dovrà controllare se esistono delle norme per lo smaltimento dei rifiuti tossici e il riciclaggio, i criteri di ricerca possono essere vari, ad esempio, il numero della legge, il termine "legge", l'interpretazione ecc. Entrambe le ricerche diventano più difficoltose quando si cerca anche su sorgenti di informazioni estere, dove possono esistere terminologie specifiche non conosciute.

Un ulteriore esempio, per coloro che necessitano escusivamente della parte normativa, in questo caso la ricerca non si limita a reperire velocemente le norme correlate all'argomento associato nella ricerca, ma deve mostrare anche i riferimenti, i collegamenti ad argomenti correlati e così via. Per questi motivi, gli utenti devono avere a loro disposizione un motore di ricerca con un'interrogazione easy-to-use[2], in grado di raccogliere e mostrare le informazioni presenti in Internet in un formato facilmente fruibile. Nel caso di metodo step-by-step, i passaggi eseguiti devo essere chiari e rispettare le caratteristiche appena citate.

[2] Utilizzabile in modo semplice da non addetti al settore

Il requisito principale è quindi quello di ottenere un risultato strutturato, che a domanda generica posta dall'utente, interpreti le risposte e fornisca elementi filtrati sulla base di criteri definiti dai progettisti e dall'"esperienza acquisita".

1.2.3 SEWASIE Virtual Network (SVN)

I tools e i metodi del progetto SEWASIE creano e mantengono le ontologie multilingue, con un livello di inferenza basato sugli standards W3C (XML, XML schema, RDF(S)), il cui scopo è "lead the Web to its full potential" ovvero portare il Web al suo massimo potenziale, mediante lo sviluppo di tecnologie (specifiche, linee guida, software e tools) che possano creare un forum per le informazioni, il commercio, le ispirazioni, il pensiero indipendente e la comprensione collettiva.

L'idea del progetto SEWASIE è quella di fornire un'architettura aperta e distribuita basata sul concetto di agenti mobili e intelligenti (brokers, mediators, e wrappers, che definirò nei prossimi capitoli) con una conseguente maggiore flessibilità e scalabilità, cioè la capacità di adattarsi ai cambiamenti, al crescere dell'ambiente e di interagire con altri sistemi, offrendo un punto centrale di accesso all'utente.

E' stata quindi implementata una rete virtuale (SVN) per affiancare l'utente nella ricerca di informazioni tra sorgenti di dati eterogenei presenti nel web. In particolare essa inoltra le richieste fatte dall'utente ad un agente, che analizza e risponde alla domanda movendosi attraverso i nodi di informazione (Sewasie Information Node o SINode) per cercare le informazioni richieste. I SINode sono componenti indipendenti che arricchiscono di informazioni semantiche le sorgenti, collegando i dati ad ontologie ed ad altri metadati.

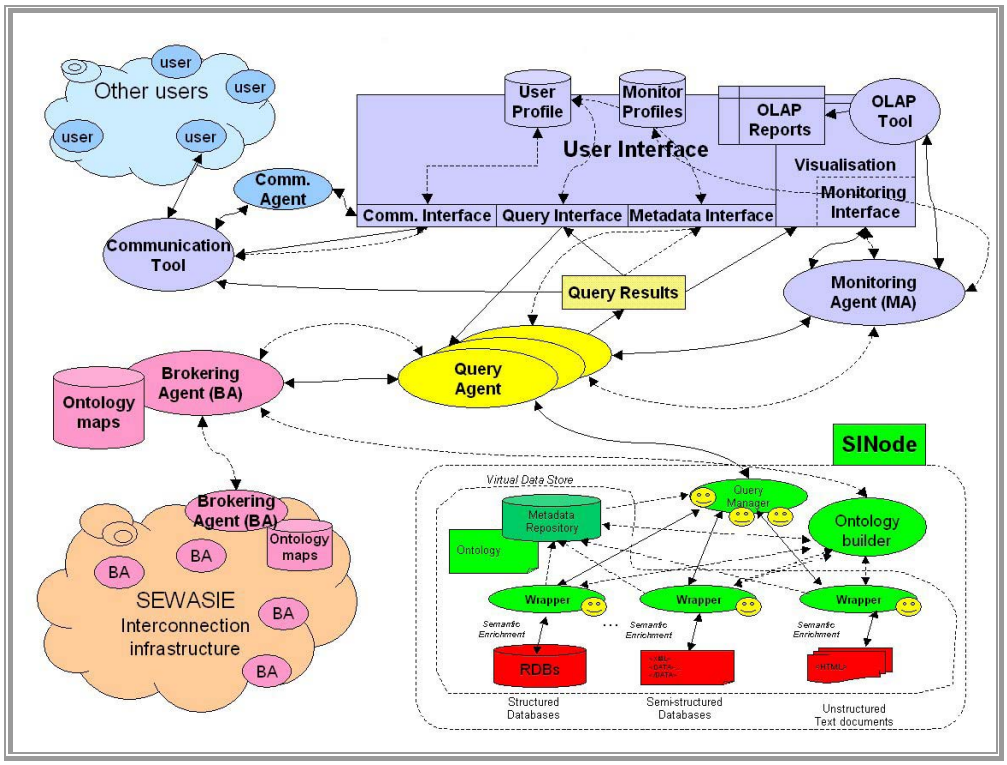

Figura 1.1: Architettura distribuita di SEWASIE (SVN)

Come mostrato in figura 1.1 gli attori[3] che interagiscono nel SVN sono:

- ✓ User Interface (UI);
- ✓ Query Agent (QA);
- ✓ Brokering Agent (BA);
- ✓ SEWASIE Information Node (SINode).

Un altro punto centrale del progetto è quello di valutare quale impatto può avere sulle PMI l'inserimento della componente semantica nelle informazioni presenti nel Web, sia da un punto di vista economico che organizzativo. A tal fine tra gli obiettivi iniziali, vi è anche di analizzare i potenziali benefici economici previsti dal sistema SEWASIE e i fattori interni ed esterni necessari per realizzare tali benefici

[3] Una più dettagliata descrizione dei componenti viene fornita da Raffaele Capezzera nella tesi di laurea in ingegneria informatica svolta presso l'Università degli Studi di Modena e Reggio Emilia [4].

ponendo una particolare attenzione sui cambiamenti commerciali ed organizzativi adottati dalle varie imprese all'inserimento dei nuova applicazione. Per poter variegare le caratteristiche delle richieste poste e reperire differenti sorgenti di dato, SEWASIE ha scelto di diversificare l'analisi su due settori distinti di mercato, avvalendosi della collaborazione di ditte tessili e meccaniche aderenti alla CNA.

1.3 MOMIS

MOMIS [5,6,7] (Mediator envirOnment for Multiple Information Source) è un sistema, realizzato in progetto presso l'Università degli Studi di Modena e Reggio Emilia, per la realizzazione di una vista globale (Gloabl Virtual View o GVV) su sorgenti di dato eterogenee, cioè la rappresentazione integrata e sintetizzata di informazioni provenienti da differenti sorgenti, indipendente dalla localizzazione e dalla natura delle sorgenti stesse. MOMIS implementa una metodologia semi-automatica di integrazione che segue l'approccio global-as-view dove ad ogni elemento dello schema globale corrisponde una vista sulle sorgenti ai quali l'elemento è associato, utilizzando questo approccio la struttura globale risulta quindi essere visibile agli utenti.

Da un punto di vista architetturale il sistema realizza un Mediatore [7] nella struttura classica a tre livelli:

- ✓ **Utente**: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un'unica risposta come se stesse interrogando un'unica sorgente di informazioni
- ✓ **Mediatore**: il mediatore è stato definito da Wiederhold [8] come "un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore […] Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno o al più pochi, esperti." Il mediatore (implementato dal *Global Schema* e dal *Query Manager*) gestisce l'interrogazione dell'utente combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper. Il modulo e, quindi, il linguaggio di interrogazione sono comuni a tutte le fonti.

- ✓ **Wrapper**: hanno il compito di standardizzare la modellazione della GVV. Ogni wrapper gestisce un singola sorgente svolgendo una funzione: da un lato converte le richiese del mediatore in una forma comprensibile dalla sorgente, dall'altro traduce informazioni estratte dalla sorgente nel modello usato dal mediatore.

La figura 1.2 mostra come gli attori di questo modello interagiscono tra loro.

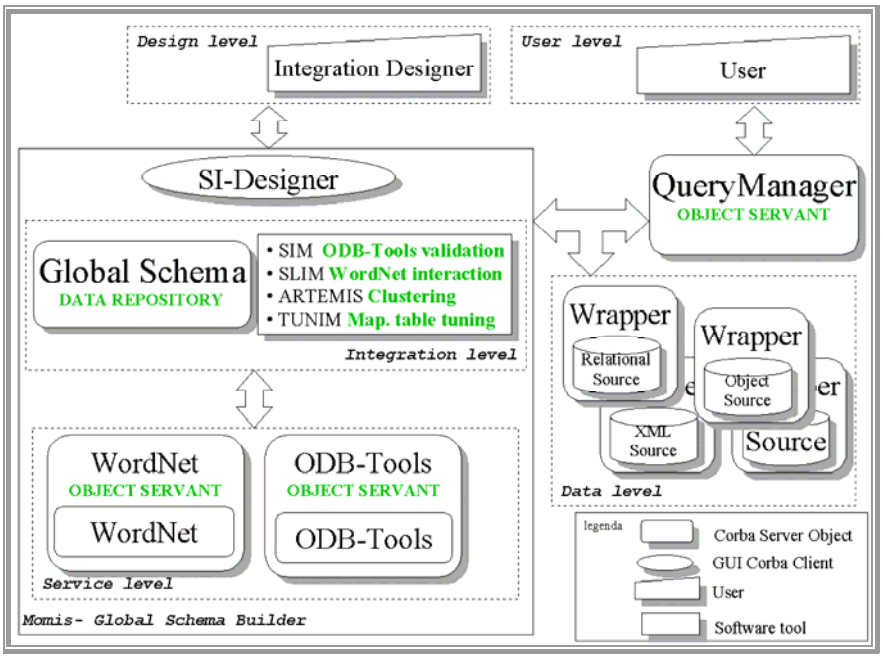

Figura 1.2: Architettura del sistema MOMIS

Un importante componente di MOMIS è lo strumento grafico Momis Ontology Builder, implementato in SI_Designer, l'interfaccia di interazione con l'utente nella fase di creazione della vista globale (lo strumento per il clustering ARTEMIS è stato integrato in SI-Designer). Tutti questi componenti comunicano tra loro mediante CORBA[4].

[4] CORBA: (Common Object Request Broker Architecture), sono specifiche del consorzio chiamato Object Management Group (OMG). CORBA permette di realizzare un'architettura distribuita mediante la definizione di standard per le chiamate alle funzioni messe a disposizione da diversi oggetti software installati su differenti sistemi operativi, in maniera che possano interagire tra loro.

Allo scopo di supportare l'integrazione semantica di informazioni eterogenee è stato introdotto un linguaggio object-oriented, chiamato ODL*I*[3] per rappresentare gli schemi concettuali che descrivono le sorgenti dati semistrutturate e non. ODL*I*[3] riprende le specifiche del linguaggio ODL (Object Definition Language) introducendo estensioni utili per modellare sorgenti semistrutturate, ODL*I*[3] è il linguaggio unico utilizzato da MOMIS, ad esempio, i Wrappers lo usano per presentare la descrizione delle sorgenti a cui sono collegati.

Il processo di integrazione per la costruzione della GVV si compone di cinque fasi principali:

- ✓ Estrazione delle sorgenti locali: wrapper dedicati per ciascuna struttura delle sorgenti ne estraggono lo schema e lo traducono in ODL*I*[3];
- ✓ Annotazione delle sorgenti locali: il progettista sceglie un significato per ciascun elemento dello schema delle sorgenti locali, utilizzando il database lessicale WordNet[5];
- ✓ Generazione del Common Thesaurus: usufruendo dall'annotazione lessicale precedentemente realizzata, il sistema genera un insieme di relazioni inter e intra-schema tra le classi e gli attributi delle sorgenti locali;
- ✓ Generazione della GVV: MOMIS genera una vista virtuale globale e un insieme di mapping tra lo schema globale e le sorgenti locali basato sulle relazioni del Thesaurus;

[5] WordNet è una risorsa linguistica sviluppata più di dieci anni fa dal linguista George Miller presso l'Università di Princeton, che organizza, definisce, descrive i concetti rilevanti della lingua inglese. La concettualizzazione del lessico è realizzata attraverso il synset: insieme di termini dal significato equivalente, che possono essere descritti da un'unica definizione, perchè esprimono un unico senso. In ogni concetto, o synset, le differenze di senso (polisemie) sono distinte, numerate e definite mediante relazioni tassonomiche e associative. Disambiguazione dei polisensi WordNet, considerato uno dei più importanti lessici standard per la lingua inglese, è disponibile gratuitamente su Internet, sia consultabile on line che scaricabile.

- ✓ Annotazione della GVV: in modo semiautomatico, un significato viene affidato agli elementi globali che compongono la GVV. Il procedimento è analogo a quello utilizzato per l'annotazione delle sorgenti locali.

Le cinque fasi saranno riprese e sviluppate maggiormente nel capitolo successivo.

Capitolo 2

LE ONTOLOGIE

2.1 Cos'è una Ontologia?

Ontologia, dal participio presente "ontos" del verbo greco "einai" (essere) e "logos" (parola), da cui "discorso sull'essere ", è la disciplina filosofica che si occupa dello studio dell'essere in quanto essere, ovvero al di là delle sue determinazioni particolari. L'ontologia si occupa quindi di studiare le qualità dell'esistenza delle cose nella loro caratteristica di essere cose che esistono, per questo motivo, ovvero per la particolarità dell'ontologia di fare riferimento al principio primo che caratterizza l'esistere delle cose, l'ontologia viene spesso identificata con la metafisica, disciplina che intende studiare le cause e il "perché" di ogni cosa, andando oltre la semplice considerazione dei dati empirici per indagare i principi primi che generano e determinano le cose esistenti.

Oggi tale termine è stato riadattato in altre discipline come l'Intelligenza Artificiale e più in generale nella comunità di ricerca dei i sistemi informativi. Esistono numerose definizioni di ontologia, al fine di chiarirne i diversi significati del termine, a seconda di quale disciplina lo adotti, in uno studio di Guarino [9] (proveniente dal settore della tecnologia dell'informazione) e Giaretta (esponente della disciplina filosofica) sono state isolate sette diverse differenti interpretazioni operative del termine "ontologia" che spaziano su discipline differenti. Potremo quindi avere l'ontologia: come disciplina filosofica; come sistema concettuale informale; come accezione semantica formale; come specificazione di una "concettualizzazione"; come rappresentazione di un sistema concettuale attraverso

una teoria logica, caratterizzata da proprietà formali specifiche o solo da obiettivi specifici; come vocabolario usato da una teoria logica; come specificazione (metalivello) di una teoria logica.

Sicuramente una definizione fra le maggiormente citate in letteratura è quella data da Tom Gruber [10] nel 1993:

> *"un'ontologia è un'esplicita specifica di una concettualizzazione. A sua volta una concettualizzazione è l'insieme di oggetti, concetti ed altre entità che si può assumere esistere in una certa area di interesse e delle relazioni che esistono tra essi."*.

Borst [11] completò tale definizione con l'aggiunta al sostantivo "concettualizzazione" dell'aggettivo "condivisa". Dalla definizione di Gruber si sono sviluppate diverse altre definizioni, come questa di M. Uschold [12]:

> *"An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are interrelated which collectively impose a structure on the domain and constrain the possible interpretations of terms"*[6]

In particolare, secondo Gruber una ontologia è esprimibile come una quintupla:

$$(C,I,R,F,A)$$

composta da:

- $C \Rightarrow$ L'insieme di **classi** che corrispondono alle entità del dominio. Esse rappresentano l'insieme di astrazioni che sono utilizzate per descrivere gli oggetti del mondo reale.

[6] Una ontologia può assumere una varietà di forme, ma necessariamente dovrà includere un vocabolario di termini e alcune precisazioni del loro significato. Questo include definizioni e un indicazione di come i concetti sono connessi, il che collettivamente impone una struttura sul dominio e un vincolo sulla possibile interpretazione dei termini.

I $\Rightarrow$ L'insieme delle **istanze**, cioè gli oggetti contenuti nel dominio che appartengono quindi al mondo reale e che si rappresentano nell'ontologia.

R $\Rightarrow$ L'insieme delle **relazioni** definite sull'insieme C in modo tale che ogni $r \in R$ sia una n-pla ordinata $r = (C_1 \times C_2 \times \ldots \times C_n)$.

F $\Rightarrow$ L'insieme delle **funzioni** che sono definite sull'insieme dei concetti e ritornano un concetto. Ogni elemento $f \in F$ è un $f : (C_1 \times C_2 \times \ldots \times C_n \rightarrow C_p)$.

I $\Rightarrow$ L'insieme degli **assiomi che** circoscrivono il senso e l'utilizzo delle classi, delle istanze, delle funzioni e delle relazioni. Essi vengono definiti tramite predicati espressi mediante un logica del primo ordine.

2.2 Classificazione delle ontologie

In questa sessione si fornirà una panoramica sulle più significative classificazioni delle ontologie presenti in letteratura.

2.2.1 Accuratezza di descrizione del dominio

Un primo criterio di classificazione può essere quello relativo al livello di accuratezza impiegato nella descrivere il dominio, questo tipo di classificazione varia soprattutto in base all'utilizzo ultimo dell'ontologia.

Come mostrato in figura 2.1 le classi possibili sono quattro:

- ✓ **Top-level ontology**: l'ontologia descrive i concetti di carattere generale che sono indipendenti da un dominio particolare o da uno specifico problema, sulle definizioni e sul significato degli elementi di tali ontologie può esserci il consenso di una grande comunità di utenti. In generale le top-level ontology esprimono una conoscenza assoluta e universale che non è necessario esplicitare.
- ✓ **Domain task ontology:** descrivono il vocabolario relativo a un generico dominio (come può essere un dominio medico o automobilistico) dando una specializzazione dei termini introdotti nella top-level ontology.

- ✓ **Task ontology** descrivono il vocabolario relativo a un generico obiettivo (come la diagnostica o le vendite), dando una specializzazione dei termini introdotti nella top-level ontology.
- ✓ **Application ontology:** descrive concetti che dipendono sia da un particolare dominio sia da un particolare obiettivo.

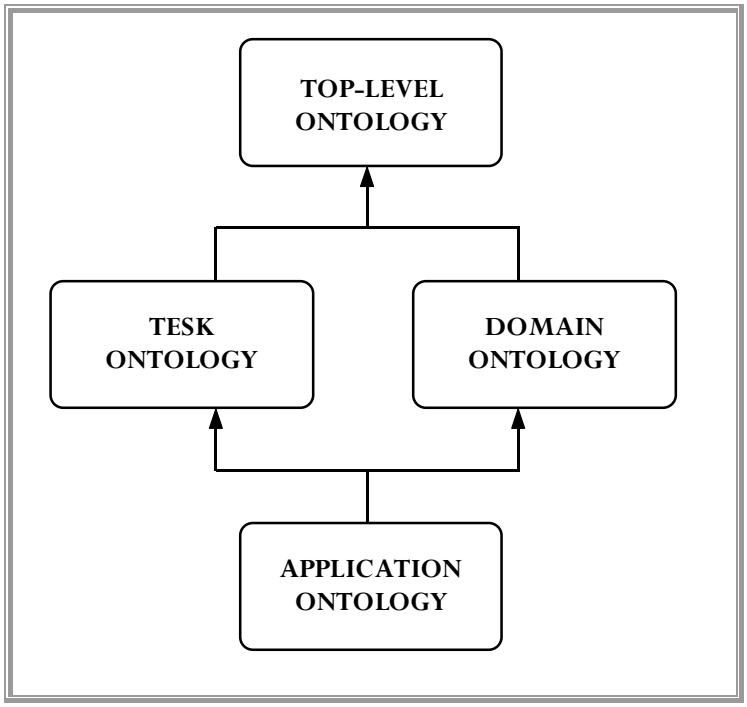

Figura 2.1: Classificazione delle ontologie
sul criterio di accuratezza nella descrizione del dominio

2.2.2 Granularità e tipo di conoscenza

Van Heijst propone due parametri di classificazione: il primo esamina la granularità dei concetti espressi, mentre il secondo esamina il tipo di conoscenza modellata nelle ontologie. Sulla base del primo parametro le ontologie vengono raggruppate in:

- ✓ **Terminological Ontologies**: queste non sono ontologie in senso stretto ma solo lessici che specificano la terminologia usata per rappresentare il dominio di un discorso non rappresenta la semantica dei termini.

- ✓ **Information Ontologies**: specificano la struttura dei dati di un database (per esempio lo schema di un database); danno significato a registrare le osservazioni di base riguardanti le istanze del database ma non definiscono i concetti che sono istanziati da queste istanze.
- ✓ **Knowledge Modelling Ontologies**: specificano concettualizzazioni di conoscenza, sono strutturalmente più ricche delle Information Ontologies e sono spesso specificate in accordo ad un particolare uso della conoscenza che descrivono.

Sulla base del secondo criterio, basato sul tipo di concetto espresso nell'ontologia, si possono avere quattro diverse categorie:

- ✓ **Ontologie di applicazione**: specificano quei concetti che sono necessari per modellare la conoscenza richiesta in specifiche applicazioni. Solitamente, O.A. specializzano i termini presi da ontologie più generali come quelle di dominio o generiche, descritte in seguito, e può estendere la conoscenza generica di dominio utilizzando componenti e metodi specifici per lo scopo. Queste ontologie non sono riutilizzabili, poiché utilizzano la conoscenza che può essere modellata in librerie di ontologie definite per la specifica operazione svolta.
- ✓ **Ontologie di dominio**: specificano i concetti che sono specifici per un particolare dominio. I progettisti definiscono una linea fra le ontologie di dominio e la conoscenza di dominio, dove la prima ha natura ontologica e la seconda epistemica[7]. La conoscenza del dominio esprime situazioni effettive in alcuni domini mentre le ontologie di dominio specificano i legami da applicare alla struttura e al contenuto del dominio della conoscenza. Ma la distinzione fra ciò che è ontologico e ciò che è epistemologico è sottile, quindi questo genere di linea spesso non può essere facilmente tracciato.

[7] Che riguarda la conoscenza, cioè la scienza esatta.

- ✓ **Ontologie generiche**: specificano concetti che sono generici in molti campi. Concetti nelle ontologie di dominio possono essere specializzati quelli che nelle ontologie generiche per poterli specializzare ad un dominio particolare. Le ontologie generiche corrispondono alle ontologie di top-level nella classificazione presentata precedentemente; tipicamente definiscono concetti come stato, evento, processo, azione, ecc.
- ✓ **Ontologie di rappresentazione**: esplicitano concettualizzazioni sottolineando i formalismi della rappresentazione della conoscenza. Provvedono una struttura senza riferirsi al mondo perché vogliono essere neutre rispetto al mondo. Le ontologie generiche di dominio sono descritte come le primitive date nelle ontologie di rappresentazione.

2.2.3 Grado di formalizzazione

Altro criterio di classificazione è il livello di formalizzazione con la quale l'ontologia viene definita, in questo caso a fare la differenza non è la conoscenza espressa, che può essere anche la stessa, ma il diverso formalismo con cui questa viene rappresentata.

Nella discussione di quanto le ontologie formali sono descritte, è di poco valore parlare di categorie, visto che il grado di formalismo è meglio compreso come un continuo piuttosto che un insieme di classi. Tuttavia possono essere individuati:

- ✓ **Ontologie altamente informali**: sono ontologie espresse in linguaggio naturale nelle quali i termini e le definizioni possono essere ambigue a causa dell'ambiguità del linguaggio naturale
- ✓ **Ontologie semi-informali**: sono ontologie espresse mediante una forma ristretta e strutturata di linguaggio naturale con una conseguente riduzione delle ambiguità.
- ✓ **Ontologie semi-formali**:sono ontologie espresse utilizzando dei linguaggi ad hoc e definiti in maniera formale.

- ✓ **Ontologie rigorosamente formali**: sono ontologie in cui i termini e le definizioni sono definiti utilizzando una semantica formale rendendole prive di ambiguità.

2.2.4 Grado di espressività

Le ontologie possono essere classificate sulla base della loro espressività quindi sulla base delle informazioni che essa deve esprimere [13]. Questa divisione permette di distinguere le diverse interpretazioni del concetto di ontologia che variano da vocabolari controllati a vincoli generali basati su un modello logico, l'elenco delle possibili categorie viene fornito seguendo una scala a formalizzazione maggiore.

- ✓ **Vocabolario**: l'ontologia esprime una lista finita di termini. Questa è la più semplice nozione di ontologia.
- ✓ **Glossario**: l'ontologia esprime una lista di termini con il corrispettivo significato, questi ultimi sono solitamente espressi in linguaggio naturale, quindi vincolati alla comprensione del lettore; ma, a causa dell'ambiguità che descrive i termini, difficilmente i glossari possono essere utilizzati da applicazioni software.
- ✓ **Elenchi di termini sinonimi**: l'ontologia è un glossario arricchito da relazioni tra i termini, queste vengono definite mediante il concetto di sinonimia. Questi elenchi spesso non hanno una struttura gerarchia esplicita ricavabile però dalla navigazione delle relazioni.
- ✓ **Gerarchie informali Is-a**: l'ontologia fornisce una nozione di specializzazione e di generalizzazione senza una rigida definizione della gerarchia, questo è il tipo di ontologia spesso adottato nei motori di ricerca in Internet che presentano liste di siti suddivisi in categorie sulla base dei contenuti.
- ✓ **Gerarchie formali Is-a**: l'ontologia fornisce una nozione di specializzazione e di generalizzazione con una rigida definizione della gerarchia, in questo tipo di gerarchie il concetto di eredità tra concetti posti

in relazione è sempre applicabile; in più queste ontologie possono racchiudere unicamente dei nomi di classi.

- ✓ **Istanze formali**: le ontologie che includono istanze formali sono una estensione naturale delle ontologie, contengono anche i contenuti dei concetti coinvolti.
- ✓ **Frame (descrizione delle proprietà dei concetti)**: l'ontologia descrive i concetti definendo le loro proprietà caratteristiche, che sono a loro volta dei concetti.
- ✓ **Restrizione dei valori**: l'ontologia consente restrizioni all'insieme di valori associati alle proprietà, queste sono di solito ereditate dalle relazioni Is-a.
- ✓ **Vincoli basati su una logica generale**: l'ontologia ha il più alto grado di espressività, ad esempio, le proprietà possono essere definite sulla base di equazioni di tipo logico matematico, possono usare dei valori definiti attraverso altre proprietà e sono generalmente descritte utilizzando un linguaggio espressivo che consente la specifica di vincoli sui concetti e sulle proprietà seguendo la logica del primo ordine[8].

2.3 Creazione di una ontologia in Momis

Nel capitolo precedente è stato descritto come il progetto MOMIS crea una ontologia avvalendosi di una interfaccia grafica, Momis Ontology Builder, che accompagna l'utente nel processo di integrazione delle sorgenti. La seguente immagine riassume la procedura di generazione della GVV adottata dal sistema.

[8] Nel libro di L. Console, E. Lamma, P. Mello, M.Milano: "Programmazione Logica e Prolog", si definisce la logica di primo ordine sia in termini ontologici, riguardo a fatti,oggetti e relazioni, sia in termini epistemologici, in termine di vero, falso o sconosciuto.[14]

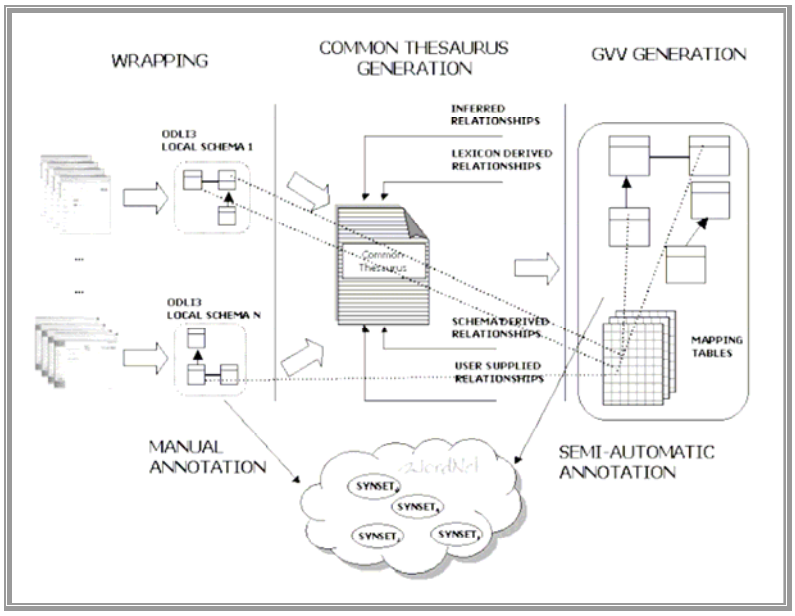

Figura 2.2: Architettura del sistema MOMIS

Le fasi precedentemente citate verranno ora descritte più nel dettaglio.

