

# Prototipo per l'aggiunta di una nuova sorgente a una ontologia di dominio

Domenico Beneventano, Sonia Bergamaschi,  
Stefania Bruschi, Francesco Guerra,  
Mirko Orsini, Maurizio Vincini

## 1 Linee guida: inserimento di una nuova sorgente

Il processo di integrazione di più sorgenti all'interno della Global Virtual View in MOMIS<sup>1</sup> prevede un successione di passaggi per consentire all'utente di creare il proprio schema delle concettualizzazioni. Il primo passo è 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 è 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 informazioni di mapping memorizzate poiché 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.

---

<sup>1</sup> MOMIS, Mediator environment for Multiple Information Sources [Benv03, Berg01, ManV2] è un sistema realizzato dal DBGroup, presso l'Università degli Studi di Modena e Reggio Emilia, per integrare e interrogare sorgenti di dato eterogenee, cioè informazioni provenienti da differenti sorgenti indipendente dalla localizzazione e dalla natura delle sorgenti stesse, attraverso la realizzazione di una vista globale (Global Virtual View o GVV).

Il passaggio da una versione già strutturata di una ontologia,  $\Omega_1$ , ad un'altra,  $\Omega_2$ , può avvenire attraverso varie trasformazioni. Secondo la teoria AGM<sup>2</sup>, 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.

La fusione di più ontologie in una unica globale nasce 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  $\Omega_{Merge}$ , partendo da due ontologie indipendenti tra loro,  $\Omega_{Old}$  e  $\Omega_{New}$ , anche se rappresentanti entrambe due versioni della stessa vista globale:

- ✓  $\Omega_{Old}$ , è l'ontologia iniziale, creata sulle vecchie sorgenti, da integrare con i nuovi elementi di informazione;
- ✓  $\Omega_{New}$ , è una ontologia creata ex-novo, che contiene sia le informazioni delle nuove sorgenti che i concetti definiti "globali" relativi alla ontologia iniziale  $\Omega_{Old}$ , 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  $\Omega_{Merge}$ : integrare  $\Omega_{New}$  con le informazioni raccolte nella precedente versione, sostituendo, quindi, i

---

<sup>2</sup> 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" [ Gard03]

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  $\Omega_{New}$ . 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)  $\Omega_{Merge}$  non deve ridefinire una classificazione tra le sorgenti appartenenti alla  $\Omega_{Old}$ , 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  $\Omega_{Old}$ : classi e attributi locali.
- c) l'integrazione della nuova sorgente deve avvenire partendo dalla  $\Omega_{Old}$ , 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  $\Omega_{Old}$ , 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  $\Omega_{Old}$  a concetti definiti “locali” della  $\Omega_{New}$ , tali relazioni inter-schema esistono per il solo fatto che l’una,  $\Omega_{New}$ , ingloba una vista globale della prima,  $\Omega_{Old}$ . 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  $\Omega_{Old}$ , 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  $\Omega_{Old}$ , 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  $\Omega_{Old}$ , questi possono essere definiti come il risultato di una classificazione di concetti locali tali per cui :

$$c_{glob_n} = \{c_{local_n^1}, c_{local_n^2}, \dots, c_{local_n^m}\}$$

L’inserimento di un 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 1. Accettare questo tipo di cambiamento a priori, cancellare quindi  $c_{glob_2}$  senza valutare quale siano le relazioni della  $\Omega_{New}$  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  $\Omega_{Old}$ , 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.

