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

Analisi di Affinitá Strutturali fra Classi ODL$_{I^3}$ nel Sistema MOMIS

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Gianni Pio Grifa

Correlatore
Dott. Ing. Domenico Beneventano

Anno Accademico 1997 - ’98

Parole chiave:
Intelligent Information Integration
Database eterogenei
Mediatore
Integrazione Semantica
Affinitá Strutturali

Ai miei genitori

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi e il Dott.Ing. Domenico Beneventano per l'aiuto fornito alla realizzazione della presente tesi, l'Ing. Maurizio Vincini e l'Ing. Alberto Corni per la preziosa collaborazione e la costante disponibilità.
Inoltre, un ringraziamento particolare va ai miei genitori Michele e Maria e alla mia famiglia, per aver reso possibile tutto ció.

Indice

Elenco delle figure

Introduzione

A causa dell'esplosione dei dati disponibili (sia sulla rete Internet, che all'interno di un'azienda), si é avuto non solo un aumento della quantitá e del genere di informazioni reperibili, ma anche della velocitá con cui queste possono essere scambiate. A dispetto di ció peró per poter accedere a grosse quantitá di dati l'utente deve far fronte a problemi di varia natura: dalla selezione delle sorgenti interessate, alla analisi dei dati reperiti che spesso sono duplicati ed inconsistenti. Per cui unificare ed integrare dati consente di aumentare il valore dell'informazione ad essi collegata purché essi vengano prima selezionati, filtrati, collegati e sintetizzati [1], ed inoltre va incontro alle esigenze degli utenti informatici che chiedono delle applicazioni capaci di sintetizzare informazioni. Quindi si puó notare che ritrovare ed integrare informazioni provenienti da differenti sorgenti è attualmente divenuto un problema critico oltre che di grande interesse sia dal punto di vista teorico che applicativo.

Le principali difficoltá che si incontrano nell'integrazione di informazioni provenienti da sorgenti distribuite, sono relative a eterogeneitá strutturali e di implementazione (quali, ad esempio, differenze nelle piattaforme hardware, DBMS, modelli di dati e linguaggi di dati) e alla mancanza di una ontologia comune che porta a una eterogeneitá semantica [16]. Tale eterogeneitá semantica é dovuta principalmente a:

- *differenti termilnologie*, cosi' che ad informazioni uguali o simili possono essere assegnati, in differenti databases, nomi diversi;
- *differenti rappresentazioni strutturali* della stessa informazione in databases diversi, dovute all'uso di differenti modelli di dati;
- *differenti contesti* dell'uso dell'informazione, relativi ad aspetti geografici, organizzativi e funzionali.

Obiettivo della presente tesi è quello di progettare e realizzare un modulo software che individui concetti ed informazioni simili presenti nei differenti tipi

di sorgenti. Il progetto realizzato si inserisce nell'ambito del modulo MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [2, 3, 4] nato dalla cooperazione tra il gruppo di lavoro della Professoressa Sonia Bergamaschi e lo staff della Professoressa Silvana Castano del Dipartimento di Scienze dell'Informazione dell'Universitá di Milano, il cui scopo é quello di definire uno schema *"globale"*, che include tutti i concetti che le diverse fonti di Informazione vogliono condividere e che sará l'unico col quale l'utente dovrá interagire. L'integrazione di informazioni in MOMIS é ottenuto attraverso una fase di *estrazione ed analisi* seguita da una fase di *unificazione* [4]. Il progetto proposto va ad arricchire la fase di estrazione ed analisi delle informazioni attraverso la progettazione e realizzazione di un modulo software che, utilizzando tecniche di Analisi degli Schemi, individui concetti simili presenti in fonti di informazioni differenti.

La struttura della tesi é la seguente.

Il Capitolo 1 costituisce l'Introduzione alle problematiche che un sistema di Integrazione di Informazioni deve affrontare: in esso è riportata una classificazione di questi problemi, come pure una architettura di riferimento, recentemente proposta in ambito internazionale, che questo tipo di sistemi dovrebbero seguire.

Nel Capitolo 2 viene data una descrizione sia del sistema ODB-Tools sviluppato presso l'Universitá di Modena, sia della tecniche di Analisi degli Schemi, entrambi utilizzati nello sviluppo di MOMIS.

Il Capitolo 3 riporta la descrizione dell'intero progetto MOMIS.