2.3.2 Estrazione delle sorgenti locali

Il primo passo quando si costruisce una vista globale con il sistema MOMIS è la costruzione dello schema concettuale, cioè una rappresentazione semantica delle informazioni contenute nelle sorgenti. Ad ognuna viene assegnato un wrapper per convertire la sua struttura dati, se ne esiste una, in una rappresentata mediante il linguaggio ODL*I*[3]. I wrapper forniscono quindi uno strumento necessario per gestire l'eterogeneità nella sorgenti da integrare, tanto da rendere la sua realizzazione un operazione cruciale nel processo di integrazione.

Per sorgenti di informazione strutturate come database relazionali o object-oriented è sempre disponibile una descrizione dello schema la cui traduzione nel modello comune adottato è sempre immediata. Per sorgenti semi-strutturate (documenti XML e pagine web) non è possibile ottenere direttamente una descrizione dello schema, poiché caratterizzati dal fatto di essere "autodescriventi", cioè la descrizione dello schema è contenuta nel dato stesso. Per affrontare questa situazione è stato sviluppato un wrapper in grado di estrarre lo schema di documenti

XML validi, cioè conformi allo standard DTD[9]. Il processo consiste nell'effettuare un parsing della DTD in modo da costruire gli alberi degli elementi ad essa associato e , quindi, la rispettiva traduzione in formato ODL$^{I^3}$, tale passaggio comporta, tuttavia, una perdita di semantica derivante dal maggiore potere espressivo di ODL$^{I^3}$ rispetto a XML.

Il wrapper per i file XML, viene inoltre utilizzato nel sistema MOMIS anche per interagire con il documento HTML. Il limite di questi documenti risiede nel fatto che non esiste una separazione tra le informazioni riguardanti la struttura e quelle riguardanti il layout. La struttura viene, quindi, estratta da un wrapper esterno di nome Lixto, che traduce il contenuto della pagina web in un file XML compatibile DTD, passandola quindi al wrapper XML che provvederà a tradurla come sopra descritto nel modello in ODL$^{I^3}$.

2.3.3 Annotazione delle sorgenti locali

Il processo di integrazione delle sorgenti richiede come secondo passaggio l'annotazione delle informazioni contenute nelle sorgenti locali. A ciascun elemento (classe o attributo) delle sorgenti importate viene affidata una forma (o significante) ed un significato.

Questo strumento è supportato da due principali motivazioni: la prima si riferisce al fatto che i termini utilizzati per descrivere uno schema o le strutture delle sorgenti possiedono una semantica intrinseca; la seconda, invece, si basa sulla possibilità di esportare il risultato del processo, rendendolo sfruttabile da parte di applicazioni esterne. Per far ciò esso deve contenere un significato che sia universalmente riconosciuto.

Lo strumento utilizzato dal sistema come giuda all'utente per svolgere queste funzioni è WordNet. Questo database è organizzato come una matrice dove ciascuna riga rappresenta un synset (un insieme dei significati che rappresentano lo stesso

[9] DTD, acronimo per Document Type Definition, è un set di regole, contenute in un archivio di testo, che definiscono la struttura, la sintassi, e il vocabolario come riferito ai tags e gli attributi del documento XML al fine di costruire un documento che possa essere interpretato e visualizzato in modo automatico.

concetto), mentre ciascuna colonna rappresenta il vocabolo con il quale viene rappresentato il concetto.

Riportiamo come esempio l'elemento *e1,2* nella tabella 2.1, esso indica che il significato *Sig_1* può essere espresso dal vocabolo *Voc_2*

	Voc_1	***Voc_2***	***Voc_3***	***Voc_4***
Sig_1		*e1,2*		
Sig_2	*e2,1*			
Sig_3		*e3,2*		*e3,4*
Sig_4			*e4,3*	

Tabella 2.1: Matrice di Wordnet

E' immediato rilevare relazioni di sinonimia tra vocaboli con relazioni sulla stessa riga, e di polisemia tra elementi della stessa colonna, cioè il vocabolo può assumere più significati differenti. A causa di questo fenomeno la fase di annotazione si compone di due parti distinte: quella di scelta del vocabolo di riferimento, e quella di scelta del significato appropriato tra quelli associati al vocabolo. Questa operazione può esser eseguita anche in forma automatica, in questo caso l'annotazione viene effettuata sulla base del nome della classe o attributo di riferimento. La funzione di questa fase è quella di generare rapporti semantici tra classi e attributi delle sorgenti, da usare come materiale per la generazione del Common Thesaurus, pertanto se non viene scelto alcun significato per una classe o un attributo nessuna relazione potrà derivare dal valore semantico del suo nome. L'importanza di questa fase, viene maggiormente evidenziata se si tiene presente che errori in questa fase si ripercuotono in quella successiva, e possono causare l'introduzione di relazioni imprecise o errate nel Common Thesaurus strumento base per la creazione dei cluster.

2.3.3 Generazione del Common Thesaurus

Una volta consolidate le conoscenze semantiche sugli schemi importati, il sistema procede a creare una libreria di relazioni (il Common Thesaurus) che guideranno la creazione dei cluster per generare la GVV. Il linguaggio ODL[3]

supporta due tipi di relazioni: quelle intenzionali, e quelle estensionali. In particolare le prime, di carattere terminologico, esprimono la conoscenza inter e intra schema delle sorgenti. MOMIS prevede quattro tipi fondamentali di relazioni definite tra classi o attributi e specificate secondo i nomi delle classi o degli attributi stessi. Queste sono:

- ✓ **SYN** (Synonim): mostra una relazione di sinonimia tra due termini t_i e t_y, con $t_i \neq t_y$, ovvero possono essere interscambiabili perché identificano lo stesso significato, per questo motivo è una relazione di tipo simmetrica;
- ✓ **BT** (Broader Term): mostra una relazione tra due termini t_i e t_y, tale che t_i ha un significato più generale di t_y. Questo tipo di relazione non è simmetrica;
- ✓ **NT** (Narrower Term): mostra le relazione inversa a BT, cioè tratta la relazione opposta alla precedente e come quella anche questo tipo di relazione non è simmetrica.
- ✓ **RT** (Related Term): è una relazione simmetrica tra due termini, t_i e t_y, che possono essere usati nello stesso contesto o tra i quali esiste un legame generico.

Le relazioni intenzionali relative a due classi possono essere a loro volta rafforzate definendole come relazioni estensionali [7]: SYS_{ext}, BT_{ext}, NT_{ext} e $DISJ_{ext}$.

Riportiamo qui di seguito un breve descrizione dei quattro procedimenti attraverso i quali vengono importare le relazioni sopra descritte. È da sottolineare il fatto che termini a cui non è stato affidato alcun significato, dove cioè l'annotazione non è stata completata, non contribuiscono all'estrazione di relazioni lessicali nel Common Thesaurus.

Relazioni derivanti dallo schema

Le relazioni derivanti dallo schema sono estratte direttamente dalla struttura delle sorgenti, analizzando separatamente ciascuno schema. Ad esempio, dai DBMS relazionali, le relazioni interschema RT possono essere ricavate dalle specifiche di foreign key, mentre le relazioni di BT e NT vengono estratte quando una foreign key

è primary key sia nella sorgente originale che in quella referenziata. Altri esempi arrivano dai DBMS ad oggetti, dove il concetto di ereditarietà genera le relazioni di BT_{ext} e di NT_{ext}, e dai documenti XML, una struttura espressa attraverso un DTD. In questo caso le relazioni di BT e di NT vengono estratte dai tags ID e dai tag IDREF, mentre gli elementi innestati generano relazioni RT.

Relazioni derivate dal lessico

SI_Designer mette a frutto l'opera di annotazione effettuata in precedenza per generare relazioni lessicali tra gli elementi delle sorgenti. Wordnet fornisce una fitta ragnatela di relazioni che mette in associazione i significati in esso contenuti. Ognuna di queste relazioni implica, quindi, una relazione all'interno del Common Thesaurus tra gli elementi a cui il progettista ha associato quel significato. In particolare uno schema di traduzione ODL*I*[3] prevede le seguenti relazioni:

- ✓ Sinonimia: due significati esprimono lo stesso concetto, genera una relazione SYN tra i due elementi.
- ✓ Ipernimia: lega un significato con il corrispondente più generale, genera una relazione BT tra i due elementi.
- ✓ Iponimia: lega un significato con il corrispondente più specifico, genera relazioni NT tra i due elementi.
- ✓ Olonimia/Meronimia: la meronimia lega un significato con un suo componente, la relazione opposta è l'olonimia, entrambe generano relazioni RT tra i due elementi.
- ✓ Correlazione: lega due significati che condividono un significato più generalizzante, genera una relazione RT tra i due elementi.

La creazione del Common Thesaurus si conclude con due fasi: valutazione delle relazioni in esso inserite e generazione di nuove relazioni. In particolare, per la prima, il sistema utilizza il modulo ODB-Tools [21], eseguendo una traduzione dell'intero contenuto del Common Thesaurus e delle sorgenti locali nel linguaggio OLCD[10], questo viene successivamente analizzato dal sistema per valutare la

[10] OLCD è l'acronimo di Object Language with Complements allowing Descriptive Cycles.

compatibilità dei domini associati con gli attributi tradotti. Per la generazione delle nuove relazioni, il sistema esegue invece la chiusura transitiva di quelle già presenti.

2.3.4 Generazione della GVV

In questa fase del processo di creazione di una ontologia il sistema analizza le informazioni presenti nel Common Thesaurus per generare le classi globali della Global Virtual integrated View (GVV). Questa attività si articola in due fasi:

- ✓ Generazione dei cluster;
- ✓ Generazione delle classi globali.

Generazione del cluster

Per integrare le classi delle differenti sorgenti, espresse nel linguaggio ODL$^{I^3}$, nelle classi globali, anch'esse espresse in ODL$^{I^3}$, MOMIS si avvale di un algoritmo di clustering gerarchico basato sul concetto di affinità. In questo modo il sistema identifica le classi ODL$^{I^3}$ che riportano le stesse informazioni nei differenti schemi delle sorgenti, oppure che descrivono informazioni collegate semanticamente, danno misura del livello di similarità della loro struttura.. Lo strumento utilizzato è ARTEMIS [15], esso assegna dei coefficienti di affinità agli elementi delle classi presenti delle relazioni contenute nel Common Thesaurus. Questi definiscono il grado di relazione semantica tra le classi considerando le relazioni che intercorrono tra i loro nomi (*Naming Affinity Coefficient*) e il rapporto tra il numero degli attributi in relazione e il numero degli attributi totali che esse possiedono (*Structural Affinity Coefficient*). La somma dei due coefficienti, chiamata *Global Affinity Coefficient,* è una combinazione lineare tra il coefficiente di affinità basata sul nome e la struttura delle classi. Questi coefficienti, fissati con parametri di default dal sistema, possono essere modificati dall'utente manualmente prima della creazione del cluster.

Il risultato di questa procedura è un albero di affinità, dove le foglie sono le classi locali e ogni nodo corrisponde un coefficiente di affinità. ARTEMIS esegue un'operazione di sintesi creando le classi globali da questo albero attraverso un

meccanismo a soglia, decretando l'unione di due classi tra di loro se il coefficiente di affinità globale supera la soglia definita.

Generazione delle classi globali

Dal mapping delle classi locali in classi globali, il sistema crea per ognuna delle seconde un set di attributi globali ognuno dei quali viene messo in corrispondenza con gli attributi locali. In molti casi, le corrispondenze sono uniche, mentre in altri casi viene richiesto al progettista di eseguire una scelta tra diverse corrispondenze trovate dal sistema.

Il costruttore degli attributi globali elimina le informazioni ridondanti contenute nelle relazioni del Common Thesaurus unendo gli attributi di tutte le classi appartenenti al cluster, e fondendo quelli similari. Il processo di fusione è automatico per gli attributi che sono associati a relazioni validate mentre non è sempre automatico quando non lo sono.

Gli attributi che hanno una relazione terminologica valida sono unificati dentro un unico attributo globale. Il processo di unificazione è svolto automaticamente per quello che riguarda i nomi con queste regole: per gli attributi che hanno una relazione di tipo SYN, solo un termine è selezionato come nome per il corrispondente attributo globale, per attributi che hanno una relazione BT/NT, al corrispondente attributo viene scelto e assegnato come nome il termine più generale per tutti.

Il progettista può visualizzare e modificare queste relazioni intensionali avvalendosi di Mapping Tables, una per ogni classe globale. Queste hanno una struttura a colonna, contenente, nella prima il nome assegnato all'attributo globale e nelle successive (ognuna relativa ad una delle classe locali della classe globale) gli attributi locali in esso mappati. La funzione di mapping $MT_{[GA][LC]}$, dove *GA* indica l'attributo globale mentre *LC* la classe locale, può assumere varie conformazioni come mostrato nella seguente tabella.

Tipo	Funzione di mapping	Definizione
IDENTITÀ	$MT_{[GA][LC]} = LA$	Il valore di *GA* è uguale a quello di LA.
CONGIUNZIONE	$MT_{[GA][LC]} = LA_1,\ldots ,LA_n$	Il valore *GA* è ottenuto dalla congiunzione di un insieme di valori LA della classe locale LC.
COSTANTE	$MT_{[GA][L]} = k$	Il valore di *GA* corrispondente alla classe locale *LC* assume un valore costante scelto dal progettista
INDEFINITO	$MT_{[GA][L]} = null$	Il valore di *GA* non è definito per la classe *LC*.

Tabella 2.2: Conformazione funzione di mapping $MT_{[GA][LC]}$

2.3.5 Annotazione della GVV

Questa fase conclusiva nel processo di integrazione delle sorgenti in una ontologia, prevede di affidare a ogni elemento della GVV un nome e un significato. La necessità di questa operazione, che verrà maggiormente discussa nel capitolo quattro, trova spiegazione nella possibilità di fornire uno strumento indispensabile nel riutilizzo della nuova GVV come sorgente locale per le successive integrazioni, facilitata dal fatto che ogni elemento ha un significato universalmente riconoscibile. Come per quanto riguarda l'annotazione delle sorgenti locali, anche per la GVV ogni elemento ha un nome, inteso come etichetta che identifica l'elemento all'interno di MOMIS, e un significato, per l'identificazione dell'elemento. Essi sono assegnati in modo automatico dal sistema, tuttavia viene data libertà al progettista attraverso comandi modificabili manualmente.

Capitolo 3

DINAMICA DI UNA ONTOLOGIA

In questa sezione verranno discussi come prima cosa due possibili approcci usati per gestire ciò che viene definita la dinamica di una ontologia, successivamente verranno riportati esempi, sempre tratti dalla letteratura, su quali possono essere le problematiche da affrontare. In particolare verrà riportato uno studio sul pruning e dell'integrazione di ontologie.

3.1 Linee generali

La gestione di grandi quantità di informazioni è di crescente importanza nelle realtà odierna, dove vede una sempre più massiccia presenza di grandi organizzazioni. Come visto nei capitoli precedenti, le difficoltà riguardati la gestione e l'accesso a questi dati è stato risolto introducendo il concetto di ontologia. Tuttavia, l'operazione di integrazione di informazioni e creazione di una ontologia richiede ancora molti sforzi da parte del progettista oppure l'impiego di tecniche per costruirne una ad hoc, questo perché, esistono poche metodologie per migliorare la situazione a posteriori. Tuttavia, anche una ontologia perfetta in origine necessita, dopo un determinato periodo di tempo, di aggiornamenti per rimanere coerente con i cambiamenti del mondo reale. Possono essere un esempio i mutamenti non prevedibili dell'ambiente in cui l'ontologia opera, oppure, l'aggiunta o la modifica dei requisiti degli utenti rispetto a quelli iniziali su cui era stata costruita. Questo discorso diventa ancor più presente se si parla di applicazioni che operano in ambiente di Internet e del Web Semantico, basate su sorgenti dati eterogenee fortemente distribuite.

Dalla definizione di Gruber, precedentemente fornita, si possono individuare tre aree di interesse dove è possibile agire se si parla di dinamica di una ontologia:

- ✓ **cambiamenti nel dominio**: riguardano cambiamenti nell'ambiente in cui l'ontologia è attiva, esempio modifiche sulla struttura dei modelli del database selle sorgenti, come il merge di due sorgenti.
- ✓ **cambiamenti nelle concettualizzazioni condivise**: un aspetto notevole da rimarcare è che questo tipo di cambiamenti può presentarsi più volte. Fensel [16] descrive le ontologie come reti dinamiche di significati, nelle quali il consenso è ottenuto in un processo sociale di scambio di informazioni e di significato. Differenti scopi possono necessitare differenti visioni sul dominio e conseguentemente una differente concettualizzazioni. Questa visione da un doppio ruolo alle ontologie nello scambio di informazioni: sia come prerequisito per lo scambio di informazioni che il risultato di questo processo di scambio. Per esempio quando una ontologia è adattata per un nuovo scopo o dominio, le modifiche rappresentano cambiamenti nella concettualizzazione. La trasformazione rimane quindi limitata al modo in cui la concettualizzazione è formalmente memorizzata, generando un cambiamento di facile soluzione, poiché dovrebbe rimanere nella semantica, come le varianti della specificazione dovrebbero essere equivalenti e causare solo cambiamenti sintattici.
- ✓ **cambiamenti nelle specificazioni**: non sempre lo scopo per il quale una applicazione viene implementata rimane lo stesso nel tempo. Esso si può estendere ed evolversi in base, ad esempio, alle esigenze degli utenti che lo utilizzano, in questo caso può risultare necessario anche modificare l'ontologia di riferimento.

Dopo questa panoramica su quali possono essere le cause dei cambiamento, è facile capire la necessità di ricercare e sviluppare oggi algoritmi, metodologie, applicazioni che gestiscano gli aspetti dinamici di una ontologia. In letteratura si possono trovare due correnti di pensiero su quale tecnica adottare per la risoluzione di questi problemi: l'approccio basato sulla evoluzione [17,18,19] e [20,21] l'approccio basato sulle versioni. Queste tipologie di approccio sono oggetto di

studio anche in altri campi della gestione della dinamica delle informazioni, quali database relazionali e altre applicazioni software, viene quindi rimandata in appendice un confronto con quanto trattato qui di seguito.

3.1.1 Approccio basato sulla evoluzione

L'approccio che si basa sulla evoluzione dell'ontologia affronta il problema della dinamica in tutta la sua complessità, gestendo ogni conseguenza che un cambiamento dell'ontologia impone sulle sorgente ad esso collegate. Questo è un approccio abbastanza complesso basato principalmente sul concetto di "**consistenza di una ontologia**". Heflin[18], nella tesi di dottorato, si avvale di esempi e di un insieme di principi per definire quali sono le basi di questo approccio:

> *"The revision of an ontology should not change the well-formedness of resources that commit to an earlier version of the ontology."*[11]

In pratica una cancellazione effettiva probabilmente accade raramente, più facilmente un termine è rimosso perché può essere fuso con un altro termine o viene preferito un nome diverso.

> *"Resources that commit to a revised ontology can be integrated with resources that commit to compatible prior versions of the ontology."*[12]

Una modifica in una parte dell'ontologia può generare delle inconsistenze sia in un'altra porzione della stessa ontologia, che nelle applicazioni che sono basate su quella ontologia.

Da qui deriva la seguente definizione:

> *Data un linguaggio di logica del primo ordine Γ, una ontologia è una quintupla $\langle V, A, E, P, B \rangle$, dove il vocabolario $V \subset S_p$, cioè l'insieme*

[11] La revisione di un'ontologia non dovrebbe cambiare la buona strutturalità delle risorse che utilizzano una versione precedente delle ontologie.

[12] Le risorse che usano una ontologia modificata devono poter essere integrate con risorse che utilizzano versioni precedenti dell'ontologia.

dei predicati inclusi nei simboli non logici di Γ, gli assiomi $A \subset W$, cioè l'insieme infinito di formule well-formed che possono essere costruite in Γ, $E \subset O_E$,cioè l'insieme delle ontologie estese di O, $P \subset O_E$, è l'insieme delle versioni precedenti dell'ontologia, e $B \subset P$ è l'insieme delle ontologie con cui O è backwards-compatible[13].

Un altro problema, estremamente rilevante nel campo del web semantico, quando si parla di evoluzione di ontologie, è quello della "scalabilità di una ontologia" [18]. L'utilizzo della logica del primo ordine non permette l'implementazione di questa caratteristica, è necessario quindi ricercare altre vie.

Un aspetto della scalabilità è quello di implementare algoritmi di ragionamento a risorse limitate (che si limitano nel tempo di computazione, nel il numero di step o altro) dove lo scopo della ricerca non è la necessità di ottenere tutte le risposte complete al problema ma solo un insieme di risposte corrette. Data l'estensione del Web, non sembra possibile che ogni ragionatore abbia accesso a tutte le asserzioni, come del resto è improbabile che un algoritmo ben formato possa essere realmente completo in senso globale.

Un'altro aspetto della scalabilità è quello di ridurre l'espressività del linguaggio. Questa è stata un'importante direzione di sviluppo per la comunità della rappresentazione della conoscenza che ha provato a specificare la complessità computazionale dei linguaggi con varie funzioni, al fine di sviluppare linguaggi che massimizzino l'espressività ma allo stesso tempo minimizzino la loro complessità.

La struttura maggiormente nota che si basa su un approccio evolutivo per la gestione degli aspetti dinamici è il framework KAON [17], articolato in sei fasi successive, come mostrato nella figura 3.1, ha una struttura circolare, dal momento che la validazione dei cambiamenti realizzati può automaticamente indurre nuovi cambiamenti per ottenere la consistenza del modello o per soddisfare le aspettative degli utenti.

[13] Un ontologia O_2 è backwards-compatible con una ontologia O_1 se ogni modello inteso di O_1 è un modello inteso di O_2 e $V_1 \subseteq V_2$, e il modello di una ontologia è una interpretazione che soddisfa ogni assioma.

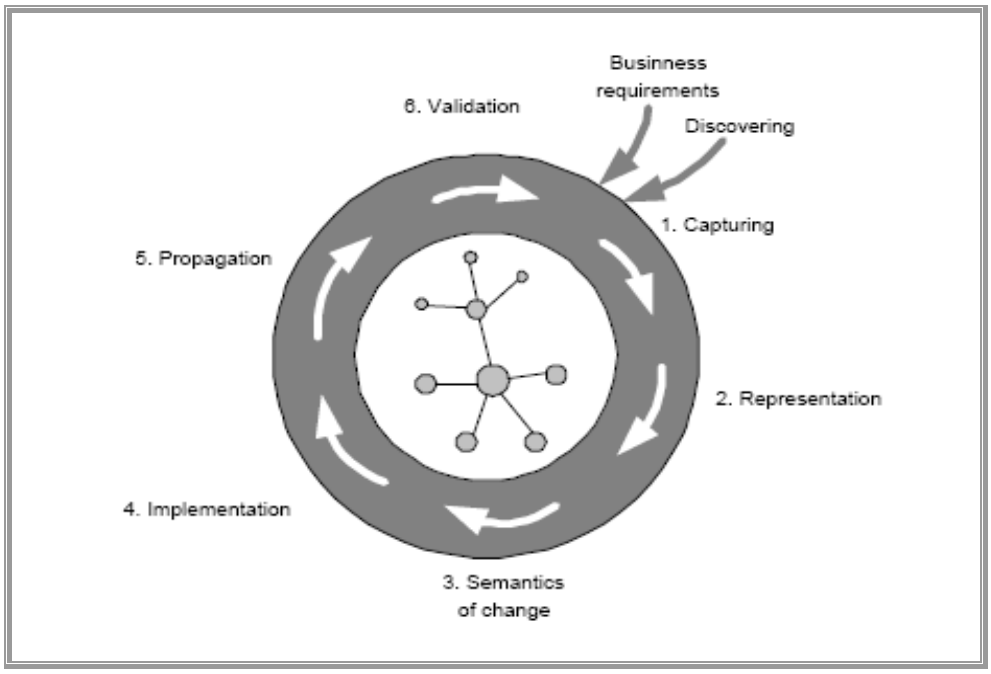

Figura 3.1: Processo di evoluzione di una ontologia in Kaoma

La parte maggiormente significativa è comunque quella inerente alla semantica dei cambiamenti, dalla fase 2 alla fase 5, meglio mostrata in figura 3.2.

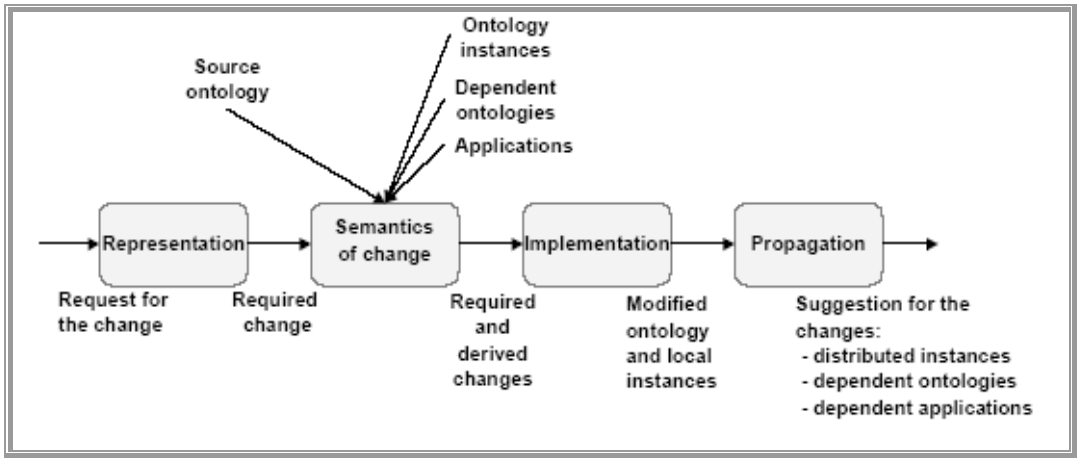

Figura 3.2: Le quattro fasi del processo di semantica dei cambiamenti

Qui l'utente può valutare l'impatto del cambiamento di una sorgente, cioè la propagazione delle necessità di cambiamenti che esso può generare, elaborare una strategia di raffinamento nei passaggi successivi per contenere la propagazione e infine valutare la necessità effettiva di procedere all'evoluzione della sorgente.

Un studio, su quali siano le trasformazioni possibili applicabili ad una ontologia, è stato effettuato nel progetto realizzato da Leehneer [19]. Partendo da un approccio simile a quello precedentemente descritto, esso sfrutta la definizione di semantica dei mondi possibili[14], sviluppata da Saul Kripke nel 1963, per definire quale possa esser l'evoluzione di una ontologia.

Il modello ha una struttura formata da tre componenti: un set *K* di mondi possibili, un insieme di relazioni di accessibilità *R(u,v)* definite su *K* e un funzione di evoluzione $\Phi(\phi,\Omega)$, in modo tale da rendere una ontologia formale come un insieme di oggetti matematici.

In questa analogia, il passaggio da una ontologia Ω_1 ad un'altra Ω_2, si riduce ad una processo di evoluzione formato da una trasformazione, definita come una

[14] Kripke intendeva innanzitutto dare una soluzione intuitivamente accettabile a un paradosso presente in tutte le logiche modali: utilizzando la legge dell'identità di Leibniz, cioè se due oggetti sono uguali allora anche due applicazioni ad essi dello stesso predicato sono identiche: $X=Y \rightarrow p(x)=p(y)$, si ottiene che l'identità di due proposizioni implica la necessità di tale identità. Nella teoria di Kripke un mondo possibile è definito come un insieme di enunciati non contraddittori. Un mondo è possibile, pertanto, relativamente a qualche altro mondo, rispetto al quale si riesce a dimostrare la non contraddittorietà, e la relazione fra i due mondi viene chiamata "relazione di accessibilità". Ogni predicato può allora essere vero in qualche mondo, e falso in qualche altro.

In questo scenario si può definire in termini più precisi la distinzione di Frege fra:

- ✓ estensione di un predicato è l'insieme dei mondi nei quali esso è vero (non soltanto l'insieme degli oggetti che lo rendono vero);
- ✓ intensione di un predicato è la funzione che assegna ad ogni mondo possibile l'estensione di oggetti che soddisfano quel predicato.