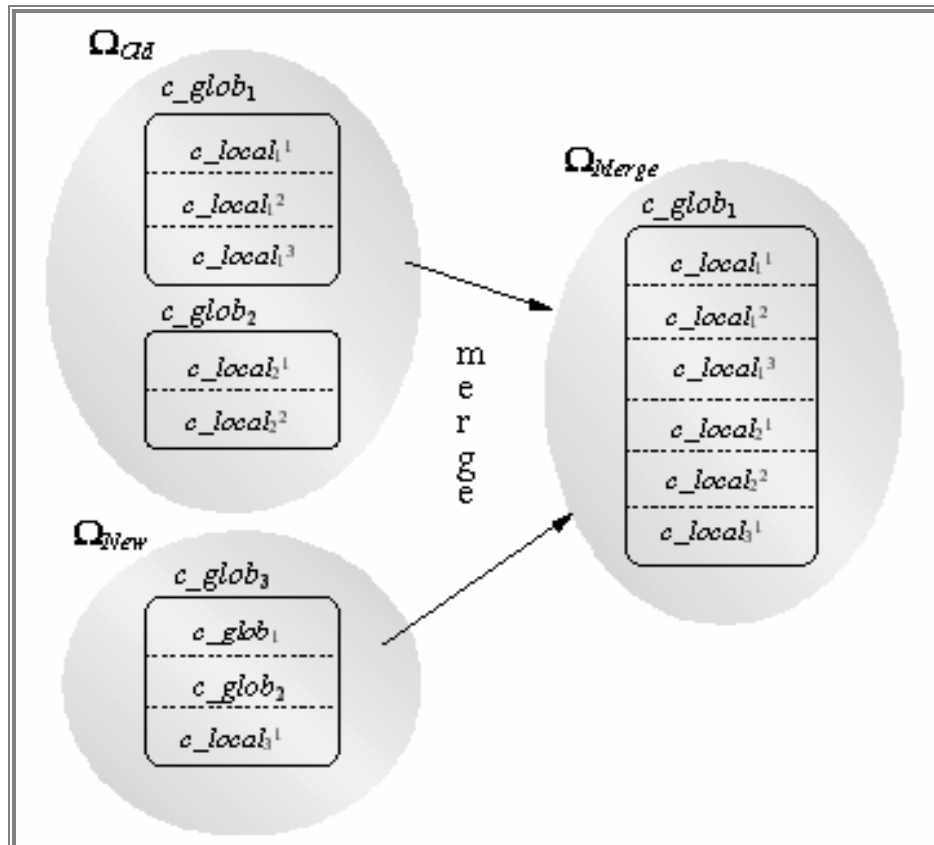


Figura 1: Violazione di uno dei presupposti iniziali: modifica dello schema iniziale

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  $\Omega_{Merge}$  con *GVVadd*, mentre con il termine *GVVold* e *GVVnew* si farà riferimento alle due ontologie al confronto, rispettivamente relative alla  $\Omega_{Old}$  e alla  $\Omega_{New}$ .

## 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 [Brus05].

## 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 una nuova interfaccia grafica strutturalmente e graficamente simile a quella precedente, alla quale viene passata la *GVVold* trasformata in sorgente locale, come mostrato in figura 2.