Per esempio, l'intensione di "rosso" assegna ad ogni mondo possibile un insieme di oggetti rossi. A differenza della logica classica, in cui un predicato ha soltanto un valore di verità, nella teoria di Kripke un predicato ha associato uno spettro di valori di verità, ciascuno relativo a uno dei mondi possibili. Kripke dimostrò che la logica "modale" dell'operatore "necessita" si poteva derivare da questa teoria imponendo opportune restrizioni alla relazione di accessibilità: un'espressione è necessaria in un certo mondo quando è vera in tutti i mondi possibili, è possibile quando è vera in almeno uno dei mondi possibili. [22]

sequenza finita di operazioni di modifica, atomiche e complesse, che permettono di passare da Ω_1 ad Ω_2. Obiettivo di questo progetto è quello di inserire all'interno dell'ontologia nuove informazioni che non entrino in conflitto con quello contenute nella ontologia di partenza.

Dalla teoria AGM[15] si possono individuare tre tipi di operazioni applicabili nella trasformazione di una ontologia:

- ✓ **Espansione**: un nuovo elemento di conoscenza ϕ viene inserito all'interno dell'ontologia Ω con la gestione della propagazione di tutte le informazioni aggiuntive ad essa collegate, non piccole nel caso di ontologie di grandi dimensioni.
- ✓ **Revisione**: un nuovo elemento di conoscenza ϕ che risulta essere inconsistente per l'ontologia Ω viene aggiunto, questo porta all'eliminazione di alcune vecchie decisioni interne ad Ω per mantenere la consistenza dell'ontologia risultante.
- ✓ **Contrazione**: alcuni elementi di conoscenza ϕ di Ω vengono ritrattati per dare spazio a nuovi elementi di conoscenza.

[15] Teoria AGM proposta nel 1985 da C. Alchourròn, P. Gärdenfors e D. Makinson, dai quali trova riferimento nella sua denominazione, ed implementata nel campo dell'intelligenza artificiale, affronta il problema della revisione, non irrazionale, della base di conoscenza alla luce di nuove informazioni, che deve essere realizzata mediante una funzione che soddisfi determinate condizioni di integrità racchiuse nei postulati AGM. Il processo che questi postulati tentano di descrivere come razionale è in massima parte non-monotono, aggiungere, infatti, informazioni alla base di conoscenza di un agente può causare la perdita di alcune delle conclusioni precedentemente accettate. Agli inizi degli anni '90, Gärdenfors e Makinson hanno esplicitato tale connessione, elaborando una "traduzione" dei postulati AGM in termini di una revisione di conseguenza razionale:"belief revision"[23].

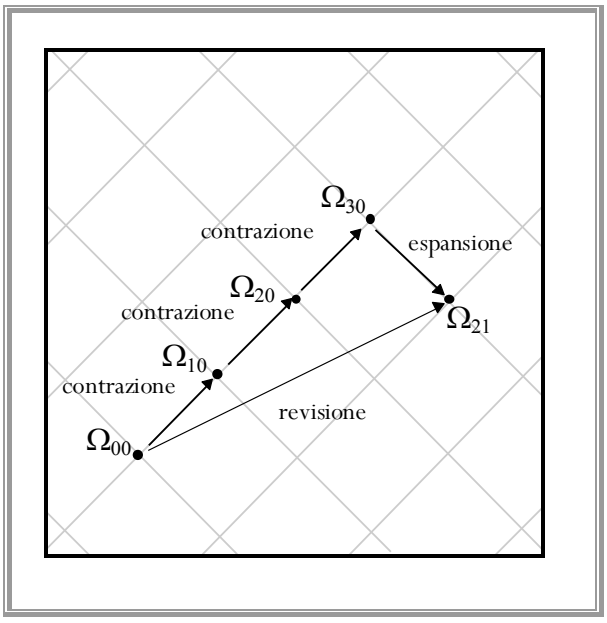

Figura 3.3: Trasformazioni di contrazione, espansione e revisione di una ontologia nello spazio di quelle possibili

Le trasformazioni possibili nello spazio delle ontologie possono essere meglio visualizzati avvalendosi della griglia di Lindenbaum dove ogni nodo rappresenta una possibile ontologia e ogni collegamento una trasformazione tra due di esse, come mostrato in figura 3.3.

Queste tre non sono, tuttavia le uniche trasformazioni che si possono avere se si parla di evoluzione di una ontologia. Il progetto di Leenheer ha esaminato altri tre tipi di relazione:

- ✓ trasformazione per preservare le equivalenze;
- ✓ propagazione dei cambiamenti a cascata;
- ✓ contenuti possibili o necessari.

Trasformazione per preservare le equivalenze

La relazione di equivalenza, mostrata in figura 3.4, è una trasformazione riflessiva, simmetrica e transitiva che genera una classificazione delle ontologie possibili in un set di classi disgiunte tra loro, creando in questo modo un insieme di rappresentazioni equivalenti della stessa concettualizzazione.

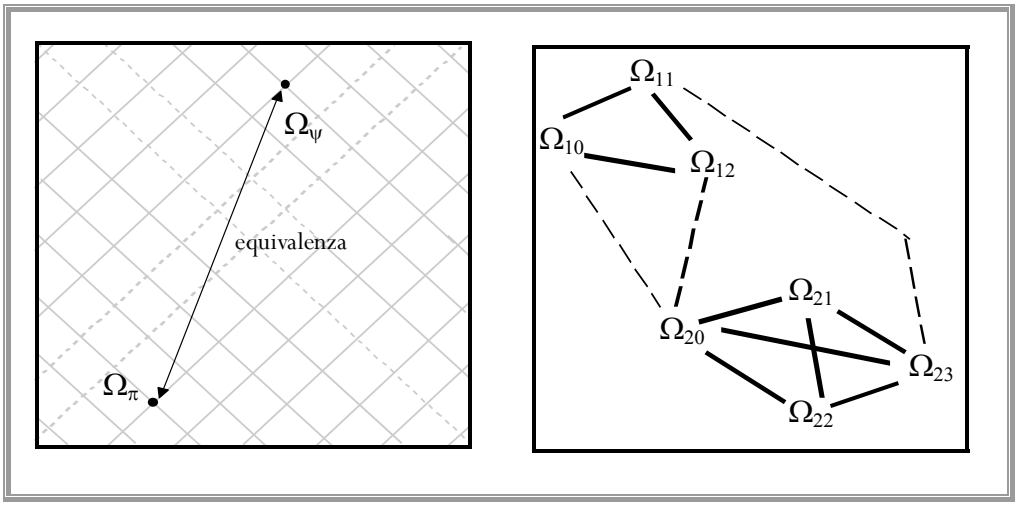

Figura 3.4: Trasformazione per preservare le equivalenze

Propagazione dei cambiamenti a cascata

Se una ontologia Ω_1 è inclusa in un'altra ontologia Ω_2, scenario illustrato in figura 3.5, una trasformazione T_1 da Ω_1 a Ω_1^*, crea un procedimento a cascata sulle ontologie che includono Ω_1, generando T_2 fra Ω_2 e Ω_2^*. Se la catena di inclusione continuasse fino a Ω_n, la trasformazione andrebbe avanti generando trasformazioni fino a T_n fra Ω_n e Ω_n^*. La propagazione dei cambiamenti è monotona verso l'alto, cioè una trasformazione a metà della catena di inclusioni tra ontologie non comporta trasformazioni in quelle interne, figura 3.5 ontologia Ω_0 .

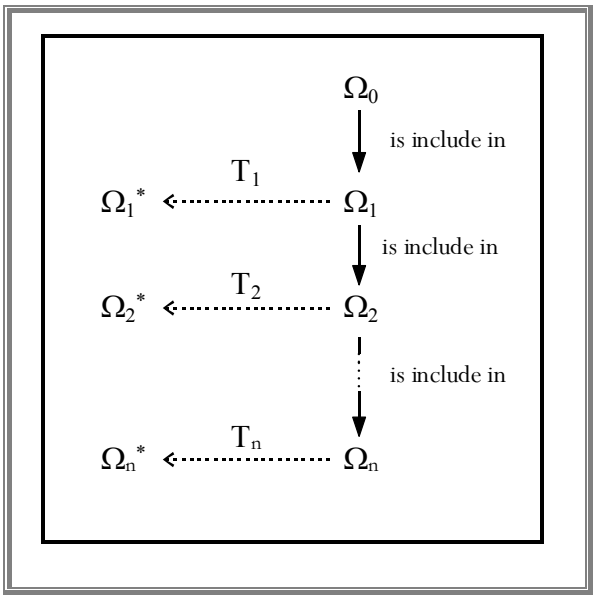

Figura 3.5: Propagazione dei cambiamenti a cascata

Contenuti possibili o necessari

Gli elementi di informazione ϕ, appartenenti ad un' ontologia secondo una relazione $\Phi(\phi,\Omega) \rightarrow \{T,F\}$ sul Ω, si possono classificare tra i contenuti necessari o possibili. Questa divisione usa come parametro di classificazione il valore di verità che l'elemento assume nelle ontologie Ω_0, accessibili dalla corrente ontologia attraverso una trasformazione consistente. Un elemento si definisce necessario se è vero in tutte le ontologie Ω_0, si definisce possibile se è vero solo in alcune di queste. In questo caso l'inserimento di tale elemento può essere semplificata, senza che l'ontologia perda consistenza.

3.1.2 Approccio basato sulle versioni

L'approccio basato sulle versioni, a differenza del primo basato sulle evoluzioni, gestisce i cambiamenti delle ontologia creando e definendo differenti versioni della stessa, ciò comporta la presenza di un numero elevato di copie della stessa ontologia. Nasce quindi la necessità di sviluppare algoritmi che consentano al progettista di riconoscere e distinguere le diverse versioni a sua disposizione per garantire la realizzazione e il mantenimento delle ontologie. Come nell'approccio basato sull'evoluzione, la propagazione dei cambiamenti necessari all'ottenimento

della nuova versione dovrebbe essere esaminata in dettaglio anche se la nuova versione dell'ontologia ottenuta con il l'approccio qui esaminato si può considerare a tutti gli effetti come una nuova ontologia, che può anche non avere dei punti di contatto con la versione precedente. Questo approccio è molto vantaggioso nel caso di ontologie ad uso distribuito, poiché agevola notevolmente la fase di aggiornamento dell'ontologie, dando la possibilità ad applicazioni che non necessitano dell'implementazione del cambiamento di continuare a lavorare con la precedente versione.

Quando si parla di relazioni tra diverse versioni di una ontologie bisogna esaminare tre aspetti[21]:

- ✓ Differenziare le versioni delle relazioni dalle concettualizzazioni delle relazioni interne all'ontologia;
- ✓ Discutere nel caso di aggiornamento dell'ontologia se ci sono delle discrepanze tra i cambiamenti nelle concettualizzazioni e quelli nelle specificazioni;
- ✓ Definire le vie per applicare questi aggiornamenti sull'ontologia, pacchetti di cambiamento, packaging of changes.

3.2 Riduzione delle dimensioni di una ontologia

Un aspetto dell'ontologia è quello di fornire all'utente uno schema concettuale di un dato dominio esaustivo e rigoroso, privo di informazioni inutili e ridondanti.

L'eliminazione da una ontologia di concetti superflui è stata affrontata da Jordi Conesa Caralt [24], in sede di tesi, in un progetto presentato al "UML' 04 Doctoral Symposium" di Lisbona e supportato dal Ministerio de Ciencia y Tecnologia and FEDER.

3.2.1 Dall'ontologia di base allo schema concettuale

Secondo Conesa, la maggior parte degli schemi concettuali dei Sistemi Informativi sono generati dal nulla, con uno spreco di tempo e risorse, in questo progetto ha quindi sviluppato un algoritmo per formulare uno schema concettuale riutilizzando le conoscenze già acquisite e presenti in ontologie più grandi. L'idea è

quindi quella di avere da un lato una l'ontologia di base più generale (O_G), contenente le fondamenta su cui viene costruito lo schema, dall'altro lato si ha lo schema concettuale finale visto come specializzazione della ontologia stessa.

Usare una ontologia di base di grandi dimensioni per crearne una specifica, richiede di l'eliminare, nel modo più automatico possibile, tutti gli elementi ritenuti irrilevanti per lo schema concettuale finale. Come mostrato dalla figura 3.6, l'algoritmo proposto da Conesa prevede l'impiego di tre azioni.

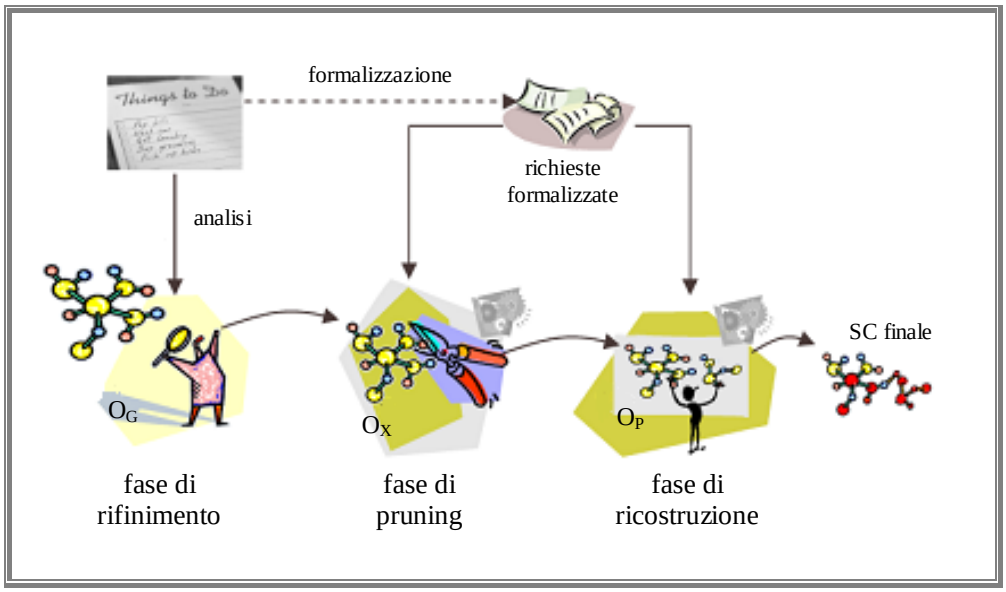

Figura 3.6: Tre fasi e struttura del progetto

Il primo passo è quello del "**raffinamento"**. L'ontologia di base può a volte non contenere alcuni degli elementi richiesti dal Sistema Informativo (classi, relazioni, attributi e vincoli di integrità), per ovviare a questo problema, il progettista esegue, quindi, un insieme di operazioni additive per creare gli elementi necessari all'ontologia finale. Normalmente un O_G è di dimensioni così elevate da richiedere l'utilizzo di più tecniche per supportare il suo raffinamento in un modo maggiormente usabile e veloce, il risultato di questo procedimento è ciò che viene chiamata ontologia estesa (O_X).

O_X ha tutte la conoscenza necessarie al sistema informativo, ma a causa delle sue dimensioni troppo elevate non può essere utilizzata come schema concettuale (si possono avere anche 10^3 classi con 10^3 relazioni). La maggior parte degli concetti

contenuti in O_X sono irrilevanti per il sistema informativo, devono quindi essere eliminati attraverso una azione di **"pruning"** (o di "**potatura"**), quindi O_X viene potata da questi elementi, ottenendo una ontologia ridotta (pruned ontology o O_P).

L'ultimo passo è quello della **"ricomposizione".** Il progettista ha in O_P tutti gli elementi rilevanti per descrivere il sistema informativo, tuttavia alcuni di questi posso ancora risultare, in un secondo confronto, ridondanti e quindi soggetti ad eliminazione, mentre altri possono richiedere un processo di ristrutturazione per poter avere un schema concettuale compatto, esaustivo e rigoroso. Il risultato di questa operazione è lo schema concettuale finale, espressione sintetica del sistema informativo.

In figura 3.7 è mostrata un esempio di attività di pruning e ricostruzione.

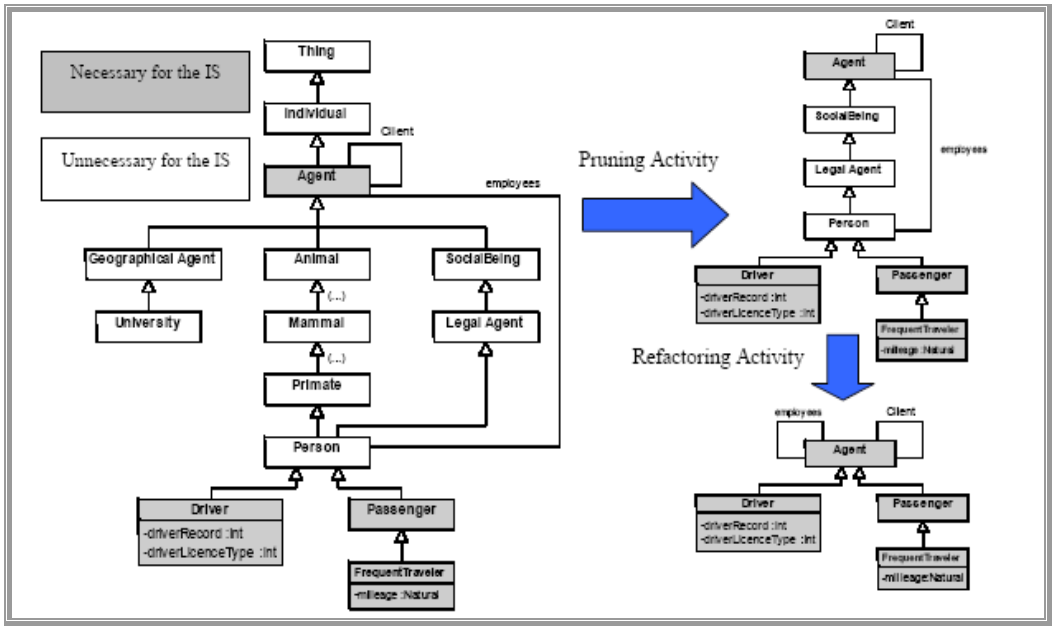

Figura 3.7: Esempio di attività di pruning e rifinitura

In questa tesi verrà analizzata più nel dettaglio la fase di pruning, per poter studiare quali possono essere le possibili applicazioni nel sistema MOMIS, in particolare verranno riportate le scelte implementative fatte da Conesa per sviluppare un metodo di pruning su ontologie OWL [24,25,26] .

3.2.2 Attività di Pruning

Come già abbiamo visto, dopo la fase di raffinamento l'ontologia ottenuta O_{xs} solitamente non è direttamente utilizzabile, perché troppo grande con una presenza notevole di concetti irrilevanti per il sistema informativo di riferimento. Quindi, l'attività di potatura deve identificare e cancellare i questi concetti ridondanti.

La potatura di una ontologia può essere separata a sua volta in due fasi:

Fase di selezione → Per eliminare gli elementi irrilevanti, è necessario conoscere quali elementi sono rilevanti per il domino di interesse, in questa fase si definiscono i concetti rilevanti.

Fase di pruning → Questa fase utilizza le informazioni raccolte dallo step precedente per eliminare gli elementi irrilevanti dal dominio.

Un esempio di un generico algoritmo di pruning senza separazione può essere: "limare tutte i concetti dove nessuno dei suoi sinonimi può essere selezionato come rilevante". In questo caso il concetto di sinonimia, proprio dell'ontologia di riferimento, non può essere riutilizzato per tutte le ontologie. Lo stesso esempio sarebbe stato diverso separando le due fasi di selezione e pruning: "selezionare i sinonimi dei relativi elementi" ed "eliminare gli elementi non rilevanti". E' palese come questo metodo sia più generico rispetto al precedente. Questa separazione permette di riutilizzare algoritmi abbinando diverse tecniche.

Concetti di interesse diretto CoI

Un concetto (dove per noi concetto si intende sia un classe ma anche una relazione o un attributo), è di interesse diretto in una data ontologia se i suoi progettisti e utenti sono interessati a descrivere la popolazione del sistema oppure ad aggiunge nuove informazioni, l'insieme dei concetti di interesse nell'ontologia di base è nominato C_OI.

Se una proprietà[16] è contenuta in C_OI allora sia il suo dominio che il range deve essere incluso dentro a C_OI. Formalmente, C_OI si definisce completo se per ogni proprietà inclusa il suo dominio e range sono inclusi in C_OI. Nell'ontologia di base ci sono molti concetti che non appartengono a C_OI ma che generalizzano quelli inclusi in C_OI. Questi risultano essere comunque di interesse, anche se non diretto, perché possono contenere vincoli che hanno effetti sulle istanze dei concetti appartenenti a C_OI. Per questo motivo viene definito come $G(C_OI)$ l'insieme di questi concetti, cioè l'insieme dei concetti che completano quelli appartenenti a C_OI e le loro generalizzazioni. Nell'esempio se C_OI = (*Agent*) allora $G(C_OI)$ = (*Agent, Individuals, Thing*).

Il metodo di pruning necessita di conoscere a priori questo insieme, tuttavia questa fase è indipendente da quella precedentemente studiata, per questo motivo verrà trattata in maggiore dettaglio nella sessione successiva.

Regole base per creare un algoritmo di pruning

Dati i concetti di interesse diretto C_OI e l'ontologia di base rifinita O_x, le regole base per creare un algoritmo di pruning per ottenere l'ontologia ridotta O_P sono:

a) Gli elementi in O_P devono essere un sottoinsieme di O_X, non ci devono essere elementi aggiuntivi.

$$(O_P \subset O_X)$$

b) O_P deve include tutti i concetti di interesse diretto contenuti in C_OI

$$(C_OI \in O_P)$$

[16] Per proprietà, in una ontologia OWL, si intende un tipo di relazione binaria e diretta. Il domino e il range possono essere esplicitamente definiti con un assioma, tuttavia la loro definizione non è esplicitata quindi è la classe *Thing* che deve assumere questa regola. Le proprietà possono essere di tipo: *ObjectProprieties*, mettono in relazione classi; *DataTypeProprieties*, mettono in relazione una classe con un *Datatype*[25].

c) Se C_1 e C_2 sono due concetti in O_P e sono in legati tra loro da una relazione di generalizzazione, diretta o indiretta, in O_X, allora la loro relazione deve esistere in O_P.

d) Se c è un concetto, i una istanza di O_P ed esiste tra loro una relazione di *InstanceOf* in O_X, allora questa relazione deve esistere in O_P.

$$\forall\, c,i\ (c \in O_{\underline{P}} \wedge i \in O_{\underline{P}} \wedge \text{InstanceOf}(i,c) \in O_{\underline{X}} \rightarrow O_P.$$

e) tutti i vincoli definiti in O_X relativi a concetti inclusi in $G(C_OI)$ devono essere inclusi in O_P. In altre parole, se un vincolo di O_X include uno o più concetti non appartenenti a questo insieme non deve essere inserito in O_P.

f) O_P deve essere una rappresentazione consistente.

g) O_P deve essere minima, cioè con l'eleminazione di un concetto vengono a mancare (b) e (f).

Algoritmo di pruning

Per ottenere O_P l'algoritmo prevede l'implementazione di quattro passaggi [25]:

- ✓ Potatura dei concetti e relazioni irrilevanti, si ottiene così O_1.
- ✓ Potatura dei parents irrilevanti, si ottiene così O_2.
- ✓ Potatura delle path non necessarie, si ottiene così O_3.
- ✓ Potatura degli individui orfani, il risultato finale che si ottiene è O_P.

Per meglio mostrare come vengono implementate queste fasi ci avvaliamo dell'esempio mostrato in figura 3.7.

Potatura dei concetti e relazioni irrilevanti

I concetti di interesse diretto che devo essere contenuti in O_P sono raccolti in C_OI, mentre $G(C_OI)$, come sopraccitato, raggruppa tutti i concetti che sono direttamene o indirettamente collegati a C_OI. In questa fase vengono eliminati tutti i concetti che non fanno parte di $G(C_OI)$. L'eliminazione di un concetto include l'eliminazione di tutte le generalizzazioni e relazioni di classificazione cui esso è interessato, e, tuttavia, è da sottolineare che in questa fase non vengono eliminate

istanze, queste infatti possono essere collegate ad altri concetti appartenenti a $G(C_OI)$. Similarmente, vengono eliminati i vincoli di O_X ritenuti non rilevanti per l'ontologia finale perché contenenti uno o più concetti non inclusi in $G(C_OI)$, al termine di questa fase si ottiene una ontologia chiamata O_1.

Nell'esempio mostrato in figura 3.7 *GeographicalAgent* e *University* non appartengono a *G(CoI)*, per questo vengono eliminate.

Potatura dei parents irrilevanti

Il contenuto di O_1 è esattamente uguale a *G(CoI)*, purtroppo non tutte le informazioni racchiuse in questo insieme sono essenziali per definire O_p. In questa fase vengono definiti i concetti strettamente necessari (*NeededConcepts*), quelli dati dall'unione dei concetti di interesse diretto e i concetti che appaiono nelle definizioni dei vincoli rimanenti, i concetti non appartenenti a questi insiemi possono essere potenzialmente eliminati. In particolare, sono soggetti a potatura tutti i concetti parents di *NeededConcepts* che non sono figli di altri concetti appartenente all'insieme. Come nella prima fase, l'eliminazione di un concetto prevede la potatura che di tutte le generalizzazioni e classificazioni ad esso associate. il risultato finale è O_2.

In figura 3.7, i parents di *Agent* (*Thing* e *Individual*) sono eliminati, perché non sono parents di un concetto necessario (*Agent*), e nessuno dei loro parents è un concetto necessario.

Potatura delle path non necessarie

In molti casi l'ontologia O_2 può contenere diverse catene di generalizzazione (*Generalization Path*), scopo di questa fase è quello di eliminare i percorsi tra concetti ritenuti ridondanti, si definisce, quindi, una *Generalization Path* tra due concetti C_1 e C_n una catena che:

a) C_1 e C_n sono due concetti di O_2;

b) IsA^+ (C_1, C_n) e

c) il path contiene due o più relazioni $IsA(C_1,C_2),\ldots, IsA(C_{n-1},C_n)$.

Un percorso IsA(C_1,C_2),…., IsA(C_{n-1},C_n) tra C_1e C_n è potenzialmente ridondante se nessuno dei concetti intermediari C_1,….C_{n-1}

d) appartiene all'insieme CoI ∪ CC(O2)[17];

e) è un super-concetto o sub-concetto di una relazione di generalizzazione.

Nel nostro caso di studio vengono individuati due percorsi tra Agent e Person. Uno di questi viene eliminato perchè formato da classi che non sono concetto necessario per l'ontologia finale: (*Person→ Primate → Mammal → Animal → Agent*). Mentre l'altro, più breve, viene mantenuto: (*Person → Legal Agent → Social Being → Agent*).

Potatura degli individui orfani

Nei passaggi precedenti sono state eliminati i concetti dell'ontologia, in questa fase invece, vengono eliminate le istanze dell'ontologia tali per cui i classificatori di appartenenza (classi o proprietà) sono stati eliminati nelle fasi precedenti, quando una istanza di una classe viene ridotta, tutti i valori delle varie caratteristiche e le relazioni di *SameAs* devono essere eliminate.

Il risultato di questa fase è l'ontologia finale O_p.

3.2.3 Tecniche per individuare l'insieme C_OI

Prima di applicare l'algoritmo per potare l'ontologia di base trasformandola in O_p, è necessario stabilire quali sono i concetti che devono entrare a far parte dello schema finale, definendo l'insieme C_OI. Come già in precedenza menzionato, nel metodo in cui vengono selezionati i concetti di interesse diretto è indipendente da quello utilizzato per la fase di potatura dell'ontologia, per questo motivo la scelta può ricadere tra varie tipologia di tecniche, come si può vedere dalla figura 3.8.

[17] CC è l'abbreviazione di *Constrain Concepts*, dove viene definito CC(*ic*) un insieme di concetti limitati da un concetto di integrità *ic*, quindi CC(O), dove O è una ontologia, è l'insieme di tutti i concetti limitati da tutte le relazioni di integrità appartenenti all'ontologia [26].

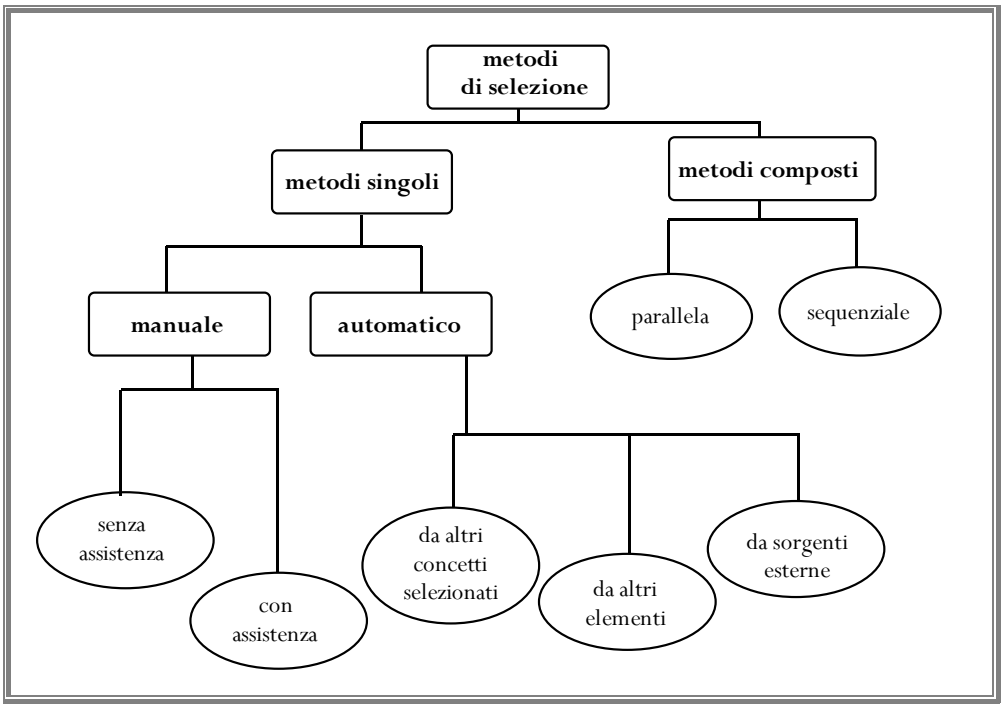

Figura 3.8: Metodi di selezione per ricercare i concetti di interesse diretto

Le tecniche a disposizione per selezionare i concetti da inserire all'interno dello schema finale possono essere classificate in base alla granularità in metodo a selezione individuale o composta.

Metodi singoli di selezione

In questo caso viene applicato un singolo criterio di selezione che può essere manuale o automatico. Nei criteri manuali è il progettista che seleziona mano a mano gli elementi della O_X secondo la sua esperienza e conoscenza dell'ambiente da rappresentare, la classificazione manuale a sua volta può essere con o senza assistenza, il progettista può quindi scegliere se avvalersi o meno di un sistema di supporto elettronico; la selezione manuale con assistenza è la tecnica spesso utilizzata nei metodi di selezione combinata.

Nella selezione automatica invece i concetti vengono selezionati automaticamente dal sistema, le informazioni necessarie per individuare i concetti rilevanti possono essere presi:

- ✓ Da altri concetti: i concetti di interesse diretto già precedentemente individuati possono essere usati per selezionare altri concetti.

- ✓ Da altri elementi dell'ontologia: a volte può succedere che non siano concetti a portare alla selezione di altri concetti ma altri elementi sempre inclusi nell'ontologia. Questa tecnica, spesso dimenticata negli algoritmi di selezione e pruning, rimane comunque uno strumento molto interessante perché spesso le informazioni per la ricerca di concetti di interesse diretto possono essere trovate anche da istanze dell'ontologia, vincoli di integrità o relazioni di generalizzazione.
- ✓ Risorse esterne: i concetti di C_OI si possono trovare utilizzando informazioni interne che giacciono in sorgenti esterne, questo è uno dei metodi maggiormente utilizzato. In questo caso i concetti possono essere estrapolati attraverso un text-minig all'interno di vari documenti oppure in base a requisiti funzionali richiesti dal sistema informativo, formalizzati da un sistema di operazioni [27].

Metodi composti di selezione

La tendenza è comunque quella di non utilizzare una solo tecnica ma di comparare più risultati di ricerca combinando tra loro vari metodi di selezione in modo parallelo o sequenziale. Nel primo caso, l'approccio è quello di eseguire le tecniche scelte contemporaneamente e di comparare gli output attraverso un algoritmo automatico che sottolinei le differenze e le somiglianze, determinando così, quali concetti includere nell'insieme C_OI e quali scartare. Sebbene questa tecnica non sia tra le più usate, fornisce un potenziale strumento di ricerca. D'altra parte spesso questo approccio necessita di una grande partecipazione del progettista, questo può rappresentare un ostacolo in caso di ontologia di notevoli dimensioni. Nel secondo caso, invece, i metodi vengono eseguiti in modo sequenziale, quindi l'output di uno diviene l'input del successivo, questa è in assoluto la tecnica più utilizzata.

Vengono qui di seguito riportati alcuni approcci composti adottati in letteratura.

Knowledge Bus

Il Knowledge Bus è un approccio che consente di creare un database di un sistema informativo usando CyC[18]di un ontologia di base [28]. In questo approccio viene utilizzata una fase di pruning per eliminare i concetti contenuti in CyC non rilevanti per il sistema informativo da rappresentare. Parte dell'algoritmo di pruning prevede una fase di selezione di questi concetti. L'approccio adottato, di tipo sequenziale, prevede tre fasi:

f) Una selezione manuale non assistita, il programmatore seleziona i concetti di interesse diretto da inserire all'interno C_OI.

g) Una selezione automatica per ottener i concetti accessibili da quelli contenuti in C_OI attraverso relazioni.

h) Una selezione automatica che selezioni tutte le relazioni dove partecipano concetti contenuti selezionati nei precedenti step.

Swartout et al.

Questo è un approccio utilizzato su grandi ontologie per definire il dominio dell'ontologia. Come nel primo caso il processo inizia con una seleziona manuale dei concetti rilevanti per un dominio da parte dell'utente. Poi per ogni concetto selezionato un sistema automatico seleziona gli elementi definiti attraverso la radice dell'ontologia e il concetto. In seguito il progettista può selezionare i sottoalberi dell'ontologia che sono vicini di quelli selezionati. Questo tipo di processo implementa quindi un approccio composto di tipo sequenziale:

a) Una selezione manuale non assistita. Il programmatore seleziona i concetti di interesse diretto da inserire all'interno C_OI. (Similare alla fase iniziale del metodo Knowledge Bus)

[18] Il progetto Cyc di Doug Lenat (da enCYClopedia http://en.wikipedia.org/wiki/Cyc) al MCC a Austin, Texas, è iniziato nel 1984 con l'obiettivo di sviluppare un sistema specializzato ed universale, in grado di capire e parlare la lingua ordinaria e di individuare le violazioni del senso comune più prontamente rispetto ad una persona.

b) Una selezione automatica per ottenere quali sono i parents dei concetti contenuti in C_OI.

c) Il progettista seleziona manualmente i sottoalberi dell'ontologia vicini a quelli selezionati.

Conesa e Olive

Il vantaggio del processo di selezione adottato da Conesa risiede nell'obiettivo di sviluppare uno schema concettuale di un sistema informativo partendo da una via semiautomatica e da un'ontologia di grandi dimensioni. L'approccio analizzato è formato da tre fasi: il rifinimento dell'ontologia dai concetti mancanti, la potatura dei concetti ridondanti e la ristrutturazione dello schema finale. In particolare la fase di potatura si compone di due azioni: selezione dei concetti di interesse diretto, eliminazione delle informazioni non necessarie.

Anche questo approccio sviluppa un metodo di selezione composto sequenziale, ma a differenza di quelli descritti precedentemente, parte da una prima fase automatica passando poi all'intervento del progettista. Il metodo si articola:

d) Una selezione automatica per stabilire tutti i concetti che si riferiscono ai requisiti del sistema informativo.

e) Una selezione manuale per includere nell'insieme creato nel primo step quei concetti ritenuti necessari per il progettista, ma che il sistema non è riuscito a rilevare in modo automatico.

Text-to-Onto

Questo metodo ha come obiettivo quello di definire un dominio di una ontologia partendo dalla ontologia di base. Text-to-Onto utilizza un algortimo di text-mining per identificare i concetti rilevanti per un dominio. Utilizza due set di documenti, il primo contiene un insieme di testi riguardanti il dominio di interesse, mentre il secondo indipendente dal dominio. Per definire se un elemento dell'ontologia di base è rilevante per il dominio, il metodo usa l'algoritmo di text-mining per calcolare la frequenza di appartenenza nei due set di documenti; quindi l'idea di base è quella che un concetto è rilevante se è ripetuto più volte in un

generico documento. Questo è un tipo di metodo individuale di selezione automatica che si avvale di risorse esterne; non è un metodo composto che può essere ulteriormente suddiviso, poiché è formato da operazioni primarie di selezione.

Un approccio simile a quello sopra descritto è stato implementato nel progetto US FAO AOS [29], il quale proponeva un algoritmo per acquisire una ontologia iniziale da una collezione di documenti, in particolare riutilizzando un thesaurus già esistente. Molte ditte stilano, per interessi di tipo commerciali, una tassonomia dei prodotti e dei servizi, utilizzando una documentazione caratterizzata da una corretta terminologia, che può essere impiegata per costruire un'ontologia iniziale. Come già sottolineato, l'utilità di creare una ontologia nella gestione dei documenti trova spiegazione nella capacità di recuperare i documenti, caratteristica strettamente legata alla assonanza dei termini ontologici con le keywords presenti nei documenti trattati. Sfortunatamente, questa assonanza non è sempre presente nei thesauri. Questo progetto ha quindi sviluppato un approccio di tipo pruning per rimuovere i termini non necessari dal thesaurus attraverso un'analisi euristica dei termini contenuti nella collezione dei documenti, in questo modo si ha la certezza che l'ontologia sia focalizzata sulla voluta collezione di documenti.

L'input del pruning è quindi sempre un vocabolario già esistente (un ontologia di piccole dimensioni, thesaurus o tassonomia) che costituisce una concettualizzazione generale relativa al domino di interesse. Come mostrato dalla figura 3.9, lo scopo della fase di pruning è quello di estrarre in modo automatico tutti i sottoinsiemi di concetti che sono relativi al dominio.

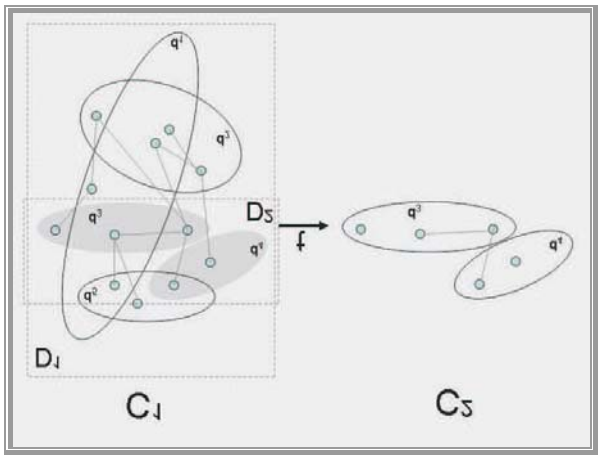

Figura 3.9: Estrazione dei concetti, fase di pruning

La decisione che possa o meno essere rilevante per il dominio si basa su un'analisi euristica dei documenti che sono contenuti. L'analisi euristica opera sul calcolo della frequenza delle parole che sono state estratte dai documenti. Gli strumenti per effettuare queste misurazione sono proprie del text-mining, e si possono limitare al semplice conteggio dell'occorrenza di una parola all'interno del documento, oppure della sua occorrenza all'interno di un set di documenti. Vengono quindi creati due set di documenti per l'estrazione dei termini: uno contenente documenti specifici per il dominio, l'altro contenente documenti generici. Ciò permette di considerare la relativa importanza dei termini di dominio (o generici termini) nel processo di pruning.

Il punto centrale di questi progetti è quello di identificare i due set di documentazione. Per quanto riguarda l'esempio specifico, l'insieme dei documenti che rappresenta il dominio di interesse e che contiene i concetti rilevanti può essere scelto con attenzione dagli esperti in materia, diventa quindi una scelta deliberata. Mentre il set di documenti generici viene raccolto da testi usati nella comunità di Information Retrieval, quali CELEX, o da archivi pubblici di notizie.

3.3 Mapping di ontologie

Il mapping di ontologie è diventato un punto chiave negli ultimi anni. Le motivazioni di questo interesse lo si può ricercare in ambito commerciale aziendale: fusione di società o riorganizzazioni comporta spesso la fusione dei propri sistemi

informativi, sempre più spesso gestiti da ontologie, inoltre a volte si ricorre alla fusione di numerose ontologie per ottenerne una di migliore qualità. Tuttavia anche il merging a tempo di esecuzione può essere cruciale specialmente se si tiene conto del fatto che le ontologie sono diventate estremamente comuni nel WorldWideWeb dove forniscono semantica alle annotazioni presenti nelle pagine di internet. Come ampliamente discusso 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.

Come conseguenza delle motivazioni appena espresse, sono nati numerosi tool per eseguire il mapping di ontologie, basati su due approcci principali collegati tra loro:

fusione (merge) → viene creata un'unica ontologia aderente alle informazioni contenute da quelle di partenza[30].

allineamento → vengono stabilite dei link di relazione tra le ontologie[31].

3.3.1 Allineamento di ontologie

In generale si parla di **mapping** di ontologie quando vengono create tra queste delle corrispondenze nel vocabolario e negli assiomi in esse contenuti, avvalendosi di strutture matematiche, come mostrato in figura 3.10.

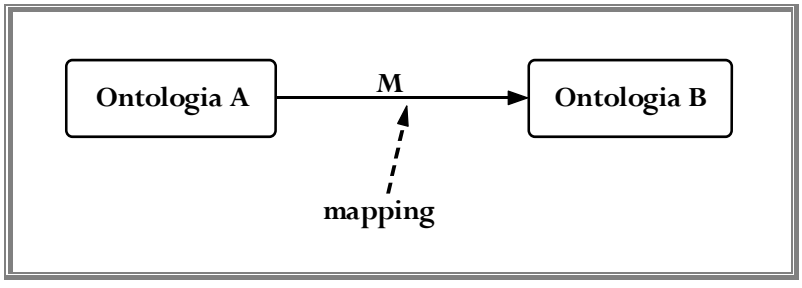

Figura 3.10: Mapping tra due ontologie

L'**allineamento** di ontologie mostrato in figura 3.11, è una tipologia di mapping di ontologie, che mette in relazione una ontologia intermediaria, chiamata "Articulation ontology", con le sorgenti ontologiche da allineare.

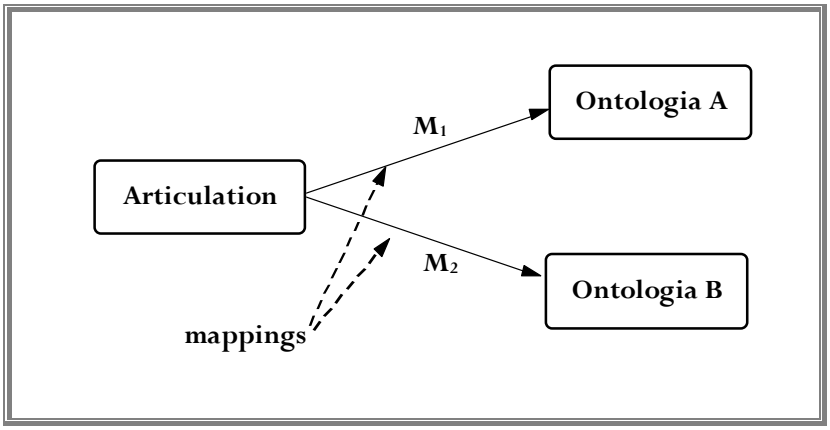

Figura 3.11: Allineamento di ontologie con Articulation Ontology

L'allineamento crea quindi un livello superiore di legami di relazione, mantenendo separate le ontologie di base, che rimangono completamente autonome tra di loro sia durante che dopo il processo di allineamento. Questa tecnica viene solitamente applicata nel caso di ontologie con schemi del domino complementari con quelli del livello superiore. Per esempio, un specifico dominio di una ontologia può essere allineata ad una ontologia centrale in CyC stabilendo dei legami in CyC's upper- and middle- level di ongologie. Questi specifici domini che non possono essere integrati con la conoscenza di base di Cyc, rimangono separati dalle ontologie che contengono riferimenti a concetti di questo tipo. Questo è uno specifico esempio di allineamento di ontologie riguardante la creazione di collegamenti tra agenti che utilizzano differenti tipi di modelli di ontologie. Infatti una volta allineate le ontologie e determinato i concetti minimi necessari per creare una comunicazione sufficiente tra gli agenti (cioè le intersezioni delle concettualizzazioni nei modelli in oggetto ritenute necessarie) gli agenti possono interagire tra loro sfruttando queste proprietà.

Uno esempio di allineamento di due ontologie è quello eseguito dal Formal Concept Analysis (FCA) è descritto nell'articolo di De Sauza pubblicato dall'IEEE '04 [31]. Il progetto utilizza il thesaurus POSET per l'allineamento delle ontologie e crea una struttura a griglia, visibilmente immediata, per comparare e ricavare le

sovrapposizioni, e quindi le differenze, dei due schemi. Per calcolare e valutare la similarità tra elementi dell'ontologia, si avvale di un algoritmo di soglia basato su due funzioni; la prima relativa alle caratteristiche comuni, e non, presenti nei termini del thesaurus, la seconda, invece, relativa alla struttura della rappresentazione a griglia dell'ontologia..

Un elenco su quali possono essere le tecniche adottate per stabilire relazioni tra concetti di ontologia, viene descritto nell'algoritmo di merge implementato nel progetto DOGMA [30], che verrà approfondito nel paragrafo successivo. L'algoritmo richiama, al suo interno, una fase di allineamento tra ontologie, formalmente definita come l'interpretazione dei termini che formano la base lessicale di una ontologia Ω_s rispetto a quelli di un'altra ontologia Ω_t. Questo processo avviene attraverso due passaggi, in seguito descritti, che portano alla identificazioni dei legami di mapping e quindi all'allineamento delle ontologie in esame.

Trovare regioni di intersezione dei domini

Il primo passaggio necessario quando si parla di comparazione, è quello di specificare quali sono le parti delle ontologie che corrispondono alle intersezione dei loro rispettivi domini.

Trovare concetti equivalenti tra le ontologie

Questo passaggio estrae ed esamina le varie tipologie di eterogeneità che ci possono essere tra concetti ritenuti semanticamente equivalenti, cioè presenti in entrambe le ontologie ma indicati con termini differenti. Per ovviare al problema di una errata corrispondenza tra ontologie (a causa di errori di digitazione, ambiguità dei termini, utilizzo di differente terminologia per indicare lo stesso concetto, e così via) la fase di comparazione dovrebbe fare riferimento ai significati del concetto, invece che le etichette dei temimi che li descrivono. Il grado di similarità tra due concetti c_1 e c_2 è misurato attraverso funzioni che riducono il calcolo al confronto con i termini del linguaggio naturale. Viene qui di seguito proposta una lista delle tecniche per poter determinare il grado di similarità fra i termini delle ontologie, attraverso una comparazione con il linguaggio naturale. Tutte queste tecniche sono basate sulle

differenze sintattiche fra i termini, non considerano il valore semantico, è quindi necessaria una successiva fase di analisi critica per l'interpretazione dei risultati.

- ✓ **Porter Stememer**. Implementato solitamente come parte del processo di normalizzazione nei sistemi di Information Retrieval, lo stemming rimuove le estensioni morfologiche e inflessioni poste alla fine dei termini. Un suo utilizzo, ad esempio, è quello di ridurre la forma plurale di una parola alla propria base singolare (papers → paper). Il simbolo → indica com'è la parola prima e dopo lo stemming.

- ✓ **Levenshtein Distance**. Questa misura, chiamata della "Edit Distance", indica la distanza di similarità fra due termini c_s e c_t, vista come numero di cancellazioni, inserimento o sostituzioni richiesti per trasformare c_s in c_t, dove maggiore è il valore risultante maggiore è la differenza tra i termini.

$$sc(c_s, c_t) = \frac{max\_transations - levenshtein\_distance}{max\_transations}$$

Max_Transations è il massimo numero di transazioni necessarie per arrivare a c_t partendo dal c_s ed è uguale a max(length(s),length(t)). Non sempre però questo parametro da analisi corrette, prendiamo ad esempio il termine "prof" e "professore", sono entrambi sinonimi uno dell'altro ma hanno un basso score di similarità.

$$sc(c_s, c_t) = \frac{10-6}{10} = \frac{4}{10}$$

In questo esempio Levenshtein Distance e' sei perché abbiamo da aggiungere sei lettere a "prof" per ottenere il termine "professor".

- ✓ **Longest Common Prefix/Suffix**. Il risultato di similarità fra 2 termini c_s e c_t, può essere calcolato come la lunghezza del prefisso o suffisso comune più lungo, definito come:

$$sc(c_s, c_t) = \frac{length \quad longest \quad common \quad prefix(c_s, c_t)}{\min(length(c_s), length(c_t))}$$

Richiamando lo stesso esempio del Levenshtein Distance, il prefisso comune più lungo fra "prof" e "professore" e' quattro, questo risultato ha un risultato di somiglianza pari a uno.

- ✓ **Longest Common Substrign**. Questo metodo individua la sottostringa comune più lunga tra c_s e c_t, cioè l'insieme più lungo di caratteri in ordine che compare in c_s e c_t, come mostrato dalla seguente formula.

$$sc(c_s, c_t) = \frac{length \quad longest \quad common \quad substring(c_s, c_t)}{\min(length(c_s), length(c_t))}$$

L'algoritmo è una versione semplificata del Levenshtein Distance, se si considerano la stringa "ricerca dipartimento" e "ricerca dipamento", nonostante l'ultimo termine, il risultato applicando questa formula è di perfetto score.

- ✓ **Metaphone Algorithm.** Questo metodo mette in relazione i termini su una base probabilistica di equivalenza. Il metaphone Algortihm riduce ogni stringa di input a caratteri in codice metafonici, utilizzando regole fonetiche relativamente semplici. Alcuni esempi: "university" e "universities" si trasformano in UNFRST e UNFRSTS, mentre due termini come "faculty" e "faculties" vengono associati allo stesso codice FKLT. Lo score tra due termini viene quindi espresso nella seguente equazione:

$$sc(c_s, c_t) = \frac{lcs(MC(c_s), MC(c_t))}{\min(MC(c_s), MC(c_t))}$$

dove MC è la lunghezza del codice metafonico e LCS indica lo score del Longest Common Substring.

La similarità globale fra due termini è la media pesata di tutte i valori dei distinti approcci di similarità, ognuno dei quali deve maggiori di una data soglia di valore $\alpha \in [0,1]$. Cosi se ogni $sc_i(c_s, c_t) < \alpha$ allora $\omega_i \equiv 0$.

$$sc(c_s, c_t) = \sum_{i=1}^{n} f(sc_i(c_s, c_t)) \text{ con } \sum_{i=1}^{n} \omega_i = 1$$

I valori pesati ω_i mostrano, quindi, come i vari valori relativi ai differenti approcci di similarità sono combinati tra loro, in particolare per poter enfatizzare i risultati alti di similarità, viene assegnato a loro un importante peso, mentre ai risultati più bassi viene assegnato un peso minore, rispettando sempre l'uguaglianza $\sum_{i=1}^{n} \omega_i = 1$.

Alcune informazioni rilevanti per individuare relazioni di equivalenza tra termini possono essere ricavate non per similarità sintattica, ma all'interno delle ontologie stesse, in particolare nel contesto di DOGMA, viene proposta come approccio iniziale la teoria linguistica della distribuzione; l'idea di fondo è che termini che si presentano nello stesso contesto linguistico formale, considerando quindi la loro distribuzione, devono essere considerati semanticamente correlati, in altre parole i termini appartengono alla stessa classe pragmatica. Questo approccio è meglio mostrato nella seguente funzione:

$$\left.\begin{array}{llll} y_i & t_i\, role & co-role\, t_2 \\ y_i & t_i\, role & co-role\, t_3 \end{array}\right\} \Rightarrow t_2 \approx t_3$$

dove *role* e *co-role* sono etichette che esprimono relazioni tra termini. Comunque questa regola non e' sempre applicabile con successo, consideriamo infatti il caso in cui *co-role* non sia stata data o *role* non sia significativamente etichettato. Prendiamo per esempio, "has" e "like_of", questi causano da un lato solo effetti negativi perchè trovano troppi concetti correlati, dall'altro lato tuttavia "has" è spesso associato a proprietà che caratterizzano un certo concetto, necessario per stabilire se due concetti sono equivalenti, in un determinato dominio, sulla base delle proprietà che li accomuna.

Trovare concetti equivalenti conduce ad un primo tipo di regola di mapping che può essere quindi così concettualizzata:

$$\langle c_{id}, \underbrace{\Omega_s.\varphi_a(t_i)}_{c_i} R, \underbrace{\Omega_t.\varphi_b(t_j)}_{c_j}, sc \rangle$$

dove c_{id} sta per un commitmenti-id che identifica in modo univoco la regola di mapping R, che può essere sia di "equivalenza" che di "eguaglianza". sc e' il risultato di similarità tra i concetti $c_i = \varphi_a(t_i)$ nell'ontologia Ω_s e i concetti $c_j = \varphi_b(t_j)$ nell'ontologia Ω_t. Sia nel caso relazione di eguaglianza, con $(sc = 1)$, o di equivalenza, con $(0 < sc < 1)$, i domini dei concetti si possono considerare più o meno identici.

Identificare inter-relazioni tra ontologie

Per poter identificare inter-relazioni tra concetti di differenti ontologie bisogna prima di tutto definire l'esatta semantica di queste relazioni. Viene qui di seguito riportata una lista delle diverse relazioni con semantica predefinita:

- ✓ **SubClassOf:** questa relazione collega un concetto con domini comuni.
- ✓ **Generalize:** questa relazione generalizza in un nuovo concetto due concetti che hanno domini intersecati.
- ✓ **Part of** : questa relazione collega un concetto con una sua parte rappresentata da un altro concetto; questo tipo di relazione è tipica tra concetti con domini disgiunti.
- ✓ **IstanceOf** : questa relazione indica che un oggetto e' un istanza di un concetto.

Una metodologia che permetta di automatizzare la ricerca di inter-relazioni tra concetti di ontologie è chiamata SUMO (Suggested Upper Merged Ontology) [19][30].

[19] http://ontology.teknowledge.com

3.3.2 Merge di ontologie

Il merge di ontologie è applicabile nel caso di ontologie con domini sovrapposti o simili tra loro. Il risultato di questo processo è un'unica ontologia contenente tutte le informazioni di quelle iniziali. La fusione di più ontologie in una unica è spesso utilizzata con lo scopo di estenderne una in particolare tra queste. Consideriamo per esempio Unified Medical Language System (UMLS) e Galen Coding Reference model (CORE), se si unissero tutte queste differenti ontologie si creerebbe un'unica ontologia del dominio medico.

Un possibile algoritmo di integrazione di due ontologie, sperimentato nel progetto DOGMA e discusso nell'articolo di Meersman [30], si può articolare in diverse fasi per mettere in relazione i differenti componenti dell'ontologia, per trovare e risolvere i conflitti tra la rappresentazione e i concetti nel mondo reale, e per effettuare il merge tra ontologie conformi in una globale. In particolare il progetto in esame implementa questo algoritmo in quattro passaggi collegati come mostrato in figura 3.12.

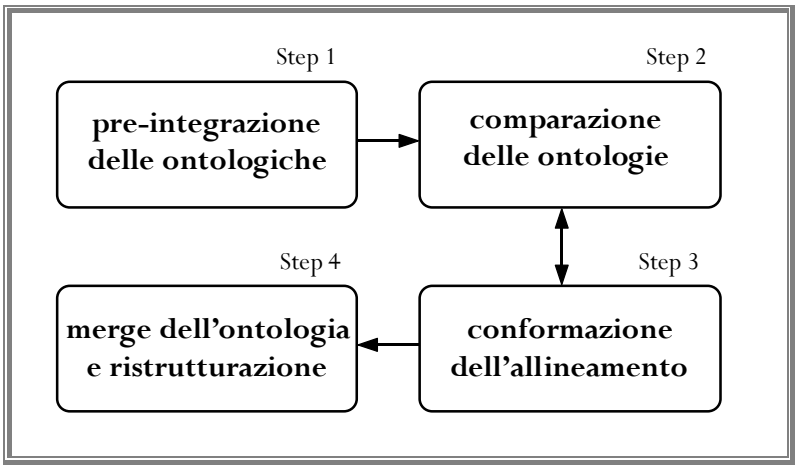

Figura 3.12: Struttura algoritmo di merge del progetto DOGMA

Pre-integrazione delle ontologie

Nella fase di pre-integrazione delle ontologie vengono stabilite la politica generale di integrazione da adottare: quali ontologie integrare, come integrarle, scegliere le strategie e le tecniche da adottare, stabilire l'ordine di integrazione e se necessario assegnare preferenze all'intera ontologia o parte di questa. Questa fase è

formata quindi da un insieme di scelte decisionali prese da un operatore umano sulla base della sua esperienza.