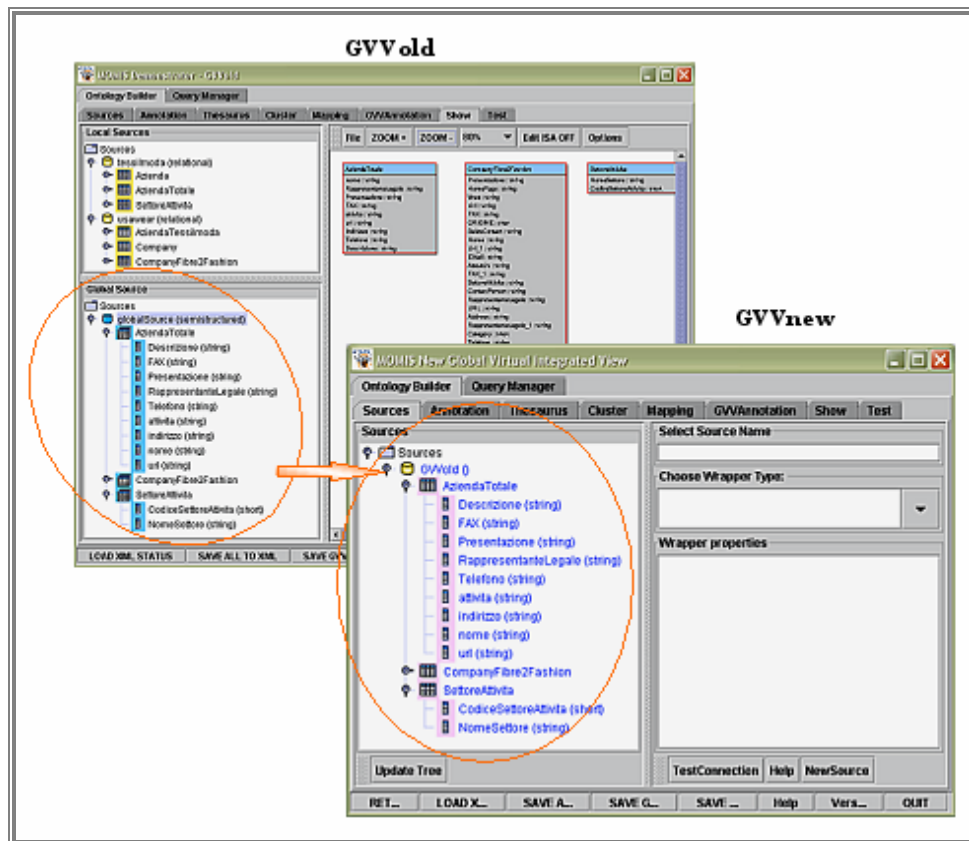


Figura 2: Passaggio della *GVVold* come sorgente locale alla *GVVnew*

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*.

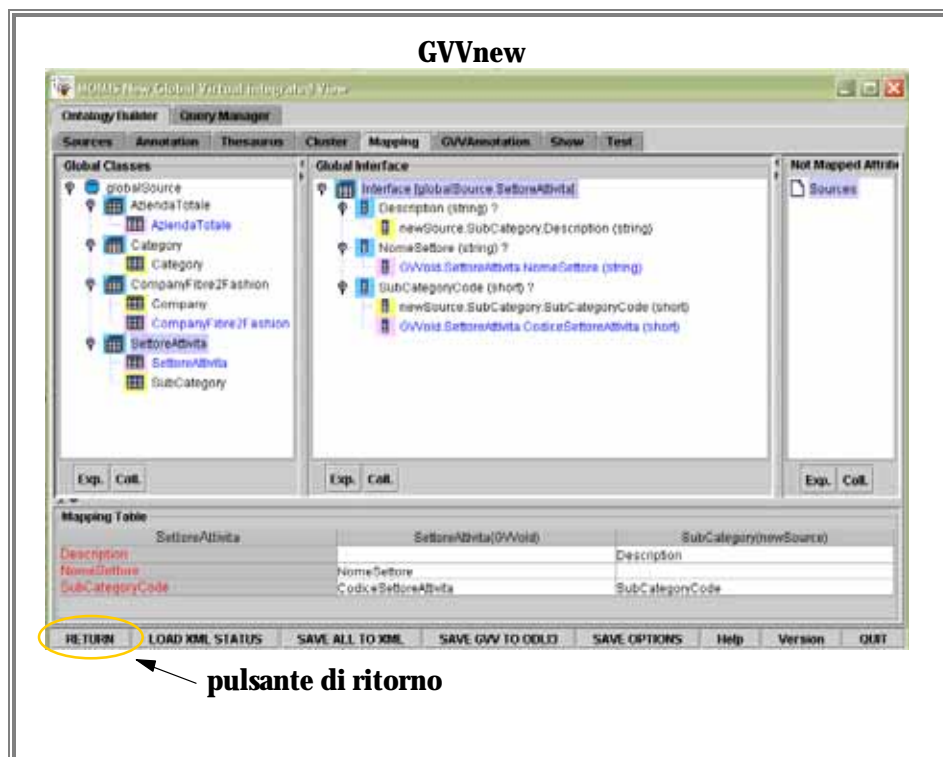


Figura 3: *GVVnew*: i diversi colori evidenziano le due tipologie di informazioni

La figura 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 *GVVoid* sono evidenziate in un colore diverso: blu; stessa cosa per le icone, quelle associate alla *GVVoid* 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.