Comparazione delle ontologie

La comparazione delle ontologie effettua un'analisi che determina le correlazioni fra i concetti equivalenti delle ontologie, individua le inter-relazioni tra ontologie. La comparazione di ontologie è sinonimo di "allineamento di ontologie", rimandiamo quindi la discussione al paragrafo successivo.

Conformazione dell'allineamento

Nella fase che controlla la conformità dell'allineamento, viene accertato il corretto adempimento del parametro di consistenza dell'ontologia. Quando una relazione di allineamento viene proposta dal sistema o dal progettista, questa viene istantaneamente comparata con quelle già presenti per evitare di inserire dei conflitti. La validazione di questo inserimento deve controllare che non ci siano cicli tra gli concetti interni all'ontologia e che non ci siano conflitti derivanti da allineamenti di classi inferiori.

Il primo è relativo al caso in cui due concetti c_{s1} e c_{s2} di una ontologia, chiamata Ω_s, sono rispettivamente allineati ai concetti c_{t1} e c_{t2} di un'altra ontologia, chiamata Ω_t ed esiste un ciclo, come mostrato in figura 3.13(a), tale per cui c_{s2} è sottoclasse di c_{s1} e c_{t1}, è sottoclasse di c_{t2}, questa situazione deve essere segnalata come errore perché non corretta.

Un conflitto generato da un allineamento di classi inferiori si riferisce, invece, al caso in cui un concetto c_s di una ontologia Ω_s è allineato con un concetto c_t di un'altra ontologia Ω_t, ed uno dei super-concetti di c_t, chiamato $c_{sovra-c}$,è in relazione con un sub-concetto di c_s, chiamato c_{sub-c}, figura 3.13(b), o viceversa un sub-concetto di c_t è in relazione con un super-concetto di c_s, queste situazioni devono essere segnalate errore dal sistema perché in conflitto.

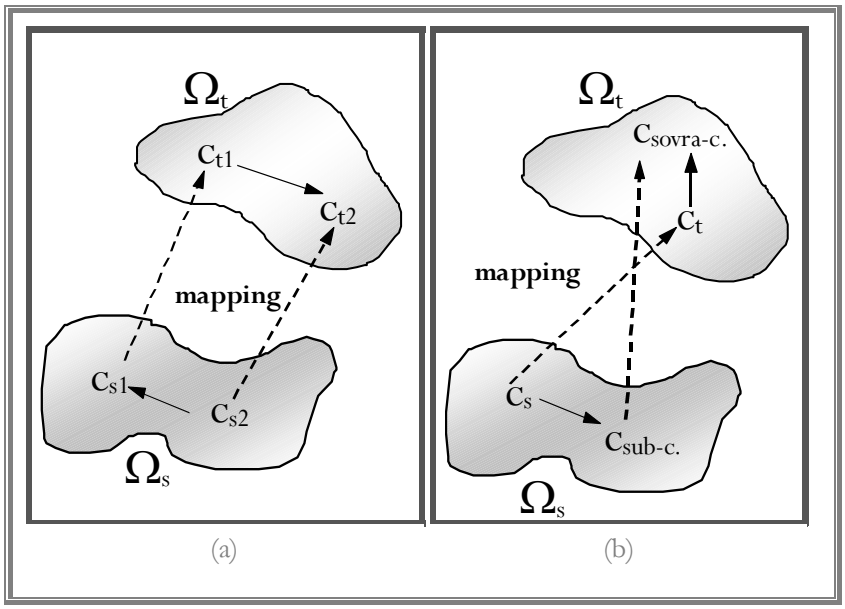

Figura 3.13: (a)Ciclo sulle relazioni di allineamento
(b) Conflitti da allineamenti di classi inferire

La procedura di ricerca su un concetto coinvolge molte convalide euristiche elementari, in primo luogo per controllare che il concetto non sia già presente, secondo, per verificare che la regola di allineamento proposta non sia già esistente.

Merge dell'ontologia e ristrutturazione

In questa fase le sorgenti ontologiche vengono unite seguendo i modelli e le relazioni stabilite nelle fasi precedenti per creare una ontologia globale che chiameremo Ω_{merge}. Le operazioni che si possono individuare nella fase di merge tra ontologie sono principalmente tre:

- ✓ fusione di elementi di informazione equivalenti;
- ✓ specializzazione;
- ✓ generalizzazione.

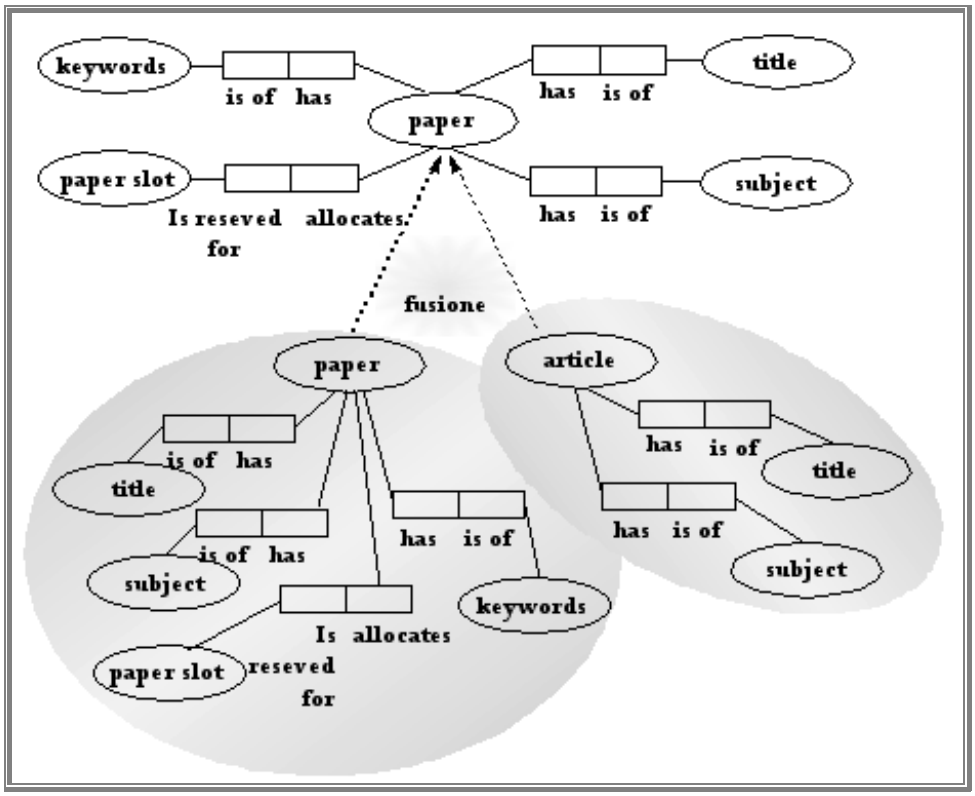

Figura 3.14: Operazione di fusione di elementi di informazione equivalenti

Operazione di fusione

Questo tipo di operazione si applica tra concetti considerati equivalenti. Fondere due concetti richiede una prima fase per nominare, tra i due, quello di riferimento come c_{merge} nell'ontologia Ω_{merge}, concetto contenente il risultato finale della fusione, ed una seconda per comparare le relazioni lessicali dei concetti equivalenti e, nel caso siano equivalenti, copiarle associandole a c_{merge}. Tuttavia, anche nel caso contrario di relazioni lessicali non lessicalmente equivalenti tra loro, in cui non ci siano inter-relazioni con altre dell'ontologia, queste vengono associate a c_{merge}. Questo operazione viene mostrata in figura 3.14, attraverso un esempio in cui i concetti di "article" e "paper" di due ontologie da integrare, relative al dominio bibliografico letterario, devono essere uniti in un unico concetto "paper" nell'ontologia Ω_{merge}. Le relazioni lessicale "paper has/is_of title" e "paper has/is_of subject", dell'ontologia sorgente contenente la classe "paper", sono equivalenti ad "article has/is_of title" e "article has/is_of subject" , dell'ontologia sorgente

contenente la classe "article", devono quindi essere unite e copiate nel concetto "paper" dell'ontologia Ω_{merge}. Le altre relazioni non allineate vengono integrate e copiate.

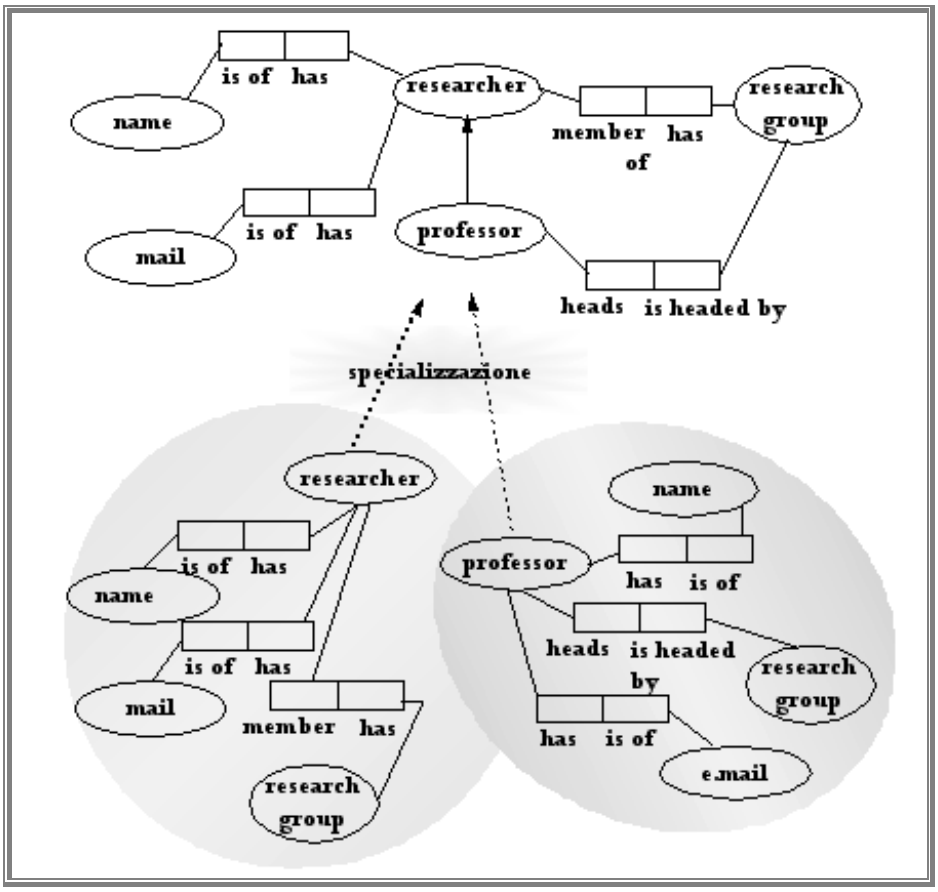

Figura 3.15: Operazione di specializzazione

Operazione di specializzazione

L'operazione di specializzazione viene applicata quando durante la fase di allineamento nasce una relazione del tipo "c_1 *SubClassOf* c_2" i cui concetti appartengono ad ontologie differenti. L'interpretazione che si può dare è questa: sia data una relazione di eredità non-monotona, nella quale, quindi, si può modificare una certa proprietà ereditata. Come mostrato in figura 3.15, supponiamo ci sia un relazione del tipo "Professor SubClassOf Researcher", in questo caso sia le proprietà nelle quali "Professor" è specificato come subconcetto sia quelle che annullano delle proprietà ereditate dal superconcetto "Researcher" vengono scritte esplicitamente,

mentre tutte le proprietà del superconcetto "Researcher" che non sono esplicitamente modificate dal subconcetto vengono implicitamente ereditate.

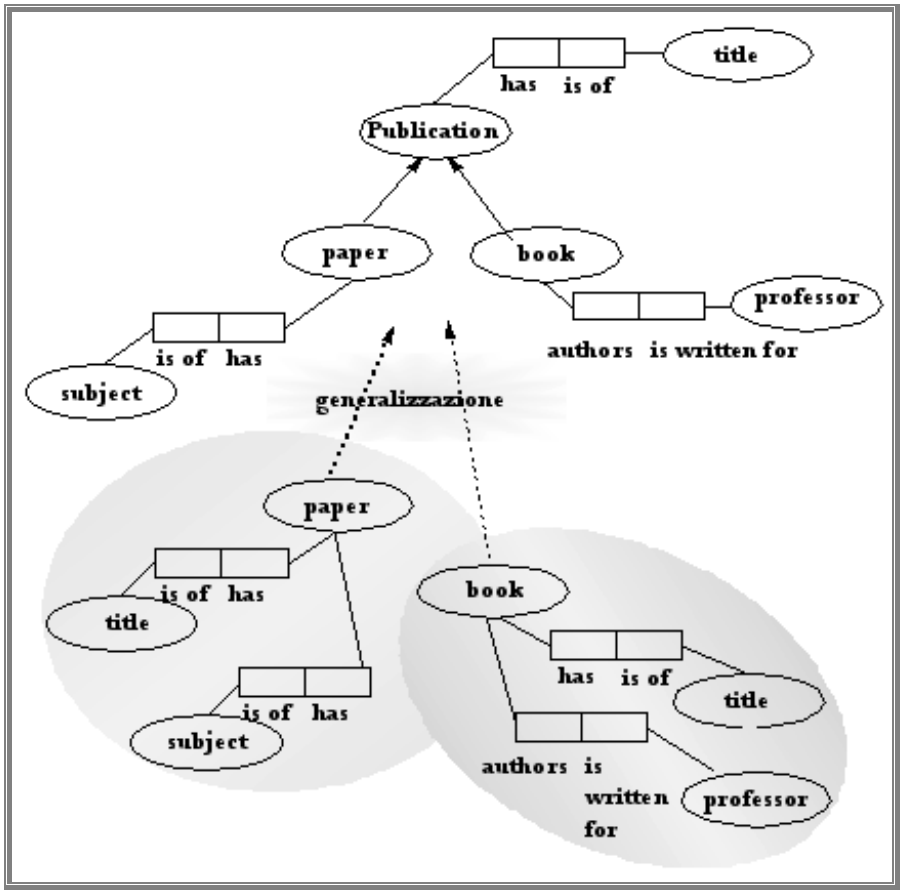

Figura 3.16: Operazione di generalizzazione

Operazione di generalizzazione

L'operazione di generalizzazione tra due concetti c_1 e c_2 si applica quando si evidenzia la necessità di creare un nuovo concetto c_g, tale per cui "c_1 is_a c_g" e "c_2 is_a c_g", si introduce, così, una generalizzazione dei concetti in un altro concetto, attraverso una operazione simile quella di fusione, prima descritta. A differenza di questa, tuttavia, la fusione è relativa solo alle relazioni equivalenti, che vengono associante al nuovo concetto c_g, mentre i due concetti c_1 e c_2 rimangono separati. Un esempio di generalizzazione viene mostrato dalla figura 3.16: "Paper" e "Book" vengono mantenuti separate perché sono due concetti separati uniti in

alcune caratteristiche, mentre le loro relazioni a concetti comuni vengono ereditate in un unico concetto più generale "Publiction".

3.3.3 Due approcci al confronto

Dopo avere presentato, nei paragrafi precedenti, in dettaglio in cosa consistono i due approcci sviluppati per operare un'integrazione fra ontologie, si può ragionare su quanto siano intimamente collegati.

L'allineamento infatti, come abbiamo visto, è una fase molto importante della procedura di merge dove vengono ricavate, attraverso le varie misure introdotte, le similitudini fra differenti ontologie; si può quindi considerare il merge di ontologie come un'estensione del puro allineamento.

La differenza di utilizzo è, di conseguenza, conseguente alle esigenze specifiche dell'integrazione; quando è sufficiente semplicemente stabilire delle corrispondenze fra differenti ontologie l'allineamento è la soluzione preferibile, perchè creando un livello superiore non va a modificare la struttura delle ontologie esistenti. Il suo svantaggio, parlando da un punto di vista computazionale, è che mantenere molte di queste sovrastrutture rallenterebbe di molto l'elaborazione. In questo caso, quindi, è meglio utilizzare il merge, per ottenere una struttura dati più compatta e che permetta una più facile elaborazione.

Concludendo, i due approcci sono praticamente l'uno, il merge, l'estensione dell'altro, l'allineamento, e l'utilizzo dell'uno o dell'altro è una scelta demandata al progettista in base alla complessità del proprio specifico dominio di lavoro.

Per valutare le performance dell'algoritmo implementazione, si può eseguire, per entrambi gli approcci, un confronto con l'allineamento manuale delle stesse ontologie [30,31]. I parametri utilizzanti sono quelli di "*recall*" e "*precision*", comunemente impiegati nei contesti di Information Retrieval.

Indicato con **H** l'insieme degli elementi simili secondo il giudizio ed esperienza dall'operatore, ed **M** l'insieme degli oggetti simili rilevati dal tool di integrazione, i due parametri si possono definire come segue:

- ✓ Il parametro *recall*, viene calcolato sui documenti rilevanti recuperati, e ne misura la percentuale rispetto al totale contenuto nella raccolta.

$$recall = \frac{|H \cap M|}{|H|}$$

- ✓ Il parametro *precision* invece riguarda i documenti restituiti dalla ricerca e rappresenta la percentuale di documenti rilevanti.

$$precision = \frac{|H \cap M|}{|M|}$$

Capitolo 4

INTEGRAZIONE DI DUE GVV : CLASSE COMPARATORE

4.1 Linee guida: inserimento di una nuova sorgente

Nel secondo capitolo, si è parlato di come il processo di integrazione di più sorgenti all'interno della Global Virtual View in MOMIS preveda un successione di passaggi per consentire all'utente di creare il proprio schema delle concettualizzazioni. Il primo passo e' la fase di annotazione delle sorgenti e di creazione del Common Thesaurus, il secondo è la classificazione della conoscenza rappresentata e per concludere la creazione della vista globale unica e la sua annotazione.

Lo schema descritto definisce dei passaggi statici, unidirezionali ma nel momento in cui si presenti la necessità di gestire delle azioni di modifica ad uno schema già creato e salvato, questo schema non e' più applicabile. Esistono due metodologie per risolvere questa trasformazione:

- ✓ ripartire dal procedimento iniziale creando una nuova ontologia,
- ✓ sviluppare tecniche per apportare le modifiche partendo dallo schema finale da modificare.

Il primo approccio è di tipo statico, poiché non prevede una vera e propria modifica dell'ontologia, ma aggira il problema ricreando un nuovo schema. Questo procedimento, oltre a portare ad un maggior impiego di tempo, non conserva le informazione di mapping memorizzate poichè ricrea un cluster ex-novo, risulta,

quindi, essere di poca utilità in vista di uno sviluppo di software industriale. Al contrario, la seconda soluzione preserva le informazioni già raccolte, come mapping e Common Thesaurus, in particolare quelle definite manualmente dall'utente. Questa ultima soluzione è stata adottata nello sviluppo di questa tesi e rientra nel campo della gestione delle dinamiche di una ontologia, in particolare è stato implementato un tool seguendo le tecniche proposte dall'approccio basato sulla evoluzione delle ontologie.

Il passaggio da una versione già strutturata di una ontologia, Ω_1, ad un'altra, Ω_2, può avvenire attraverso varie trasformazioni. Secondo la teoria AGM, è possibile creare dei raggruppamenti di queste operazioni in tre tipologie standard di modifica:

- ✓ Espansione o integrazione di una nuova sorgente: una nuova sorgente deve essere integrata all'interno della GVV, apportando nuove classi locali e nuovi attributi locali da mappare nello schema già formato.
- ✓ Contrazione o eliminazione di informazioni: una sorgente, o parte di questa, deve essere eliminata perché dichiarata obsoleta, cioè non riflette le necessità richieste nelle concettualizzazioni del mondo reale.
- ✓ Revisione o aggiornamento di sorgenti già presenti: una sorgente deve essere aggiornata nella sua struttura di classi e attributi, questo può essere visto come una combinazione delle due azioni precedentemente citate, cioè eliminazione del concetto da modificare ed inserimento del nuovo concetto modificato.

Nel capitolo precedente è stato riportato come queste problematiche siano state trattate in letteratura e come siano state affrontate nei progetti di ricerca, soprattutto nell'ambito dell'Information Retrieval.

Nell'ambito specifico in questa tesi, si è voluto focalizzare l'attenzione sulla prima tipologia di modifica, con l'obiettivo, di sviluppare un algoritmo per l'integrazione di una o più sorgenti sfruttando quello che si può definire una comparazione o confronto di due versioni della stessa ontologia. La trattazione di questo algoritmo viene rimandata al paragrafo successivo.

Nella sezione relativa al merge di ontologie, si è già parlato del fatto che la fusione di più ontologie in una unica globale, nasca a volte dalla necessità di

estendere particolari ontologie iniziali. In questa ottica è stato sviluppato un algoritmo per effettuare la fusione di due ontologie, creando una vista globale unica Ω_{Merge}, partendo da due ontologie indipendenti tra loro, Ω_{Old} e Ω_{New}, anche se rappresentanti entrambe due versioni della stessa vista globale:

- ✓ Ω_{Old}, è l'ontologia iniziale, creata sulle vecchie sorgenti, da integrare con i nuovi elementi di informazione;
- ✓ Ω_{New}, è una ontologia creata ex-novo, che contiene sia le informazioni delle nuove sorgenti che i concetti definiti "globali" relativi alla ontologia iniziale Ω_{Old}, vista come unica sorgente locale.

Nella fase di pre-integrazione di due ontologie di questo tipo il primo problema che si pone è quale sia la "direzione" più opportuna da seguire per la creazione della Ω_{Merge}: integrare Ω_{New} con le informazioni raccolte nella precedente versione, sostituendo, quindi, i concetti globali con le informazioni definite sulle sorgenti locali della vista iniziale, oppure aggiornare quest'ultima con i nuovi concetti raccolti. I due approcci sono ambivalenti, entrambi presentano, infatti, le stesse problematiche riguardante la comparazione delle schemi e, quindi, la ricerca delle relazioni tra concetti globali dell'ontologia iniziale con quelli locali della Ω_{New}. Nonostante ciò esiste una sostanziale differenza da un punto di vista computazionale: le primitive di trasformazioni richieste dal primo approccio risultano essere di maggiore complessità rispetto al secondo. La sostituzione di una informazione globale già presente richiede, infatti, la cancellazione e l'inserimento di diverse informazioni, mentre il secondo approccio risulta essere, da questo punto di vista, meno gravoso poiché implementa esclusivamente un algoritmo di espansione, cioè di inserimento di nuove informazioni. La scelta ricade quindi sulla tipologia delle applicazioni che utilizzano la struttura della ontologia.

Regole base per creare un algoritmo di comparazione

L'algoritmo di comparazione tra due versioni di una stessa ontologia deve tener conto dei seguenti presupposti:

a) Ω_{Merge} non deve ridefinire una classificazione tra le sorgenti appartenenti alla Ω_{Old}, ma deve tutelare e mantenere le informazioni raccolte relative al

Common Thesaurus e alla Mapping Table con i concetti globali (classi e attributi).

b) l'integrazione di una nuova sorgente non deve perdere le informazioni relative alle struttura delle sorgenti locali della Ω_{Old}: classi e attributi locali.

c) l'integrazione della nuova sorgente deve avvenire partendo dalla Ω_{Old}, considerata come vista globale e non come insieme di sorgenti separate l'una dall'altra.

d) valutare i casi particolari che si possono presentare nell'inserimento di una nuova sorgente relativi alla modifica alla struttura dell'ontologia iniziale.

Questi presupposti non sono necessariamente scollegati tra di loro, si può affermare infatti che alcune siano la conseguenza di altre, esempio non si può pensare di rispettare la clausola (a), cioè di mantenere le informazioni raccolte nel Common Thesaurus della Ω_{Old}, senza rispettare la clausola (b).

Ricerca delle inter-relazioni tra le due ontologie

Come già citato, gli approcci che si possono adottare per creare l'allineamento di due ontologie possono essere varie, sia di tipo semantico che sintattico, tuttavia quando si parla di integrazioni di due versioni della stessa ontologia per l'inserimento di una nuova sorgente le tecniche che si possono usare possono diventare più mirate. Si ha come punto di partenza che esistono già delle relazioni intrinseche tra schemi delle due ontologie, che collegano concetti definiti "globali" della Ω_{Old} a concetti definiti "locali" della Ω_{New}, tali relazioni inter-schema esistono per il solo fatto che l'una, Ω_{New} , ingloba una vista globale della prima, Ω_{Old}. I concetti in esame rappresentano, quindi, la stessa concettualizzazione anche se non sempre formalizzata in modo equivalente. Le relazioni tra questi concetti forniscono una base indispensabile per trovare i collegamenti inter-schema tra le due ontologie e quindi, per far migrare le informazioni secondo il verso stabilito. La ricerca di tali legami non è tuttavia banale, specialmente se viene modificato il nome con cui i concetti vengono definiti; questo problema è presente soprattutto nei progetti dove sono implementati algoritmi di cluster che non prevedono l'assegnamento di livelli di

priorità alle sorgenti per la scelta del nome da dare alla classe globale. In questo caso i legami sintattici di similarità su concetti comuni vengono persi, risulta quindi utile seguire un approccio sintattico per effettuare la ricerca delle inter-relazioni. Una possibile alternativa è ricercare la provenienza di tale concetto definendo qual è la sorgente di informazione ad esso associata e, nel caso questa risulti essere la Ω_{Old}, effettuare tutti gli aggiornamenti necessari per mantenere la consistenza dell'ontologia. Viceversa, se l'algoritmo di cluster dà la possibilità di mantenere una relazione di omonimia tra concetti comuni, definiti nelle due ontologie, allora si possono applicare gli algoritmi di ricerca sintattica precedentemente scartati.

Modifica della struttura dello schema ontologico iniziale

La regola base (d) stabilisce un vincolo che conserva lo schema iniziale della Ω_{Old}, cioè la propagazione dei cambiamenti, legati all'inserimento di nuovi concetti, non deve comportare una modifica alla struttura della ontologia iniziale. Ad esempio se si considerano due concetti globali c_glob_1 e c_glob_2 della Ω_{Old}, questi possono essere definiti come il risultato di una classificazione di concetti locali tali per cui :

$$c\_glob_n = \{c\_local_n^1, c\_local_n^2, ..., c\_local_n^m\}$$

L'inserimento di una nuovo concetto potrebbe portare all'eliminazione di uno dei due concetti globali, supponiamo c_glob_1, e alla migrazione dei suoi collegamenti locali verso c_glob_2, come mostrato in figura 4.1. Accettare questo tipo di cambiamento a priori, cancellare quindi c_glob_2 senza valutare quale siano le relazioni della Ω_{New} che hanno portato a questa modifica e, soprattutto, senza considerare le relazioni che esprimono i legami tra i concetti globali e quelli locali interni alla Ω_{Old}, non è sempre corretto.

In queste situazioni le strade che si possono percorrere sono diverse e specifiche caso per caso, possono variare sia in base alla struttura dello schema dell'ontologia che al valore che questo attributo globale occupa all'interno di questo schema.

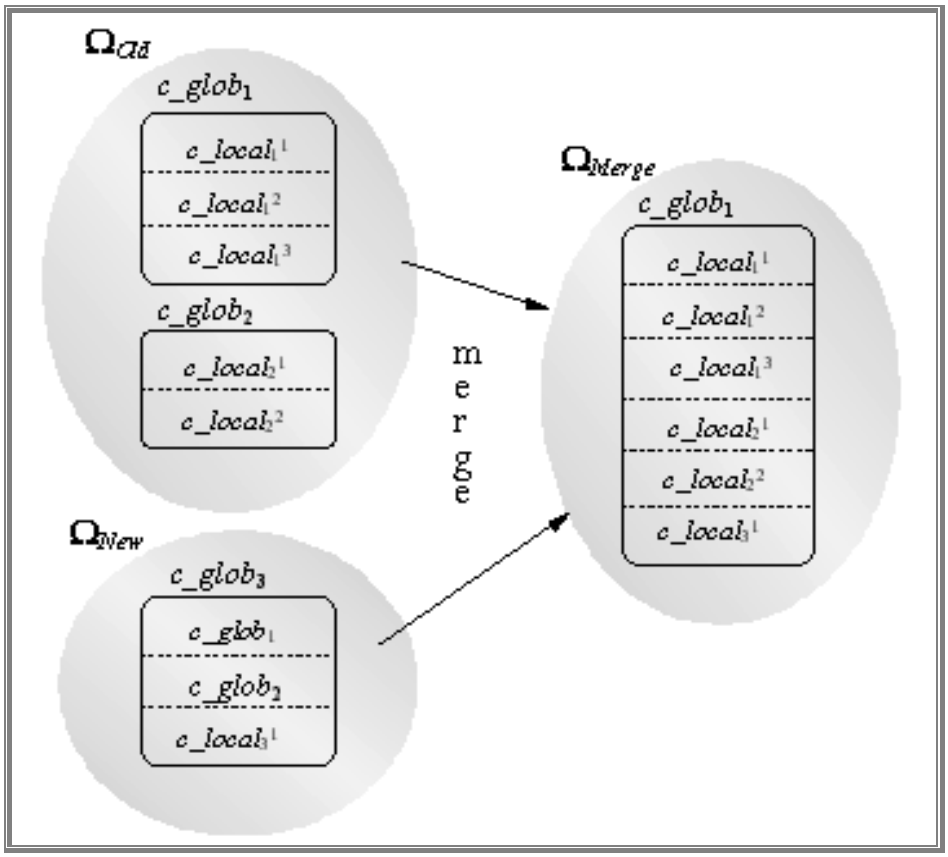

Figura 4.1: Violazione di uno dei presupposti iniziali: modifica dello schema iniziale

Il paragrafo successivo descrive più dettagliatamente il tool implementato nel sistema MOMIS, evidenziando i passaggi che portano alla creazione di una vista globale aggiornata. Per uniformità con la nomenclatura utilizzata, in questa sezione verrà introdotto un nuovo formalismo più specifico per l'applicazione in studio, in particolare, si farà riferimento all'ontologia finale Ω_{Merge} con *GVVadd*, mentre con il termine *GVVold* e *GVVnew* si farà riferimento alle due ontologie al confronto, rispettivamente relative alla Ω_{Old} e alla Ω_{New}.

4.2 Algoritmo di Comparazione

Il corpo dell'algoritmo è contenuto nella classe Java ***Comparatore.class*** nel package ***SI_Designer***, in cui è stata sviluppata l'interfaccia grafica tra il sistema e il progettista per coordinare l'esecuzione dei diversi software che partecipano all'integrazione della GVV, in particolare essa è strutturata in una sequenza predefinita e unidirezionale di pannelli, uno per ogni fase: partendo dal primo il

progettista passa da uno all'altro fino al completamento dell'integrazione delle sorgenti.

4.2.1 Creazione della GVVnew

L'algoritmo prevede come primo passaggio la creazione della *GVVnew*, questa azione viene attivata attraverso il pulsante ***Update GVV***, posto nel primo di questi pannelli, relativo all'estrazione delle sorgenti locali. L'evento ad esso associato, chiama la funzione "***void buttons_newmomis()***", la quale provvede alla creazione di un nuova interfaccia grafica strutturalmente e graficamente simile a quella precedente, alla quale viene passata la *GVVold* trasformata in sorgente locale, come mostrato in figura 4.2.

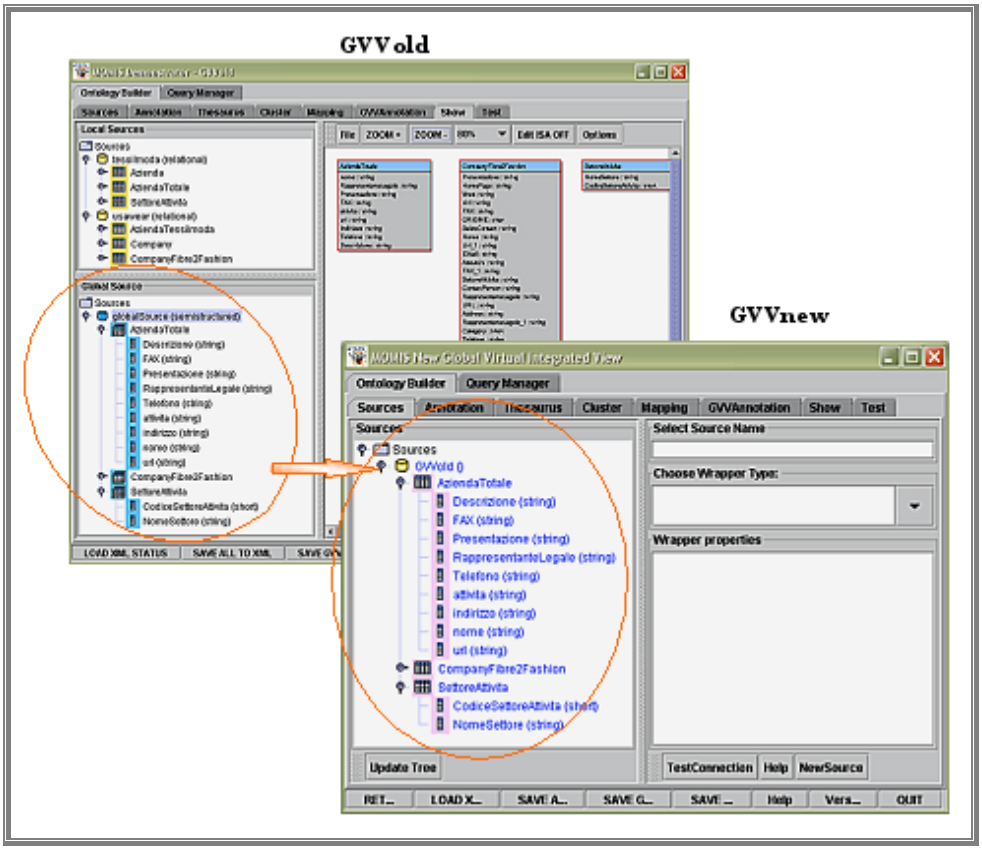

Figura 4.2: Passaggio della *GVVold* come sorgente locale alla *GVVnew*

La presenza di uno schema completo nella prima interfaccia diventa, quindi, elemento essenziale, nel caso in cui questa condizione non sia vera l'evento non può essere attivato. La *GVVold* viene passata come unica sorgente locale, cioè dallo

schema viene chiamata una funzione che trasforma le classi e gli attributi da globali in locali, mantenendo le informazioni sulle annotazioni dello schema globale. Non vengono riportate le informazioni del Common Thesaurus perché non sono necessarie nel processo di integrazione della *GVVnew*. La nuova integrazione tratta la *GVVold,* da modificare, come le nuove sorgenti da inserire, perdendo ogni tipo di informazione riguardate le *lsourceOld*, sorgenti locali della *GVVold*.

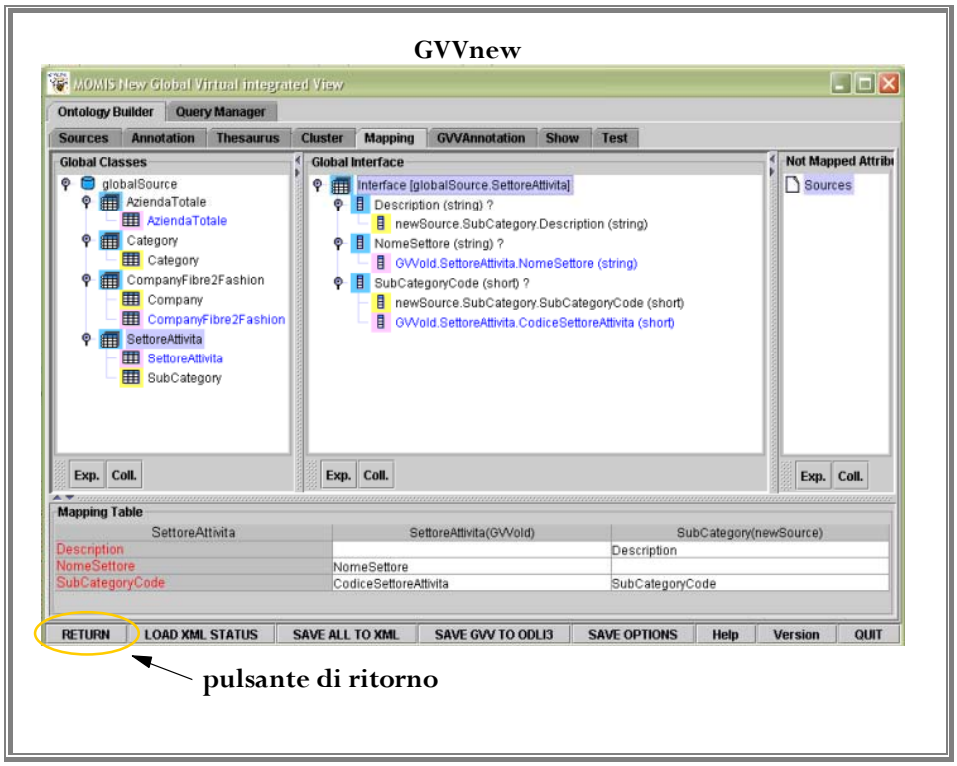

Figura 4.3: *GVVnew*: i diversi colori evidenziano le due tipologie di informazioni

La figura 4.3 mostra un esempio di *GVVnew*, questa viene creata seguendo tutti gli step previsti nella procedura di MOMIS, vengono così integrati insieme i concetti globali della vecchia vista con quelli derivanti dalle nuove sorgenti. Per rendere maggiormente visibile le diverse categorie di informazioni, quelle derivanti dal *GVVold* sono evidenziate in un colore diverso: blu; stessa cosa per le icone, quelle associate alla *GVVold* sono uniformate in un unico colore: rosa. In questo modo l'utente può velocemente capire quale sia l'origine dell'informazione, in particolare può accorgersi della presenza di casi particolari, trattati nei seguenti paragrafi.

Come risultato finale si ottengono, come già anticipato, due viste rappresentati ognuna una versione diversa della realtà.

```
private void buttons_newmomis(){
        Schema schemaGVVold = new Schema();
        String file = "conf/siDesigner.conf";
        try {
            schemaGVVold= _gsProxy.getSchema();
            InputStream conf = new FileInputStream(file);
            Properties configuration = new Properties();
            configuration.load(conf);
            conf.close();
            if (schemaGVVold.getGlobalSource()== null) {
             JOptionPane.showMessageDialog(null, "Old GVV not exist", "ERROR",
                JOptionPane.ERROR_MESSAGE );
             } else {
                schemaGVVold = schemaGVVold.getGlobalSchemaAsLocal();
                //schemaGVVold.getSource("localSource").setName("GVVold");
                _newsd = new SI_Designer (configuration, schemaGVVold,"open");
                _newsd.returnbutton.addActionListener( new ActionListener() {
                    public void actionPerformed(ActionEvent e_est) {
                        UpdateGVVold();
                    }
                });
            }
        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(null, "Error open name file: " + file );
            e.printStackTrace();
        } catch (IOException e) {
            String message = "Error load configurationa ";
            System.out.println(message);
            e.printStackTrace();
        }catch (Exception e) {
            String message = "Errore creazione nuovo pannello GVVnew";
            System.out.println(message);
            e.printStackTrace();
        }
    }
```

```
   /**funzione che sostituisce GVVold con la Gvvnew integrata con i collegamenti
alle classi locali*/
 private void UpdateGVVold (){
    Comparatore compare = new Comparatore(_newsd.getGsProxy(),_gsProxy );
    //update source
    compare.addSource();
    //update tree
    compare.compare();
    //update Thesaurus
    compare.updateThesaurus();
}
```

Funzioni che gestiscono la creazione della *GVVnew*
e la procedura di confronto tra le due ontologie

4.2.2 Comparazione delle due viste

Il processo di integrazione di una nuova sorgente non si conclude con la creazione della *GVVnew,* essa non può essere, infatti, accettata come schema finale, in quanto non rispetta i requisiti iniziali, perde ogni informazione riguardante le *lsourceOld*, il Common Thesaurus ed il Mapping Table con le classi e gli attributi globali della *GVVold*. Una volta che le due viste sono state create, si deve procedere alla loro integrazione in una vista globale unica. La *GVVall* può essere ottenuta in due modi: in avanti verso la *GVVnew*, oppure indietro verso la *GVVold*. Nel primo caso, che prevede una sostituzione delle informazioni contenute nella *GVVnew* relative alla vista precedente con quelle delle *lsourceOld* ad esse associati, l'aggiornamento e integrazione del Common Thesaurus e la Mapping Table, non si ha una modifica della *GVVold* e la *GVVall* viene creata da modifiche sulla *GVVnew*. Nel secondo caso, accade l'opposto, la *GVVold* viene incrementata dalle nuove informazioni raccolte. Nonostante i due approcci siano entrambi esatti, è sembrato più conveniente seguire il secondo. Le motivazioni sono principalmente di ordine computazionale, mentre il primo richiede una sostituzione degli attributi e delle classi e un'aggiunta delle vecchie informazioni relative alle relazioni delle vecchie sorgenti, il secondo prevede solo una allargamento della vecchia vista con le nuove informazioni.

L'algoritmo di comparazione delle due viste viene eseguito dalla classe ***Comparatore***, chiamata dall'utente tramite il pulsante "***RETURN***"della barra delle applicazioni della *GVVnew*, cerchiato in figura 4.2.. L'evento ad esso associato chiama la funzione "***void UpdateGVVold()***", che genera una instanza di questa classe, chiamata ***compare***. Il costruttore della classe comparatore richiede come parametri di ingresso gli schemi delle due viste globali ed ha la seguente stuttura:

```
public Comparatore (GlobalSchemaProxy newsc, GlobalSchemaProxy oldsc){
    _GspGVVnew=newsc;
    _GspGVVold=oldsc;
    _schemaGVVnew= newsc.getSchema();
    _schemaGVVold= oldsc.getSchema();
    estractGVVold(); }

//Variabili globali

/**Schema globale della GVVnew e GVVold*/
private GlobalSchemaProxy _GspGVVnew;
private GlobalSchemaProxy _GspGVVold;

/**Schema GVVnew e GVVold*/
private Schema _schemaGVVnew = new Schema();
private Schema _schemaGVVold = new Schema();

/**sorgente globale della GVVold*/
private GlobalSource _globalsourceOGVV = new GlobalSource();

/**Array che contiene le Classi Globali della GVVold*/
private GlobalInterface[] _GlobalClassGVVold;

/**numero classi globali GVVold*/
private int dim;

/**Array che contiene le Classi Globali della GVVnew*/
private GlobalInterface[] _GlobalClassGVVnew;
```

Costruttore della classe Comparatore

Questa classe contiene metodi per eseguire il confronto tra due viste globali, per fare ciò ha bisogno di estrarre tutta la struttura sia dallo schema della vecchia che dalla nuova vista. Il costruttore inizia questo processo di analisi, chiamando la

funzione "***void extractGVVold()***", la quale memorizza all'interno di un array le classi globali della *GVVold*

```
public void extractGVVold(){
    _globalsourceOGVV = _schemaGVVold.getGlobalSource();
    try { _GlobalClassGVVold= globalsourceOGVV.getGlobalInterface();
    } catch (OdlException e) {
        JOptionPane.showMessageDialog(null, "Global class not find." );
        e.printStackTrace();
    }
    dim = _GlobalClassGVVold.length;
    message = "NUMERO CLASSI GLOBALI GVVold:["+ dim +"]\n";
    System.out.println(message);
    for (int i=0; i < dim; i++){
        message = "LA GlobalInterface INDIVIDUATA E':[" +
        _GlobalClassGVVold[i].getDottedName() +"]";
        System.out.println(message);
    }
}
```

Funzione che gestisce l'estrazione delle classi globali della GVV iniziale

Come primo passo di aggiornamento della *GVVold*, il sistema inserisce dalla *GVVnew* i riferimenti alle nuove sorgenti, questo procedimento viene eseguito dalla funzione "***void addSource ()***".

```
public void addSource (){
    Object [] v1 = new Source [0];
    v1 = _schemaGVVnew.getSourcesOrderByName();
    Source[] source = new Source[v1.length];
    for (int i=0; i<v1.length; i++){
        source[i]=(Source)v1[i];
        String namesource = source[i].getName();
        boolean change = false;
        if (namesource.equals("GVVold")==false){
            //controllo sul nome della sorgente
            while(_schemaGVVold.getSource(namesource)!= null){
                namesource= JOptionPane.showInputDialog
                    (null,"The name [" + namesource +
```

```
                    "] already exists.\nPlease enter a new name for the source:",
                    "CHANGE NAME OF SOURCE",
                    JOptionPane.PLAIN_MESSAGE );
                while(namesource.equals("")){
                 namesource= JOptionPane.showInputDialog(null,
                    "Please enter a new name for the source:",
                   "CHANGE NAME OF SOURCE",
                   JOptionPane.PLAIN_MESSAGE);
                }
                 change = true;
            }
            if (change){ source[i].setName(namesource); }
            try { _schemaGVVold.addSource(source[i]);
            } catch (OdlException e) {e.printStackTrace(); }
        }
    }
}
```

Funzione che gestisce l'aggiornamento delle nuove sorgenti dati nella *GVVadd*

Questa funzione esegue un controllo sul nome della sorgente da inserire, ad esempio se esiste già una sorgente omonima nella *GVVold*, allora l'utente dovrà modificare tramite un pannello il nome della nuova sorgente, non è previsto che ad una sorgente non venga assegnato alcun nome, anche questo caso viene segnalato dal sistema come errore.

4.2.3 Tre casi di gestione delle classi globali

Una volta inserite le nuove sorgenti nella *GVVnew*, viene invocata la funzione "***void compare()***", con il compito di gestire i differenti casi che si possono presentare a causa dell'inserimento di una nuova sorgente in una vista già creata [32].

```
public void compare(){
    int corrispondenze;
    boolean aggiornaAttributi;
    boolean findGVGVVold;
    int dimGCGVVnew;
    GlobalInterface GlClassO = new GlobalInterface();
```

```
GlobalSource globalsourceNGVV = _schemaGVVnew.getGlobalSource();
try {_GlobalClassGVVnew = globalsourceNGVV.getGlobalInterface();
} catch (OdlException e) {
    JOptionPane.showMessageDialog(null, "Global class not find." );
    e.printStackTrace();
}
dimGCGVVnew = _GlobalClassGVVnew.length;
//ciclo sulle classi globali
for (int i=0; i < dimGCGVVnew; i++){
    message ="Classe Globale: ["+_GlobalClassGVVnew[i].getDottedName()+"]";
    System.out.println(message);
 //estrapolo le classi locali della GVVnew in ordine alfabetico
Interface[] localclass;
localclass = _GlobalClassGVVnew[i].getAllLocalInterfacesOrderByName();
message = "NUMERO Classi locali:["+ localclass.length +"]";
System.out.println(message);
//Setto variabili inizio ciclo
GlClassO = null;
aggiornaAttributi = false;
findGVGVVold=false;
//controllo se ci sono più classi globali
corrispondenze=controlloappartenenzaLocalClass(localclass);
if(corrispondenze == 1){
    for (int y=0; y < localclass.length; y++){
        message = "Classe Locale: [" + localclass[y].getDottedName() +"]";
        System.out.println(message);
        if(!findGVGVVold){
            if(localclass[y].getSource().getName().equals("GVVold")){
                GlClassO = findLocalClass (localclass[y].getName());
                indGVGVVold=true;
                /inserisco le classi locali precedenti delle nuove sorgenti
                for(int z=0; z<y; z++){
                    GlClassO.addLocalInterface(localclass[z]);
                //setto la variebile booleana per l'aggionramento degli attributi
                    aggiornaAttributi=true;
                }
            }
        }else{ //inserisco le classi locali successive delle nuove sorgenti
            GlClassO.addLocalInterface(localclass[y]);
```

```
            aggiornaAttributi=true;
            }
        }
    }else{
        if(Corrispondenze == 0){
            try {
                _globalsourceOGVV.addInterface(_GlobalClassGVVnew[i]);
            } catch (OdlException e1) {
                int suffisso=1;
                String GCname=_GlobalClassGVVnew[i].getName();
                String namesuff=GCname;
                while((controlloNameGlobalClass(namesuff))==true){
                    namesuff=GCname+"["+suffisso+"]";
                    suffisso++;
                }
                _GlobalClassGVVnew[i].setName(namesuff);
                //ritento l'inserimento
                try {
                _globalsourceOGVV.addInterface(_GlobalClassGVVnew[i]);
                } catch (OdlException e2) {
                    // TODO Auto-generated catch block
                    e2.printStackTrace();
                }
                JOptionPane.showMessageDialog( null, +
                    "\n Exception found on the name of Global Class:\n [" +
                    GCname +"].\nIt's been changed in [" +
                    _GlobalClassGVVnew[i].getDottedName()+"]",
                    "INFORMATION: Exception on the name",
                    JOptionPane.INFORMATION_MESSAGE );
                e1.printStackTrace();
            }
        }else{// corrispondenze > 1
            JOptionPane.showMessageDialog(null, "The global class [" +
                _GlobalClassGVVnew[i].getName() +"] collects " +
                corrispondenze +" global class of GVVold,\n" +
                "that must be maintained separated", "INFORMATION",
                JOptionPane.INFORMATION_MESSAGE);
        }
    }
```

```
    if(aggiornaAttributi){
        updateLocalAttr(_GlobalClassGVVnew[i],GlClassO);
    }
   }
}

/** controlla se nellaGVVold esiste una classe globale uno specifico nome*/
private GlobalInterface findLocalClass(String name){
    for (int z=0; z<dim; z++){
        String namegvvold= _GlobalClassGVVold[z].getName();
        if(name.equals(namegvvold)){
            message = "[" + name + "]: è una classe globale della GVVold";
            System.out.println(message);
            return _GlobalClassGVVold[z];
            }
    }
    return null;
}
```

Funzione che gestisce l'aggiornamento delle classi globali della *GVVadd*

L'analisi delle procedure messe in atto dal sistema hanno fatto emergere tre casi, è tuttavia possibile che se ne verifichino di differenti. Per descrivere quali sono i casi che si possono incontrare bisogna prima di tutto definire un glossario delle abbreviazioni usate:

- ✓ *gcOld* sono classi locali nella *GVVnew*, relative a classi globali della vecchia sorgente, composte da un nome *gcOldname*, e da un set di attributi (locali nella *GVVnew* - globali nella *GVVold*) definiti *gcOldAtt*.
- ✓ *lcOld s*ono le classi locali della vecchia sorgente già presenti nella *GVVold*, composta da un nome *lcOldname*, e da un set di attributi locali definiti *gcOldAtt*, tale per cui:

$$gcOld = \{lcOld_{11},...lcOld_{1z}, lcOld_{p1},...,lcOld_{pq}\}$$

- ✓ *lcNew* è una classe locale della nuova sorgente da integrare nello schema vecchio, composta da un nome *lcNewname*, e da un set di attributi locali definiti *lcNewAtt*
- ✓ *gcNew* è la classe globale della *GVVnew*, composta da un nome *gcNewname*, e da un set di attributi globali definiti *gcNewAtt*, tale per cui

$$gcNew = \{gcOld_1,...,gcOld_p, lcNew_1,...,lcNew_n\}$$

Il sistema definirà il Common Thesaurus sulle relazioni derivate dallo schema e le relazioni lessicali intraschema derivate dall'annotazione della nuova sorgente e della *GVVold*, da queste la fase di clustering genererà il mapping tra classi locali in classi globali. Come conseguenza una nuova classe globale *gcNew* sarà generalmente definita sulle vecchie classi globali della *GVVold* e sulle nuove classi locali della nuova sorgente. In questo modo una nuova classe globale è una classe del tipo

$$gcNew = \{lcOld_{11},...lcOld_{1z}, lcOld_{p1},...,lcOld_{pq}, lcNew_1,...,lcNew_n\}$$

Considerata questa composizione, si possono distinguere tre casi:

[1] Una nuova classe globale si compone di una classe locale relativa ad una classe globale della vecchia GVV e di una o più classi locali della nuova sorgente;

[2] Una nuova classe globale si compone unicamente di classi locali della nuova sorgente;

[3] Una nuova classe globale si compone di più classi locali relative a classi globali della vecchia GVV e di una o più classi locali della nuova sorgente.

Nello stesso processo di integrazione si possono presentare contemporaneamente anche tutte le tre diverse situazioni su classi globali diverse. L'algoritmo si propone di individuare e gestire queste tre situazioni, e come primo passo, analogamente a quanto visto con la *GVVold*, estrapola dalla *GVVnew* tutti le classi globali salvandole all'interno di un array.

Per capire in quale casistica il sistema si deve porre, per ogni *gcNew* viene calcolato il numero di corrispondenze, cioè quante delle sue classi locali sono classi globali della vecchia vista, *gcOld*. Il controllo di corrispondenza si può basare su due caratteristiche della classe locale: il nome o la sorgente da cui proviene. Inizialmente l'algoritmo eseguiva un controllo sul nome, questo approccio però dava adito ad errori, poiché non sempre c'è uguaglianza tra *gcNewName* e *gcOldname*, proprio per il fatto che il cluster della *GVVnew* viene ricreato da zero. Per questo motivo è stato modificato l'approccio di controllo, eseguendolo sul nome della sorgente che contiene la classe locale esaminata. Il numero di corrispondenze viene memorizzato nella variabile *corrispondenze* e viene calcolato dalla funzione "***int controlloappartenenzaLocalClass (Interface[] locInt)***", la quale richiede come parametri di ingresso la classe globale su cui deve essere attuato il controllo.

```
private int controlloappartenenzaLocalClass(Interface[] locInt){
    int cont=0;
    for(int i=0; i< locInt.length; i++){
        if(locInt[i].getSource().getName().equals("GVVold"))
            cont++;
    }
    return cont;
}
```

Funzione che definisce il numero di corrispondenze
tra le classi della *GVVold* e la *GVVnew*

Caso 1: Una nuova classe globale contiene una corrispondenza a una classe globale della *GVVold*

Questo caso si ha quando la variabile corrispondenza ha valore uno, e cura le situazioni di questo tipo:

$$gcNew = \{gcOld, lcNew_1, ..., lcNew_n\}$$

in cui una nuova classe globale *gcNew* è formata da un insieme di classi locali appartenenti alla *GVVold* e un insieme appartenente alla nuova sorgente.

Se un'applicazione basa il suo funzionamento sulla struttura della GVV, questo scenario non costituisce problemi, poichè non si ha una variazione dello schema della vecchia vista globale ma solo un suo incremento di informazioni. L'algoritmo ricerca nella lista delle classi globali della *GVVold* quella che a cui si riferische la *gcOld* della *gcNew,* e incrementa le sue classi locali con quelle della nuova sorgente mappate nella *gcNew* in studio. Nello specifico: esegue una scansione su tutte le classi globali della *GVVold*, memorizzate nell'algoritmo precedentemente definito, fino a che non trova la corrispondenza; a questo punto memorizza la corrispondenza nella variabile *GlClassO* e la estende con le classi locali della nuova sorgente. Le *lcNew* sono sia le classi locali esaminate precedentemente, che non hanno dato rapporto falso al controllo di corrispondenza, che quelle successive a quella in esame. Su queste ultime non viene svolto il controllo, perché ritenuto ridondante. Il controllo, viene sempre stabilito sulla base del nome della sorgente e non il nome della classe. Quando non c'è corrispondenza tra quest'ultimi, poiché l'algoritmo prevede di estendere le informazioni verso la vecchia vista, viene mantenuto come nome quello di gcOld, cioè *gcOldname*.

Stessa cosa vale per gli attributi *gcNewAtt*, questi sono costituiti da un insieme di attributi in parte comuni alle vecchie classi globali e in parte generati dall'apporto delle nuove classi locali. L'aggiornamento viene fatto sugli attributi globali della classe globale nella vecchia GVV, *gcOld,* inoltre, non viene eseguito ad ogni inserimento, ma viene svolto una sola volta alla fine del ciclo sulla *gcNew*. Questa trattazione viene rimandata al paragrafo successivo dove verrà esaminata la funzione che si occupa del loro aggiornamento: ***updateLocalAttr***.