## 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 *GVVoid*. 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 un 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 2.. L'evento ad esso associato chiama la funzione "**void UpdateGVVold()**", che genera una istanza 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 struttura:

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*

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 ()**". 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.

### 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[Freg02].

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* sono 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:

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

- ✓ *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

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

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

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

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.

### **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:

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

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, poiché 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 riferisce 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 *GIClassO* 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:

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

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.

### **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.

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

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.

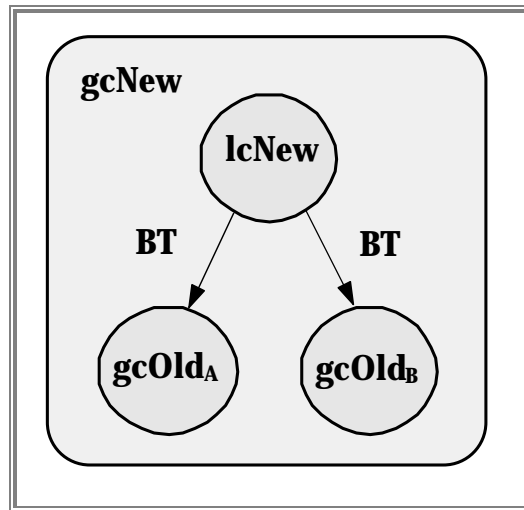


Figura 4: *lcNew* è una generalizzazione di due *gcOld*

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, 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*.

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 5.

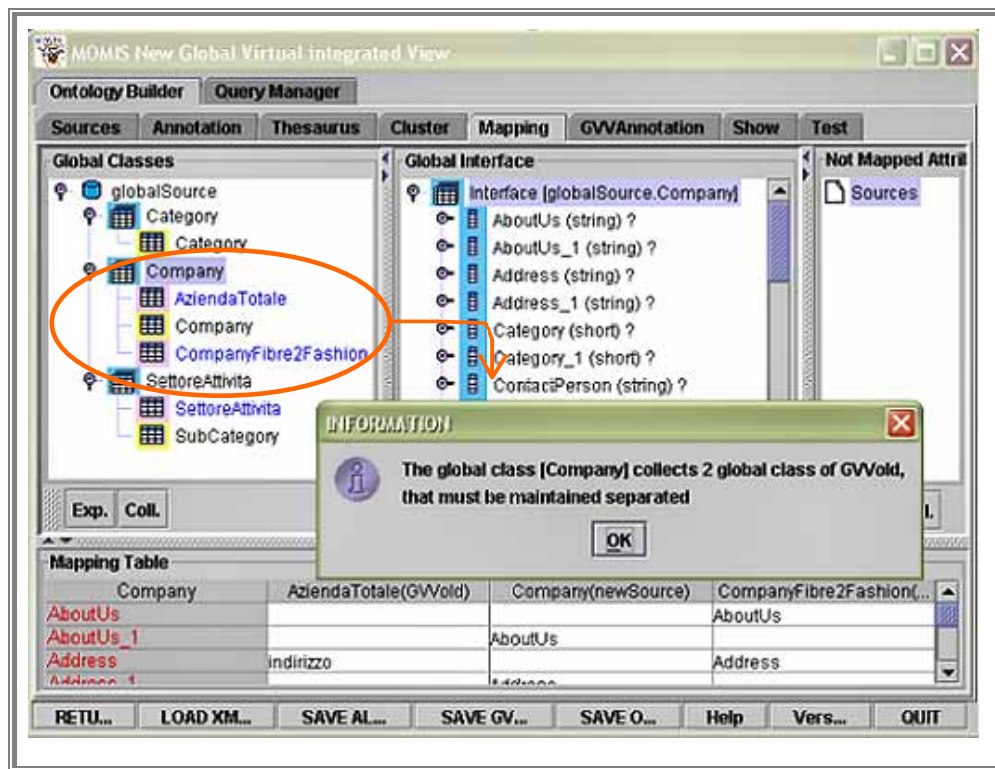


Figura 5: Pannello informativo di più *gcOld* unite in un'unica *gcNew*

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.