Caso 2: Una nuova classe globale si compone unicamente di classi locali della nuova sorgente

Si tratta del caso più semplice, in cui la variabile *corrispondenza* ha valore zero. Viene creata, nella *GVVnew,* una nuova classe globale *gcNew* con nome *gcNewName*, che mappa una o più nuove classi locali; non vi sono quindi relazioni che uniscono classi locali della nuova sorgente con quelle della vecchia vista, *GVVold*. In questo caso *gcNew* ha la seguente struttura:

$$gcNew = \{lcNew_1, ..., lcNew_n\}$$

L'algoritmo compirà un inserimento totale della *gcNew* nella vecchia vista. Inserire una classe globale significa inserire tutte le corrispondenze con le classi locali, gli attributi globali e il mapping con quelli locali. A differenza del primo caso, questo tipo di inserimento non richiede alcun tipo di aggiornamento degli attributi. L'unico tipo di controllo viene fatto sul nome della classe, questo infatti dovrà essere unico nella lista di quelle presenti nella *GVVold*. La funzione chiamata per questo controllo, riporta qui di seguito, è "***boolean controlloNameGlobalClass (String name)***", essa prende come parametro di ingresso *gcNewName*, e restituisce un booleano a seconda che la funzione ***findGlobalClass*** trovi o meno nella lista delle *gcOld* una classe locale con lo stesso nome.

```
private boolean controlloNameGlobalClass (String name){
    GlobalInterface GlIntfind = findGlobalClass(name,_GlobalClassGVVold);
    if(GlIntfind != null)
        return true;
    else
        return false;
}

private GlobalInterface findGlobalClass (String nameGC, GlobalInterface[] GlCls){
    for (int z=0; z<dim; z++){
        String s = GlCls[z].getName();
        if(s.equals(nameGC)){
            return GlCls[z] ;
        }
    }
    return null;
}
```

Funzioni per la gestione delle corrispondenze tra classi globali della vista iniziale e classi locali di quella modificata.

Caso 3: Una nuova classe globale contiene più corrispondenze a classi globali della *GVVold*

Si tratta della situazione più critica in quanto la struttura della GVV è modificata pesantemente attraverso il processo di integrazione. La distinzione tra le classi viene modificata poiché più vecchie classi globali vengono agglomerate in una nuova classe globale.

$$gcNew = \{gcOld_1, ..., gcOld_p, lcNew_1, ..., lcNew_n\}$$

Se un'applicazione appoggia il suo funzionamento sulla struttura della GVV, questo scenario potrebbe costituire dei problemi. Questa situazione si può riscontrare se una *lcNew* ha forti legami con più classi globali definite nello schema della *GVVold*. Questo caso ha richiesto una maggiore attenzione, in quanto le decisioni che si possono prendere variano a seconda dei casi, e in base al livello di libertà da dare all'utente.

Un primo approccio è quello di unire le classi come suggerito dal nuovo schema, questa soluzione è stata subito scartata, perché questo è esito di un cluster eseguito su un Common Thesaurus che non tiene conto delle relazioni tra *lsourceOld* e, quindi, delle *lcOld*, ma che mantiene solo delle relazioni limitate ai concetti globali della vecchia vista. Inoltre esso genera una modifica della struttura precedentemente formata, creando possibili disagi alle applicazioni.

Un altro approccio è quello di mantenere divise le *gcOld* e creare una nuova classe globale nella *GVVold* contenente le *lcNew*. Inizialmente l'algoritmo prevedeva questo tipo di soluzione, successivamente l'analisi e la sperimentazione hanno portato alla considerazione che questa soluzione può portare ad errori di errata concettualizzazione della realtà nello schema. Se si prende il caso mostrato in figura 4.4, dove una classe locale della nuova vista è una generalizzazione di *gcOld*, creare, a priori, tre classi separate l'una dall'altra è concettualmente sbagliato.

A differenza di quanto potrebbe accadere in una relazione di similitudine, in questo caso le classi non sono scollegate tra loro ma sono unite da una relazione, quella di generalizzazione, che è indipendente da ciò che può restituire un'analisi delle relazioni con le *lsourceOld*.

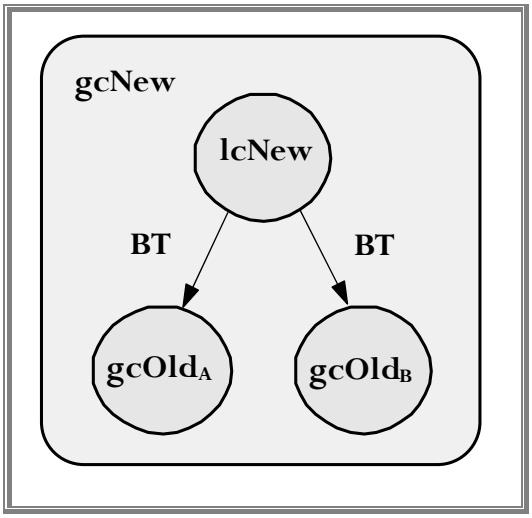

Figura 4.4: lcNew è una generalizzazione di due gcOld

Nell'algoritmo si è scelto, quindi, di adottare una ulteriore soluzione dando massima libertà all'utente. Le classi globali, a cui fanno riferimento le *gcOld*, non vengono modificate in modo tale da non alterare lo schema della *GVVold* senza tener conto delle relazioni generate dalle le sue sorgenti locali, mentre le *lcNew* non vengono mappate nel nuovo schema, ma vengono lasciate nelle mani del progettista, il quale deciderà in prima persona come comportarsi, in base alla sua esperienza. Questa situazione anomala viene, comunque, notificata al progettista con un pannello informativo, come mostato in figura 4.5.

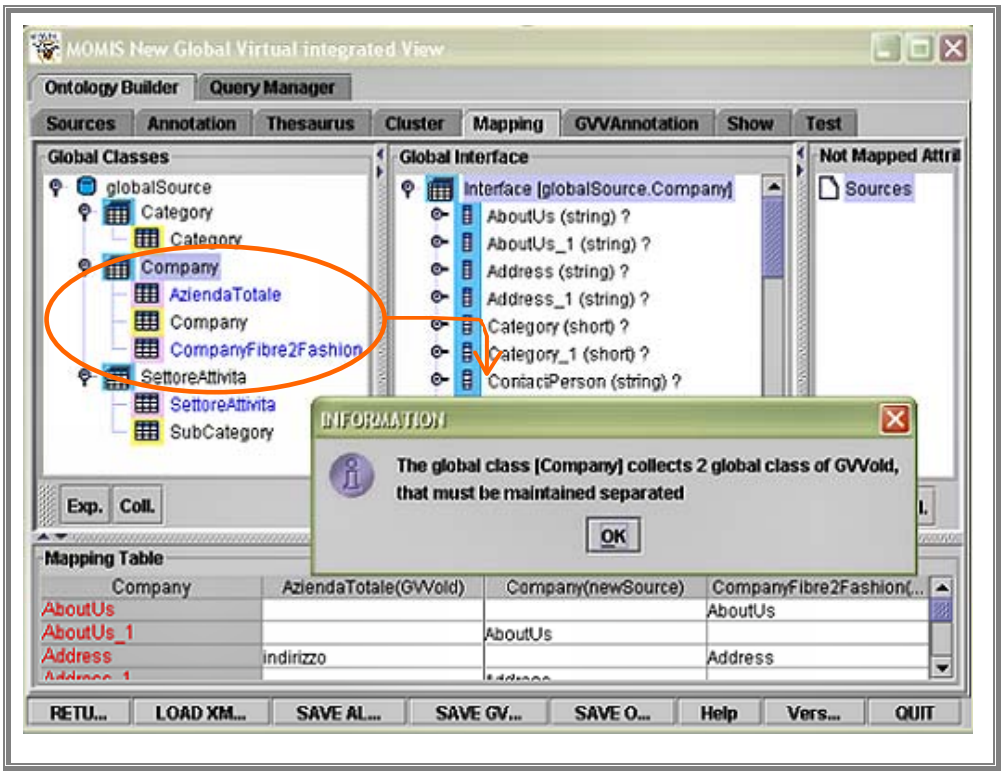

Figura 4.5: Pannello informativo di più gcOld unite in un unica gcNew

In questo caso la classe globale della vecchia vista, GVVold.AziendaTotale, è stata annotata con lo stesso significato di un'altra classe, appartenente sempre alla *GVVold*, GVVold.CompanyFibre2Fashion. Il Common Thesaurus risulta, quindi, ampliato con la seguente relazione di similitudine: "GVVold.AziendaTotale SYN GVVold.CompanyFibre2Fashion"; che crea l'unione delle due classi. In questo caso la *lcNew,* NewSource.Company, non viene mappata.

4.2.4 Aggiornamento degli attributi

Abbiamo visto che nel caso in cui una nuova classe globale è composta da una classe locale relativa alla sorgente della vecchia GVV e da una o più classi locali appartenenti alla nuova sorgente, l'algoritmo prevede l'aggiornamento degli attributi. Come per le classi globali della *GVVnew*, anche i *gcNewAtt*, sono costituiti da un insieme di attributi locali in parte comuni alle vecchie classi globali, e in parte generati dall'apporto delle nuove classi locali. La tabella 4.1 illustra la situazione più generale del mapping degli attributi nella *GVVold*.

<table>
<tr><th></th><th>$lcOld_1$</th><th>...</th><th>$lcOld_k$</th><th>$lcNew_1$</th><th>...</th><th>$lcNew_n$</th></tr>
<tr><td>$gcOldAtt_1$</td><td colspan="3" rowspan="3">mapping di gcOld</td><td colspan="3" rowspan="6">nuovo mapping</td></tr>
<tr><td>...</td></tr>
<tr><td>$GcOldAtt_m$</td></tr>
<tr><td>$gcNewAtt_1$</td><td colspan="3" rowspan="3">nessun mapping</td></tr>
<tr><td>...</td></tr>
<tr><td>$GcOldAtt_p$</td></tr>
</table>

Tabella 4.1: Mappatura attributi nel caso di in cui una gcNew
si compone di una vecchia classe globale e di una o più classi locali

Il gruppo di vecchi attributi globali, *gcOldAtt*, viene mantenuto nella nuova classe globale, mentre il loro mapping viene aggiornato tenendo conto della corrispondenze di questi con quelli appartenenti alle classi locali *lcNew*. Il gruppo di nuovi attributi globali, ***gcNewAtt,*** non presenta una mappatura in corrispondenza dei vecchi attributi locali, in quanto la loro introduzione è dovuta unicamente all'apporto degli attributi delle nuove sorgenti locali.

L'aggiornamento degli attributi viene gestito all'interno della funzione "***void updateLocalAttr (GlobalInterface GlClassN, GlobalInterface GlClassO)***". I parametri di ingresso richiesti, per eseguire il confronto, sono le due classi ***GlClassN*** e ***GlClassO***, rispettivamente corrispondenti alle classi globali gcNew e gcOld.

```
private void updateLocalAttr (GlobalInterface GlClassN, GlobalInterface GlClassO)
{
MappingElement MapEllO;
Vector VctLocAtt = new Vector();
int puntLA;
//variabile che mi segnala se devo creare un nuovo attributo globale
boolean create ;
GlobalIntBody[] GlIntBodyO GlClassO.getGlobalIntBodiesOrderByName();
//ciclo su GlobalIntBody della GVVold
for (int g=0; g < GlIntBodyO.length; g++){
    GlobalSimpleAttribute[] GlAttO;
    GlAttO = GlIntBodyO[g].getGlobalAttributes();
    //ciclo su GlobalIntBody della GVVnew
```

```
GlobalIntBody[] GlIntBodyN = lClassN.getGlobalIntBodiesOrderByName();
for (int t=0; t < GlIntBodyN.length; t++){
GlobalSimpleAttribute[] GlAttN = GlIntBodyN[t].getGlobalAttributes();
//ciclo sugli attributo globale della GVVnew
for (int i=0; i < GlAttN.length; i++){
    MappingElement MapEllN = GlAttN[i].getMappingElement();
    Attribute[] locAttN = new Attribute[0];
    try { locAttN = MapEllN.getLocalAttributesSortByDottedName();
    } catch (Exception e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    //azzero i parametri iniziali
    MapEllO = null;
    VctLocAtt.clear();
    puntLA = 0;
    create = false;
    //ciclo sugli attributo locali della GVVnew
    for (int s=0; s< locAttN.length; s++){
    if(MapEllO == null){
        if(locAttN[s].getInterface().getSource().getName().equals("GVVold")){
                //trovo la classe globale della GVVold corrispondente
                    for (int y=0; y < GlAttO.length; y++){
                        if(GlAttO[y].getName().equals(locAttN[s].getName())){
                            MapEllO = GlAttO[y].getMappingElement();
                            for(int x=0; x<s; x++){
                                VctLocAtt.add(locAttN[x]);
                                puntLA++;
                            }
                        }
                    }
                }
             }else{ //MapEllO != null
            if(locAttN[s].getInterface().getSource().getName().equals("GVVold")){
                JOptionPane.showMessageDialog(null,  "The global class [" +
                    GlClassO.getName() +"] collect more global attributes.\n"+
                    "global attribute:"+ locAttN[s].getDottedName() +
                    "\nglobal attribute:" +
                    MapEllO.getGlobalAttribute().getDottedName(), "ERRORE
```

```
                FASE DI AGGIORNAMENTO DEGLI ATTRIBUTI",
                JOptionPane.INFORMATION_MESSAGE);
            //devo creare un nuovo attributo globale
            create = true;
            }else{
                VctLocAtt.add(locAttN[s]);
                puntLA++;
            }
        }
    }//conclusione ciclo sugli attribute locali
    if(MapEllO == null){
        int suffisso=1;
        String GAname=GlAttN[i].getName();
        String namesuff=GAname;
        while((controlloNameGlobalAttribute(namesuff,GlAttO))==true){
                namesuff=GAname+"["+suffisso+"]";
                suffisso++;
        }
        GlAttN[i].setName(namesuff);
        if (namesuff.equals(GAname)==false) {
            JOptionPane.showMessageDialog(null, +
                "\n Exception found on the name of Global Attribute:\n[" +
                        GlClassN.getName() + "." + Ganame +
                        "].\nIt's been changed in [" + namesuff+ "]",
                        "INFORMATION: Exception on the name",
                        JOptionPane.INFORMATION_MESSAGE );
                }
                try { GlIntBodyO[g].addAttribute(GlAttN[i]);
                } catch (OdlException e) {
                    JOptionPane.showMessageDialog(null, +
                        "\nAdding not executed on the global attribute:\n" +
                        GlAttN[i].getDottedName(), "ERROR",
                        JOptionPane.ERROR_MESSAGE );
                    e.printStackTrace();
                }
            }else{  //MapEllO != null
            if (puntLA>0){
                if(create){ //creazione di un unovo attributo globale
                    GlobalSimpleAttribute newGlAtt;
```

```
                    NewGlAtt= new GlobalSimpleAttribute();
                    int suffisso=1;
                    String GAname;
                    GAname=((Attribute)VctLocAtt.elementAt(0)).getName();
                    String namesuff=GAname;
                while((controlloNameGlobalAttribute(namesuff,GlAttO))==true){
                    namesuff=GAname+"["+suffisso+"]";
                    suffisso++;
                }
                newGlAtt.setName(namesuff);
                try {
                    GlIntBodyO[g].addAttribute(newGlAtt);
                    newGlAtt.setTypeFromMapping();
                    }   catch (OdlException e) {
                        JOptionPane.showMessageDialog(null,
                            "\nAdding not executed on the global attribute:\n" +
                            GlAttN[i].getDottedName(), "ERROR",
                            JOptionPane.ERROR_MESSAGE );
                            e.printStackTrace();
                    }
                    MapEllO = new MappingElement(newGlAtt);
                    newGlAtt.setMappingElement(MapEllO);
                }
            //ciclo per inserire gli attributi
                for(int z=0; z<puntLA; z++){
            try { MapEllO.addLocalAttribute((Attribute)VctLocAtt.elementAt(z));
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(null,
                                "\nAdding not executed on the local attribute:\n" +
                                ((Attribute)VctLocAtt.elementAt(z)).getDottedName(),
                                "ERROR", JOptionPane.ERROR_MESSAGE);
                                e.printStackTrace();
   }}}}}} } }
/**funzione controlla se nella GVVold esiste una attributo globale col nome passato
*come parametro */
private boolean controlloNameGlobalAttribute ( String name,

            GlobalAttribute[] GlAttO){
   for (int y=0; y < GlAttO.length; y++){
```

```
        if(GlAttO[y].getName().equals(name))
            return true;
        }
    return false;
}
```

Funzioni che gestiscono l'aggiornamento degli attributi della *GVVadd*

La funzione estrae tutta la struttura di ogni attributo globale lavorando su degli array doppi, uno per la vecchia vista globale e uno per la nuova vista globale e aggiorna la ***GlClassO*** quando sorgono informazioni derivanti dalla nuova sorgenti. Prima di spiegare l'algoritmo è bene, quindi, dare prima una descrizione di come sono strutturate le classi relative a questa parte dello schema, figura 4.6.

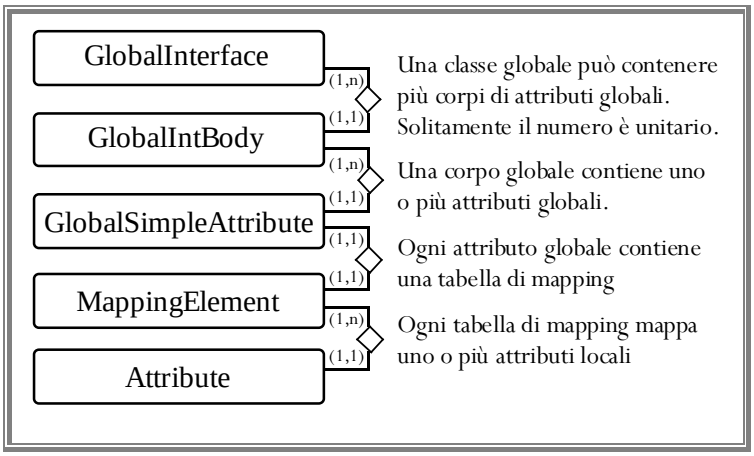

Figura 4.6: Struttura classi tra classe globale e attributo locale

In generale un attributo globale lo si può definire sempre come un insieme di attributi locali, come nella seguente espressione:

$$gAttNew = \{lAtt_1, ..., lAtt_m\}$$

Tuttativa, analogamente a quanto succede con le classi locali, gli attributi locali possono dividersi in due categorie: quelli relativi alle nuove sorgenti inserite nella *GVVnew*, e quelli per cui esiste una corrispondenza con attributi globali della *GVVold*. Su tale ipotesi si possono definire i seguenti casi:

[4] Un attributo globale della *gcNew* si compone unicamente di attributi locali della lcNew;

$$gNewAtt = \{lNewAtt_1, ..., lNewAtt_n\}$$

[5] Un attributo globale della *gcNew* si compone di un attributi locali relativo ad un attributo globale della *gcOld* e uno o più attributi locali della *lcNew*;

$$gNewAtt = \{gOldAtt, lNewAtt_1, ..., lNewAtt_n\}$$

[6] Un attributo globale della *gcNew* si compone di più attributi locali relativi ad attributi globale della *gcOld* e uno o più attributi locali della *lcNew*;

$$gNewAtt = \{gOldAtt_1, ..., gOldAtt_p, lNewAtt_1, ..., lNewAtt_n\}$$

A differenza della funzione ***Compare*** questa non esegue un controllo a priori per capire in quale comportamento adottare. Le scelte del sistema vengono prese in modo sequenziale su controlli di esistenza, qui di seguito descritte.

Dalla classe globale della *GVVold* vengono estratti tutti gli attributi globali, e vengono memorizzati in un apposito array. Gli attributi locali dei *gcOldAtt*, non vengono mai considerati perché su di essi non vengono eseguite nessuna modifica. Anche per la *GVVnew*, vengono estratti tutti gli attributi globali, *gNewAtt*, in questo caso però vengono salvati in un array anche tutti gli attributi locali.

L'algoritmo si muove in cicli innestati, partendo dagli attributi globali fino a quelli locali. Il ciclo su questi ultimi controlla prima di tutto se esiste una corrispondenza con un attributo globale della *GVVold*, questa viene memorizzata nella variabile ***MapEllO***, che viene settato null ad ogni ciclo su un nuovo attributo globale della *GVVnew*. Sempre come per le classi, non sempre dal nome del *gNewAtt* si riesce a risalire alla relazione con *gOldAtt* perché può cambiare, il mapping, infatti, viene creato ex-novo, quindi l'attributo globale può assumere il nome di un attributo locale della nuova sorgente, anche se contiene una o più corrispondenze. Anche in questo caso quindi è risultato più opportuno cambiare il controllo di corrispondenza eseguendolo sulla sorgente dell'attributo.

Se iniziando il controllo su un attributo locale, risulta che precedentemente non è stata trovata nessuna corrispondenza, e questo risulta essere *gOldAtt*, allora viene settata la variabile ***MapEllO*** con l'attributo globale trovato e vengono inseriti

all'interno di un vettore, chiamato ***VctLocAtt***, tutti gli attributi locali precedetemene esaminati risultati *lNewAtt*. A differenza delle classi, gli attributi locali non possono essere inseriti nella *GVVold* fino a che non è finito il ciclo. Questo perché non si ha un controllo a priori sul numero di corrispondenze. Nel momento in cui almeno una corrispondenza è stata trovata e quindi ***MapEllO*** è diverso da null, il ciclo continua anche sugli attributi locali successivi. In questo caso però se questi risultano essere tipo *lAttNew*, vengono inseriti all'interno dello stesso vettore , mentre se viene trovata una ulteriore corrispondenza la variabile ***create*** viene settata a vero e l'algoritmo individua una situazione definita nel terso caso, dove un attributo globale della *gcNew* si compone di più attributi locali relativi ad attributi globale della *gcOld* e uno o più attributi locali della *lcNew*.

Concluso il ciclo sugli attributi locali, sulla base delle informazioni raccolte, la funzione si adopera per gestire le casistica che si è definita. Se la *gcNew* è del primo tipo, cioè raccoglie solo informazioni sulla nuova sorgente, il sistema copia l'attributo globale nella *gcOld*, eseguendo un controllo sulla unicità del nome. Se il caso risulta essere il secondo, gli attributi locali contenuti nel vettore ***VctLocAtt*** vengono inseriti nella ***MapEllO***, quindi nella classe globale dello schema inziale riferita al *gOldAtt*.

Nel terzo caso, le soluzioni possibili possono essere diverse, da valutare caso per caso:

- ✓ Si possono unire gli attributi locali della nuova sorgente con solo una degli *gOldAtt* trovati;
- ✓ Si possono unire tutti gli attributi globali trovati insieme agli attributi locali della nuova sorgente;
- ✓ Si possono tenere divisi tutti gli *gOldAtt,* mentre gli attributi locali della nuova sorgente vengono uniti in una nuova classe globale;
- ✓ Si può non mappare *lAttNew* senza modificare attributi globali della vecchia vista.

Inizialmente l'algoritmo implementava la prima soluzione, cioè univa tutti gli *lAttNew* in uno degli attributi globali *gAttOld* trovati. La scelta di questo ultimo non era pilotata, ma cadeva sul primo che veniva trovato. Questa soluzione è stata poi

scartata perché corretta solo in caso di relazioni di similitudine, purtroppo sperimentando si è visto che questo tipo di situazione nasce soprattutto quando si hanno relazioni di specializzazione e generalizzazione. Consideriamo ad esempio il caso in cui si ha una annotazione su due attributi globali di questo tipo: uno viene annotato come "firstname" mentre l'altro come "lastname". Questa genera relazioni nel Common Thesaurus che separano i due attributi nella *GVVold*. Se un attributo locale della nuova sorgente viene annotato come "name", la relazione di generalizzazione, può portate all'unione di questo attributi. In questo caso sarebbe stato allora più corretto unire tutti gli attributi nella *GVVold*, anche se questo comporta una modifica dello schema della *GVVold* che non tiene conto delle relazioni interne al Common Thesaurus in essa contenuto. Per questi motivi, anche questa soluzione è stata scartata, stessa cosa per la quarta proposta, di non mappare gli attributi locali relativi alla nuova sorgente, per rendere meno gravoso e pesante il lavoro manuale del progettista. L'algoritmo adotta quindi la terza soluzione, unendo gli attributi locali relativi alla nuova sorgente in un nuovo attributo globale, anche in questo caso la funzione esegue un controllo di unicità del nome.

4.2.5 Aggiornamento del Common Thesaurus

Il sistema conclude il processo di integrazione di due versioni della stessa ontologia con l'aggiornamento del Common Thesaurus. Questa azione viene chiamata direttamente dalla funzione ***UpdateGVVold***, entra in esecuzione, quindi, un'unica volta alla conclusione del ciclo di comparazione delle due GVV. L'aggiornamento del Common Thesaurus viene coordinata dalla ***funzione updateThesaurus(),*** la quale si avvale dell'utilizzo di più funzioni, metodi della classe Comparatore, per gestire le diverse tipologie di relazione che si possono presentare, come verrà successivamente descritto.

```
public void updateThesaurus(){
    ThesRelation[] trNG = _schemaGVVnew.getAllThesRelation();
    String sname1 = null;
    //variabili booleane per il controllo di corrispondenza con la GVVold
    boolean first, second;
    for(int i=0; i< trNG.length ; i++){
```

```
        if(trNG[i].getSourceName1().equals("GVVold"))
            first = true;
        else
            first = false;
        if(trNG[i].getSourceName2().equals("GVVold"))
            second = true;
        else
            second = false;
        if(first){
            if(!second){
                if(trNG[i] instanceof AttributeRel){
                    update1AttributeAttributeRel((AttributeRel)trNG[i]);
                }else{
                    if(trNG[i] instanceof AttrIntRel){
                        update1InterfaceAttrIntRel((AttrIntRel)trNG[i]);
                    }else{
                        if(trNG[i] instanceof InterfaceRel);
                            update1InterfaceInterfaceRel((InterfaceRel)trNG[i]);
                    }
                }
            }
        }else{
            if(second){
                if(trNG[i] instanceof AttributeRel){
                    update2AttributeAttributeRel((AttributeRel)trNG[i]);
                }else{
                    if(trNG[i] instanceof AttrIntRel){
                        update2AttributeAttrIntRel((AttrIntRel)trNG[i]);
                    }else{
                        if(trNG[i] instanceof InterfaceRel);
                            update2InterfaceInterfaceRel((InterfaceRel)trNG[i]);
                    }
                }
            }else
                _schemaGVVold.addThesRelation(trNG[i]);
        }
    }
}
```

```
/**funzione che restituisce l'array  degli attributi globali data la classe globale*/
private GlobalSimpleAttribute[] findGlobalAttr (GlobalInterface GlCls){
    GlobalSimpleAttribute[] GlAtt = null;
    GlobalIntBody[] GlIntBody = GlCls.getGlobalIntBodiesOrderByName();
   for (int y=0; y < GlIntBody.length; y++){
      GlAtt = GlIntBody[y].getGlobalAttributes();
  }
   return (GlAtt);
}
```

Funzione che gestisce l'aggiornamento del Common Thesaurus della *GVVadd*

Il metodo inizia analizzando le sorgenti dei due concetti legati dalla relazione terminologica (***ThesRelation***) per verificare se sono concetti globali della *GVVold*, e memorizza il valore trovato in due variabili booleane. Questo studio può portare i seguenti risultati:

- ✓ entrambi i concetti sono elementi informativi della nuova sorgente;
- ✓ entrambi i concetti rappresentano un concetto globale della *GVVold*.
- ✓ solo uno dei due concetti è un elemento informativo della nuova sorgente mentre l'altro rappresenta un concetto globale della *GVVold*;

Le prime due situazioni sono le più semplici da gestire: nel primo caso si ha un nuovo legame che non richiede alcun tipo di completamento con le informazioni delle sorgenti della *GVVold*, perché espresso su concetti direttamente collegati a sorgenti locali, il Common Thesaurus della *GVVold* viene quindi aggiornato con l'inserimento della nuova relazione. Al contrario, il secondo caso non produce alcuna modifica, si suppone, infatti, che questo tipo di relazione sia una espressione globale di altre già presenti nel Common Thesaurus della *GVVold*, espresse in forma più specifica sulle *lsorceOld*.

L'ultimo caso, invece, richiede una attenzione maggiore: queste relazioni, che legano concetti globali provenienti da informazioni sulla vecchia GVV e concetti della nuova sorgente, devono essere completate, sostituendo le informazioni globali con quelle specifiche alle sorgenti della *GVVold*. L'integrazione viene implementata in una successione di passaggi:

[1] individuazione della classe o attributo globale di riferimento nella *GVVold*;

[2] estrapolazione del mapping con le *lsorceOld*;

[3] sostituzione del concetto globale con quelli locai relativi alle *lsorceOld*;

[4] inserimento della nuova relazione nel Common Thesaurus della *GVVold*.

Queste azioni vengono eseguite nei metodi prima citati, in base al tipo di relazione terminologica presente. In ODL^3, infatti, le relazioni tra concetti di una ontologia possono rappresentare un legame: tra due interface (***InterfaceRel***), tra due attributi (***AttributeRel***) oppure tra una interface e un attributo (***IntAttrRel***).

```
//aggiorna le relazioni tra attributi con sostituzione nel primo attributo
private void update1AttributeAttributeRel (AttributeRel oldrel){
    AttributeRel newrel = new AttributeRel();
    String nameGC = oldrel.getIName1();
    String nameGA = oldrel.getAName1();
    GlobalInterface GlInt = findGlobalClass (nameGC, _GlobalClassGVVold);
    GlobalSimpleAttribute[] GlAttr = findGlobalAttr(GlInt);
    Attribute[] locAtt = null;
    for(int s=0; s<GlAttr.length; s++){
        if(GlAttr[s].getName().equals(nameGA)){
            MappingElement MapEll = GlAttr[s].getMappingElement();
            try {
                locAtt = MapEll.getLocalAttributesSortByDottedName();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    if(locAtt!=null){
        for(int s=0; s<locAtt.length; s++){
            newrel= (AttributeRel)oldrel.clone();
            newrel.setAttribute1(locAtt[s]);
            _schemaGVVold.addThesRelation(newrel);
        }
    }else {
        message = "Errore: attributi locali non trovati. Attributo globale: "+
                        nameGC + "."+ nameGA;
        System.out.println(message);
```

```
    }
}
//aggiorna le relazioni tra attributi con sostituzione nel secondo attributo
private void update2AttributeAttributeRel (AttributeRel oldrel){
    AttributeRel newrel = new AttributeRel();
    String nameGC = oldrel.getIName2();
    String nameGA = oldrel.getAName2();
    GlobalInterface GlInt = findGlobalClass (nameGC, _GlobalClassGVVold);
    GlobalSimpleAttribute[] GlAttr = findGlobalAttr(GlInt);
    Attribute[] locAtt = null;
    for(int s=0; s<GlAttr.length; s++){
        if(GlAttr[s].getName().equals(nameGA)){
            MappingElement MapEll = GlAttr[s].getMappingElement();
            try {
                locAtt = MapEll.getLocalAttributesSortByDottedName();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    if(locAtt!=null){
        for(int s=0; s<locAtt.length; s++){
            newrel= (AttributeRel)oldrel.clone();
            newrel.setAttribute2(locAtt[s]);
            _schemaGVVold.addThesRelation(newrel);
        }
    }else {
        message = "Errore: attributi locali non trovati. " + "attributo globale" +
                        nameGC + "."+ nameGA;
        System.out.println(message);
    }
}
//aggiorna le relazioni tra istanza (appartenente alla GVVold) e un attributo
private void update1InterfaceAttrIntRel(AttrIntRel oldrel){
    AttrIntRel newrel = new AttrIntRel();
    String nameGC = oldrel.getIName1();
    GlobalInterface GlInt = findGlobalClass (nameGC, _GlobalClassGVVold);
    Interface[] LcInt = GlInt.getAllLocalInterfacesOrderByName();
    if(LcInt!=null){
```

```
        for(int s=0; s<LcInt.length; s++){
            newrel= (AttrIntRel)oldrel.clone();
            newrel.setInteface1(LcInt[s]);
            _schemaGVVold.addThesRelation(newrel);
        }
    }else {
        message = "Errore: classi locali non trovate. Classe globale"+ nameGC;
        System.out.println(message);
    }
}
//aggiorna le relazioni tra istanza e un attributo (appartenente alla GVVold)
private void update2AttributeAttrIntRel (AttrIntRel oldrel){
    AttrIntRel newrel = new AttrIntRel();
    String nameGC = oldrel.getIName2();
    String nameGA = oldrel.getAName2();
    GlobalInterface GlInt = findGlobalClass (nameGC, _GlobalClassGVVold);
    GlobalSimpleAttribute[] GlAttr = findGlobalAttr(GlInt);
    Attribute[] locAtt = null;
    for(int s=0; s<GlAttr.length; s++){
        if(GlAttr[s].getName().equals(nameGA)){
            MappingElement MapEll = GlAttr[s].getMappingElement();
            try {
                locAtt = MapEll.getLocalAttributesSortByDottedName();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    if(locAtt!=null){
        for(int s=0; s<locAtt.length; s++){
            newrel= (AttrIntRel)oldrel.clone();
            newrel.setAttribute2(locAtt[s]);
            _schemaGVVold.addThesRelation(newrel);
        }
    }else {
        message = "Errore: attributi locali non trovate. " + "attributo globale"+
                    nameGC + "."+ nameGA;
        System.out.println(message);
    }
```

```
}
//aggiorna le relazioni tra istanze (la prima appartenente alla GVVold)
private void update1InterfaceInterfaceRel (InterfaceRel oldrel){
    InterfaceRel newrel = new InterfaceRel();
    String nameGC = oldrel.getIName1();
    GlobalInterface GlInt = findGlobalClass (nameGC, _GlobalClassGVVold);
    Interface[] LcInt = GlInt.getAllLocalInterfacesOrderByName();
    if(LcInt!=null){
        for(int s=0; s<LcInt.length; s++){
            newrel= (InterfaceRel)oldrel.clone();
            newrel.setInterface1(LcInt[s]);
            _schemaGVVold.addThesRelation(newrel);
        }
    }else {
        message = "Errore: classi locali non trovate. Classe globale"+ nameGC;
        System.out.println(message);
    }
}
//aggiorna le relazioni tra istanze (la seconda appartenente alla GVVold)
private void update2InterfaceInterfaceRel (InterfaceRel oldrel){
    InterfaceRel newrel = new InterfaceRel();
    String nameGC = oldrel.getIName2();
    GlobalInterface GlInt = findGlobalClass (nameGC, _GlobalClassGVVold);
    Interface[] LcInt = GlInt.getAllLocalInterfacesOrderByName();
    if(LcInt!=null){
        for(int s=0; s<LcInt.length; s++){
            newrel= (InterfaceRel)oldrel.clone();
            newrel.setInterface2(LcInt[s]);
            _schemaGVVold.addThesRelation(newrel);
        }
    }else {
        message = "Errore: classi locali non trovate. Classe globale"+ nameGC;
        System.out.println(message);
    }
}
```

Funzione che gestisce la sostituzione delle relazioni da aggiornare nel Common Thesaurus della *GVVadd*

CONCLUSIONI

Il lavoro di questa tesi ha portato alla realizzazione di un tool, all'interno del sistema MOMIS, per l'inserimento di una nuova sorgente in una Global Virtual View già esistente. Questo applicativo, rientra in quello che si può definire il campo della "dinamica delle ontologie" ed è stato svolto nell'ambito del progetto SEWASIE.

Nel precedente capitolo della tesi è stata quindi riportata una descrizione dell'algoritmo realizzato, che implementa una metodologia di fusione tra due versioni della stessa ontologia, per integrare quella iniziale con le nuove informazioni acquisite. Sono state particolarmente evidenziate le problematiche che possono insorgere nella fusione di due viste: mantenimento delle informazioni precedentemente raccolte, e coerenza con le sorgenti di partenza, in particolar modo se si generano situazioni richiedenti importanti modifiche allo schema globale iniziale.

La politica scelta, specialmente per quanto riguarda l'integrazione tra classi globali iniziali, è stata quella di lasciare massima libertà decisionale al progettista, dandogli la possibilità di valutare caso per caso la soluzione da adottare. Una possibile alternativa potrebbe essere quella di ricercare all'interno dei Common Thesaurus delle due GVV le relazioni, specificate sulle sorgenti, che hanno causato tali situazioni e ricreare il cluster su queste, evitando di apportare modifiche sulla base di soli concetti globali.

L'integrazione è stata verificata su alcune sorgenti al fine di valutarne i tempi di esecuzione, il risultato, nonostante la presenza di ripetuti cicli di confronto, e' stato buono, cioe' i tempi di computazione sono ampiamente accettabili.

Lo sviluppo di questo applicativo è stato preceduto da un lavoro di studio su quali siano gli approcci in letteratura sul tema della di "dinamica di una ontologia", di conseguenza sono stati riportati e descritti i due approcci di "evoluzione" e "con versioni", dando particolare risalto, soprattutto al primo. Fra queste due sono state affrontate in modo piu' esaustivo: il pruning di una ontologia, cioè la riduzione delle sue dimensioni dai concetti ritenuti ridondanti e il mapping di più ontologie attraverso le tecniche di merge e di allineamento.

Nel pruning di una ontologia è stato presentato un possibile algoritmo per gestire la cancellazione di un concetto e l'effetto di propagazione derivante da questa azione, questo è da intendersi come primo spunto per gestire, nel sistema MOMIS, sia la modifica di sorgenti già presenti, sostituendo i concetti da modificare con quelli nuovi, che per garantire una corretta eliminazione di quelli ritenuti obsoleti. Si è inoltre, evidenziato come la difficoltà maggiore, non sia tanto nella fase di pruning, ma nell'individuazione di quali siano i concetti rilevanti che costituiscono il *CoI*, insieme dei concetti di interesse.

Per quanto concerne i due approcci di mapping si è visto come siano intimamente collegati tra loro, il merge risulta essere, infatti, l'estensione dell'allineamento, è emerso inoltre come la scelta sul loro utilizzo sia demandata al progettista in base alla complessità del proprio specifico dominio di lavoro.

Un'ulteriore problematica, non affrontata in questa tesi, che richiede ulteriori studi nel futuro e' come mantenere traccia delle modifiche effettuate, cioe' risucire a stabilire il versioning. Il versioning, in questo contesto definito anche come "ontology evolution", consiste nell'abilità di gestire i cambiamenti dell'ontologia ed i loro effetti attraverso la creazione ed il mantenimento di diverse varianti dell'ontologia stessa.

APPENDICE

DINAMICA DI SCHEMI RELAZIONALI

La problematica della gestione dinamica dei dati, discussa in questa tesi riguardante l'evoluzione della struttura di una ontologia, non è argomento nuovo nel campo della gestione delle informazioni, dal momento che i database e i sistemi software hanno strutture complesse e sono sottoposti a continui cambiamenti durante il loro ciclo vitale. I problemi di evoluzione di schema e di versioning nascono nel contesto dei database a "lunga vita" dove i dati memorizzati sono considerati degni di sopravvivere ai cambiamenti dello schema del database. Quindi modifiche allo schema del database sono comuni ma spesso causa di notevoli problemi nell'amministrazione del database. Si hanno quindi due distinte definizioni per affrontare questa problematica:

- ✓ Schema evolution: un sistema di database sopporta l'evoluzione di schema se permette modificazioni dello schema del database senza la perdita dei dati esistenti e non è richiesto nessun supporto per gli schemi precedenti.
- ✓ Schema versioning: un sistema di database supporta il versioning di schema se permette richieste su tutti i dati attraverso interfacce di versioni definibili dall'utente.

Nel campo relazionale sono sorte molte proposte riguardanti lo schema versioning come naturale estensione degli studi sui modelli di dati temporali e i database temporali, i quali sono stati più propriamente classificati come lavori sul versioning di schema temporale. I database temporali permettono il mantenimento della storia dei dati attraverso il supporto di una semantica temporale a livello di

sistema, in particolare sono solitamente considerate due dimensioni: il *tempo di validità* che rappresenta il tempo del mondo reale ed è solitamente dato dall'utente e il *tempo di transazione* che è il tempo del sistema. Negli anni passati uno dei più significativi risultati ottenuti nel versioning di schema temporale è stata l'introduzione del versioning di schema ottenuto da Roddick e Snodgrass nel linguaggio standard temporale TSQL2 [33], in particolare è stato aggiunto uno schema temporale ortogonale alle esistenti funzionalità temporali per consentire di gestire i cambiamenti di struttura di database e le specificazioni di un determinato schema attraverso il quale i dati sono recuperati.

Benché molti articoli di ricerca siano stati dedicati all'approccio di schema evolution, Killiman in [34] ha suggerito che

> *"Schema evolution is versioning of types. It's very difficult and very expensive to support schema evolution without versioning"*[20]

Nella stessa discussione Anderson ha commentato che

> *"Schema evolution can be seen as an use of versioning management"*[21]

Quindi le precedenti definizioni indicano che le risorse offerte dallo schema versioning sono comprese in quelle dell'evoluzione di schema, cioè lo schema evolution può essere considerato come un caso particolare dello schema versioning dove solo la versione corrente è mantenuta, da questo in [35] si è visto come la definizione della semantica del cambiamento di schema può garantire la correttezza del processo e più in generale l'interazione tra l'evoluzione dello schema e il database di riferimento. È stato definito un modello formale, $OODM|_{SV}$ (Object Oriented Data Model contextualized to Schema Version), per gestire lo schema versioning temporale in database orientati ad oggetti. Questo modello mantiene diverse versioni dello schema ed esegue interrogazioni su di esse, creando ad ogni modifica strutturale un nuovo schema. E' stata quindi definita una semantica operazionale, cioè come un

[20] Lo Schema evolution è il versioning di tipi. E' molto difficile e molto costoso supportare lo schema evolution senza il versioning.

[21] Lo schema evolution può essere vista come l'utilizzo di una gestione di versioni.

database deve essere modificato per essere coerente con la nuova versione di schema. Questo ha portato a due problematiche: la modifica semantica e la propagazione dei cambiamenti.
La prima può essere affrontata seguendo due approcci distinti: l'adozione di invarianti e regole oppure l'introduzione di assiomi.

Nel primo gli invarianti definiscono la consistenza di uno schema e le regole definite devono essere seguite per mantenere gli invarianti soddisfatti dopo ogni cambiamento di schema.

Nel secondo un completo set di assiomi, provvisto con un meccanismo di inferenza, formalizza la dinamica di evoluzione di uno schema, cioè l'attuale gestione dei cambiamenti dello stesso a seguito di una operazione. L'accoppiamento delle primitive di cambio di schema con gli assiomi, assicura automaticamente la consistenza di uno schema senza la necessità di formalizzare un esplicito controllo, quindi versioni non corrette di uno schema non possono essere generate.

La seconda problematica può essere affrontata seguendo quattro approcci fondamentali:

a) coercizione o conversione immediata, cioè un istantanea conversione di oggetti;

b) screening o conversione deferita, i cambiamenti sono propagati attraverso una conversione ritardata degli oggetti;

c) filtering, i cambiamenti non sono mai propagati, quindi gli oggetti sono assegnati a differenti versioni di uno schema in accordo alla propria semantica;

d) hybrid, si possono combinare uno o più degli approcci precedenti.

In ogni caso si possono utilizzare semplici meccanismi di default, oppure definire funzioni di conversione che richiedono l'intervento dell'utente per evitare l'aggiornamento di oggetti errati. In [36] viene introdotto un modello assiomatico per la propagazione dei cambiamenti che identifica in maniera dichiarativa il set di oggetti che sono stati influenzati dal cambiamento di schema.

Un altro approccio può essere quello di utilizzare per la validazione dello schema un linguaggio di programmazione, il vantaggio è dovuto alla formale nozione di consistenza dovuta al linguaggio, che quindi comporta meccanismi statici di consistenza.

Un ulteriore approccio è quello di utilizzare degli algoritmi per analizzare i cambiamenti, comparando due versioni dello schema estraendone regole per la propagazione dei cambiamenti degli oggetti esistenti.

Detto ciò risulta immediato un confronto con lo studio riportato nel capitolo tre, riguardante la dinamica delle ontologie. Anche in questo caso si è visto come sia possibile affrontare questa problematica seguendo due strade: una basata sull'evoluzione di una ontologia, l'altra basata sulle diverse versioni della stessa.

Nel campo delle ontologie, parlare di evoluzione significa mantenere una struttura aggiornata sulla concettualizzazione del dominio che si vuole rappresentare, questa infatti, per il tipo di ontologia a mediatore, non può portare a perdite di dati esistenti. La modifica di concetti appartenenti allo schema di una ontologia, si suppongano relativi ad un database di tipo relazionale, non comporta la modifica di tali informazioni all'interno del database ma la loro rappresentazione in una vista globale che li rappresenta. Allo stesso modo la cancellazione, affrontata nell'algoritmo di pruning, di concetti interni ad una ontologia non comporta l'eliminazione di informazioni interne a database ma di concetti obsoleti o ridondanti nella concettualizzazione che li vuole rappresentare. Possono essere anche in questo campo adattate le affermazioni di Killiman ed Anderson, cioè l'evoluzione di una ontologia non può non passare dalla gestione delle diverse versioni di questa ultima. L'applicazione sviluppata nell'ambito di questa tesi ne è un esempio, infatti, l'evoluzione dell'ontologia sviluppata nel progetto MOMIS, riguardante l'inserimento di una nuova sorgente senza la perdita di informazioni precedentemente raccolte, si è basata sul confronto e il successivo mapping di due ontologie rappresentanti due versioni successive della stessa.

BIBLIOGRAFIA

[1] P. Ceravolo, "Cos'è e a cosa serve il Web semantico". *In http://pro.html.it/index.asp (2003)*

[2] http://www.sewasie.org, *Semantic Webs and AgentS in Integrated Economies*

[3] http://www.w3c.it/w3cin7punti.html, *World Wide Web Consortium, 7 punti per descrivere gli obiettivi e i principi strategici del W3C.*

[4] R. Capezzera, S. Bergamaschi, M. Vincini, "Query Processing by Semantic Reformulation". *Tesi di laurea presso l'Università degli Studi di Modena e Reggio Emilia, facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, pp 34-40* (2003)

[5] D.Beneventano, S. Bergamaschi, F.Guerra and M. Vincini, "Synthersizing an intergraded ontology". *In IEEE Internet Computing Magazine* (2003)

[6] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, M. Vincini, "A Semantic Approach to Information Integration: the MOMIS project". *Sesto Convegno della Associazione Italiana per l'Intelligenza Artificiale, AI*IA 98, Padova IT* (Settembre 1998)

[7] F.Guerra, "Dai Dati all'Informazione: il sistema MOMIS". *Tesi di dottorato di ricerca in Ingegneria dell'Informazione presso l'Università degli Studi di Modena e Reggio Emilia* (2003)

[8] Gio Wiederhold "Mediators in the Architecture of Future Information Systems". *In IEEE Computer 25(3): pp 38-49* (1992)

[9] N. Guarino, P. Giaretta "Ontologies and knowledge bases: Towards a terminological clarification". *In Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, N. Mars (ed.), pp 25-32. IOS Press, Amsterdam,* (1995)

[10] T. R. Gruber, "A translation approach to portable ontology specifications". *In Knowledge Acquisition, Vol. 5, No. 2, pp. 199* (1993)

[11] W. N. Borst "Construction of Engineering Ontologies for Knowledge Sharing and Reuse". *PhD dissertation, University of Twente, The Netherlands,* (1997).

[12] M. Uschold, "Knowledge level modelling: Concepts and terminology". *In the Knowledge Engineering Review" pp. 5.* (1998)

[13] O. Lassila and D. McGuinness, "The role of frame-based representation on the semantic web". *Electronic Transactions on Artificial Intelligence (ETAI) Journal: area The Semantic Web, To appear,* (2001)

[14] L. Console, E. Lamma, P. Mello, M.Milano. Libro: "Programmazione Logica e Prolog", *Seconda Edizione UTET editore*

[15] D.Beneventano, S.Bergamaschi, C.Sartori, And M.Vincini. "ODB-Tools: A description logics based tol for schema validation and semantic query optimization in object oriented databases". *In Proc. Of. Int. Conf. on Data Engineering, ICDE'97, Birmingham, UK,* (April 1997).

[16] M. Klein and D. Fensel. "Ontology versioning on the semantic web". *In International Semantic Web Working Symposium (SWWS), pp. 75-91, Stanford University, USA* (July 30 - August 1 2001)

[17] L. Stojanovic, A. Maedche, B. Motik and N. Stojanovic, **"**User-driven Ontology Evolution Management". *In EKAW02 Siguena,Spain pp. 285-300* (Ottobre 2002)

[18] J. D. Heflin, "Towards the Semantic Web: Knowledge representation, in a dynamic, distributed environment". *PhD thesis University of Maryland, Collega Park,MD, pp. 22-47 Usa* (2001)

[19] P. De Leenheer, "Revising and Managing Multiple Ontology Versions in a Possible Worlds Settino". *In Proceedings of the OTM 2004 Workshops, LNCS, Springer Verlag.* (2004)

[20] M. Klein and D. Fensel. "Ontology versioning for the Semantic Web". *In Proceedings of theInternational Semantic Web Working Symposium (SWWS), pp. 75 – 91, Stanford University, California, USA* (July 30 – Aug. 1, 2001)

[21] Michel Klein, Dieter Fense, Atanas Kiryakov, and Damyan Ognyanov "Ontology versioning and change detection on the Web". *EKAW02 Siguena,Spain pp. 197-212* (Ottobre 2002)

[22] P.Scaruffi. Libro: "La mente Artificiale", *edizione Franco Angeli (1991) disponibile sul sito www.thymos.com*

[23] P. Gardenfors and C. J. Van Rijsbergen. Libro:"Belief Revision" *edizione Cambridge University Press* (dicembre 2003)

[24] J. Conesa Caralt, "Ontology-Driven Information Systems: Pruning and Refactoring of Ontologies" . *In UML'04 Doctoral Symposium* (2004)

[25] Conesa, J. and A. Olivé, "A General Method for Pruning OWL Ontologies". *In Ontologies Databases and Applications of Semantics'04 (ODBAS'04). Larnaca, Cyprus: Springer.* (Ottobre 2004)

[26] Conesa, J. and A. Olivé. "Pruning Ontologies in the Development of Conceptual Schemas of Information Systems". *International Conference on Conceptual Modeling ER2004. Shangai, China,* (2004)

[27] Larman, C., "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design". *Prentice Hall. pp. 507* (1998)

[28] D. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd. "Cyc: Toward programs with common sense". *Communications of the ACM, 33(8):30--49* (August 1990)

[29] R. Volz, R. Studer, A. Maedche, and B. Lauser. "Pruning-based Identification of Domain Ontologies". *In Journal of Universal Computer Science volume 9/ issue 6:520-529* (June 2003)

[30] J.De Bo, P.Spyns, and R.Meersman "Asisting Ontology Integration with Existing Thesauri" *In, Meersman R., Zahir T. et al.,(eds.), On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE (part I), LNCS 3290, pp. 801 - 818, 2004. Springer Verlag.* (2004)

[31] K.X.S. de Souza and J.Davis "Aligning Ontologies and Evaluating Concept Similarities" , *Ontologies Databases and Applications of Semantics'04 ODBASE'04,* (October 2004)

[32] A.Fergnani, S. Bergamaschi, F.Guerra, "Ontology dynamics for Semantic Web: the MOMIS approach". *Tesi di laurea presso l'Università degli Studi di Modena e Reggio Emilia, facoltà di Ingegneria, corso di laurea in Ingegneria Informatica.* (2002)

[33] R.T.Snodgrass, editor, "The TSQL2 Temporal Query Language". *Kluwer Accademic Publishing, New York* (1995)

[34] J.Joseph, S. Thatte, C.W.Thompson and D.L.Wells, "Report on the Object-Oriented Database Workshop". *Held in Conjunction with OOPSLA'88. SIGMOD Record, 18(3) pp. 78-101* (1989)

[35] F.Mandreoli and F.Grandi "A Formal Model for Temporal Schema Versioning in Object Oriented Databases". *In Data tecnoleg Engineering vol. 46 n.2* (2003)

[36] R. J. Peters, K. Barker, "Change Propagation in an Axiomatic Model of Schema Evolution for Objectbase Management Systems". *In: H.Balsters, b. de Borck, S. Conrad (Eds.), "Database Schema Evolution and Meta-Modeling – Proc. Intl' Workshop on Foundations of Models and Languages for Data and Objects" FoMLaDO/DEMM2000, Selected papers, n.2065 in LNCS, Springer – Verlang pp. 142-162* (2001)

RINGRAZIAMENTI

Desidero ringraziare anzitutto la Professoressa *Sonia Bergamaschi* per il patrocinio e i consigli che ha saputo darmi nella stesura del presente documento. Un ringraziamento particolare va all'Ingegnere *Francesco Guerra* e all'Ingegnere *Daniele Miselli* per la loro incondizionata disponibilità e per l'aiuto che mi hanno dimostrato durante tutto il cammino di tesi.

Un GRAZIE di cuore ai miei genitori, *Maria Rosa* e *Carlo*, senza i quali non sarei mai arrivata a questo traguardo così importante per la mia vita: vi ringrazio per l'appoggio nei momenti di sconforto, per la vostra gioia nel condividere con me le tappe superate ed, in particolar modo, per essermi stata vicina nei momenti più difficili che hanno caratterizzato questo percorso.

Un Grazie con la G maiuscola a mia sorella *Ilaria* per avermi supportato ma soprattutto sopportato durante il cammino universitario: ora tocca a te ed il mio augurio è che tu riesca il prima possibile a raggiungere questa meta così bella ed importante.

Un ringraziamento profondo va a due persone a me care, *Diego* e *Marco*, perché con il loro amore sono riusciti a capire ciò che stavo affrontando; per il sempre presente appoggio durante le varie peripezie che hanno caratterizzato il raggiungimento di questo traguardo e l'aiuto che mi hanno regalato nelle decisioni da prendere nei momenti più difficili.

Un grazie a tutti i parenti per il loro sostegno, in particolare ai miei cugini *Nico*, *Patrizia* e *Giulia* ed ai miei carissimi "*nonna Rina*", "*nonno Matteo*" e "*nonna Bianca*".

Un grazie a *Don Gildo*, per avermi ascoltato e guidato soprattutto nei momenti più difficili.

Ringrazio tutti i miei amici con i quali ho intrecciato conoscenze diverse ma ugualmente importanti: ogni ricordo con voi è un piccolo passo nel cammino della mia vita e quindi anche di questa mia esperienza universitaria.

Un grazie a tutte le mie meravigliose amiche, compagne di week-end e non solo, *Cinzia*, *Cristina*, *Tania*, *Elisa*, *Federica*, *Sara M.*, *Sara F.*, *Silvia V.*, *Marzia*, *Sara C.*, *Eleonora*, *Silvia B.*, *Alessia*, *Mariangela* e *Chiara*: vi ringrazio perché "la vita non è nulla senza l'amicizia".

Al gruppo di Modena, *Vitto*, *Paga*, *Oscar*, *Marina*, *Maria Grazia*, *Marco*, *Lucio*, *Luca*, *Francesca*, *Fausto B.*, *Fausto*, *Enrico*, *Erica*, *Elisa, Davide* e *Claudio*, un grazie per le serate e le cene che hanno caratterizzato questo ultimo anno.

Un grazie particolare ai gruppi A.C.R "I Pulcini" e "I Delfini", a tutti i bambini e gli educatori, *Annarita*, *Cinzia*, *Maxi*, *Rita* e *Don Antonio* con i quali sto condividendo questa entusiasmante esperienza, a volte anche fonte di una buona scusa per staccare dallo studio.

Grazie anche a *Simona*, mia socia in affari nel "Bar della Calcetto", e al trio *Sauro*, *Gaspa* e *Christian*.

Ringrazio inoltre tutti gli amici della parrocchia di Cibeno: ho la fortuna che siete talmente tanti da non avere spazio per inserire tutti i vostri nomi, un GRAZIE davvero perché siete tutti ugualmente e straordinariamente importanti per me.

Un grazie di cuore a tutti coloro con cui ho condiviso la gioia dello spirito universitario, ho avuto la fortuna di conoscere tantissime persone, risulta quindi impossibile elencarvi tutti. Vorrei tuttavia esprimere un grazie particolare a *Marianna*, insieme abbiamo iniziato e concluso questa avventura che ci ha fatte conoscere e ci ha unito in una profonda amicizia che spero ci sosterrà anche in futuro; *Sara* per le cene universitarie organizzate insieme, rimarrà con profonda soddisfazione il ricordo di "One night for Vod"; *Daniele* mio esilarante compagno di viaggio; *Simona* per la sua dolcezza infinita; tutti i miei compagni di tesine *Pedro*, *Antonio*, *Devid*, *Fabio*, *Laura*, *Alessia* e *Gabriele*.

Un grazie anche a tutti i ragazzi del laboratorio *Denis*, *Enrica*, *Fabio, Francesca, Marco, Rossano* e *Simone* con cui ho condiviso il periodo ballerino di realizzazione della tesi, in particolare ricordo il nostro angelo *Erry*.

Infine vorrei veramente accomunare tutte le persone che ho citato, e non solo, per dirvi grazie di cuore per avermi sostenuto e sopportato durante tutta la mia carriera scolastica e per aver condiviso con me la gioia della Laurea.