## 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 1 illustra la situazione più generale del mapping degli attributi nella *GVVold*.

	<i>lcOld<sub>1</sub></i>	...	<i>lcOld<sub>s</sub></i>	<i>lcNew<sub>1</sub></i>	...	<i>lcNew<sub>n</sub></i>
<i>gcOldAtt<sub>1</sub></i>	mapping di gcOld			nuovo mapping		
...						
<i>GcOldAtt<sub>m</sub></i>						
<i>gcNewAtt<sub>1</sub></i>	nessun mapping					
...						
<i>GcOldAtt<sub>p</sub></i>						

Tabella 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 GIClassN, GlobalInterface GIClassO)***”. I parametri di ingresso richiesti, per eseguire il confronto, sono le due classi ***GIClassN*** e ***GIClassO***, rispettivamente corrispondenti alle classi globali *gcNew* e *gcOld*.

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 ***GIClassO*** 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 6.

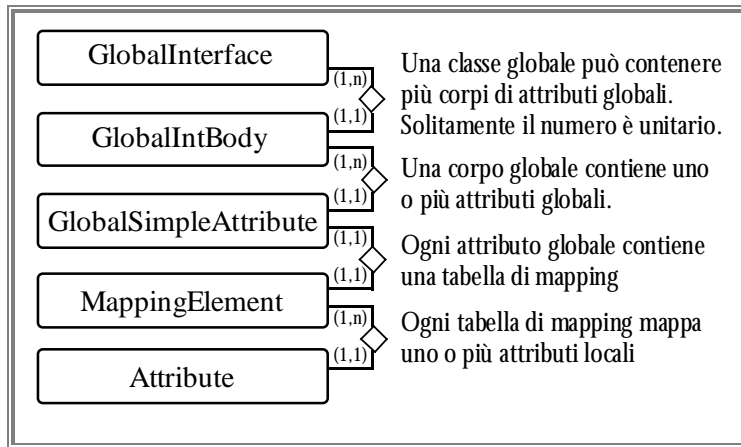


Figura 6: Struttura classi tra classe globale e attributo locale

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

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

Tuttavia, 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*;

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

- [ 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*;

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

- [ 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*;

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

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 precedentemente 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 terzo 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 iniziale 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 portare all'unione di questi 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.

## 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.

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 *IsorceOld*.

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 *IsorceOld*;
- [ 3 ] sostituzione del concetto globale con quelli locali relativi alle *IsorceOld*;
- [ 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<sup>B</sup>, 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***).

---

## Bibliografia

- [Benv03] D.Beneventano, S. Bergamaschi, F.Guerra and M. Vincini, "Synthesizing an intergraded ontology". In *IEEE Internet Computing Magazine* (2003)
- [Berg01] S. Bergamaschi, S. Castano, D. Beneventano, M. Vincini: "Semantic Integration of Heterogeneous Information Sources", *Special Issue on Intelligent Information Integration, Data & Knowledge Engineering, Vol. 36, Num. 1, Pages 215-249* (2001).

- [ManV2] <http://dbgroup.unimo.it/Momis/momis-iswc/MomisSewasieManualItaVer2.pdf>  
“SEWASIE - MOMIS Ontology Builder - Guida alla creazione di uno schema integrato”.
- [Brus05] S.Bruschi, “Dinamica delle ontologie: inserimento di una nuova sorgente nel sistema MOMIS”. *Tesi di laurea presso l'Università degli Studi di Modena e Reggio Emilia, facoltà di Ingegneria, corso di laurea in Ingegneria Informatica.* (2005)
- [Freg02] A.Fernani, “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)
- [Gard03] P. Gardenfors and C. J. Van Rijsbergen. Libro: “Belief Revision” *edizione Cambridge University Press* (dicembre 2003)