Il Capitolo 4 descrive l'approccio utilizzato da MOMIS per l'Integrazione di Informazioni, descrivendo, con l'aiuto di un esempio di riferimento, tutti i passaggi che portano alla unificazione di un insieme di sorgenti eterogenee. Il Capitolo 5 descrive il modulo software **ESCA** (**E**valuation of **S**chema **C**lasses **A**ffinity) che è stato realizzato, riportandone sia l'architettura, sia gli algoritmi più significativi.

Capitolo 1

L'Integrazione Intelligente di Informazioni

La presenza di un numero sempre maggiore di fonti di informazione, all'interno di un azienda come sulla rete Internet, ha reso possibile oggi accedere ad un vastissimo insieme di dati, sparsi su macchine diverse come pure in luoghi diversi. Parallelamente quindi all'aumento delle probabilità di trovare un dato sulla rete informatica, in qualsivoglia fonte e formato, va costantemente aumentando la difficoltà di recuperare questo dato in tempi e modi accettabili, essendo tra di loro le fonti di informazione fortemente eterogenee, sia per quanto riguarda i tipi di dati (testuali, immagini . . .), sia per quanto riguarda il modo di descriverli, e quindi di *segnalarli* ai potenziali utenti. Contestualmente alla difficoltà di reperire un dato, pur nella sicurezza di ritrovarlo, si va inoltre delineando un altro tipo di problema, che paradossalmente nasce dall'abbondanza di informazioni, e che viene percepito dall'utente come *information overload* (sovraccarico di informazioni): il numero crescente di informazioni (e magari la loro replicazione) genera confusione, rendendo pressocché impossibile isolare efficientemente i dati necessari a prendere determinate decisioni.
In questo scenario, al momento fortemente studiato, e che coinvolge diverse aree di ricerca e di applicazione, si vanno oggi ad inserire i sistemi di supporto alle decisioni (DSS, Decision Support System), l'integrazione di basi di dati eterogenee, i *datawarehouse* (magazzino), fino ad arrivare ai sistemi distribuiti. I *decision maker* lavorano su fonti diverse (inclusi file system, basi di dati, librerie digitali, . . .) ma sono per lo più incapaci di ottenere e fondere le informazioni in un modo efficiente.
L'integrazione di basi di dati invece, e tutto ciò che va sotto il nome di datawarehouse, si occupa di materializzare presso l'utente finale delle viste, ovvero delle porzioni delle sorgenti, replicando però fisicamente i dati, ed affidandosi a complicati algoritmi di "mantenimento" di questi dati, per assicurare la loro consistenza

a fronte di cambiamenti nelle sorgenti originali. Con *Integration of Information* invece, come è descritto in [1], si rappresentano in letteratura tutti quei sistemi in grado di combinare tra di loro dati provenienti da intere sorgenti o parti selezionate di esse, senza fare uso della replicazione fisica delle informazioni, bensì basandosi sulle loro descrizioni. Quando inoltre questa integrazione utilizza tecniche di intelligenza artificiale, sfruttando le conoscenze acquisite, possiamo parlare di Intelligent Integration of Information (I^3), che si distingue quindi dalle altre forme di integrazione prefiggendosi non una semplice aggregazione di informazioni, bensì anche un aumento del loro valore, ottenendo nuove informazioni dai dati ricevuti. Con questi obiettivi si è quindi inserita, nell'ambito dell'integrazione, l'Intelligenza Artificiale (IA), che già aveva dato buoni risultati in domini applicativi più limitati. Naturalmente, è ovvio come sia pressocché impossibile pensare ad un sistema che vada bene per tutti i domini applicativi, e che magari integri un numero altissimo di sorgenti. Per questo motivo, per realizzare sistemi molto ampi, è stata proposta una partizione delle risorse e dei servizi che questi sistemi devono supportare, e che si articola su due dimensioni:

1. orizzontalmente, in tre livelli: livello utente, moduli intermedi che fanno uso di tecniche di IA, risorse di dati;
2. verticalmente: molti domini, con un numero limitato (e minore di 10) di sorgenti.

I domini nei vari livelli si scambieranno dati e informazioni tra di loro, ma non saranno strettamente collegati. Per esempio, in un sistema di recapito merci navale, le informazioni sulle navi saranno integrate da un modulo intermedio, quelle sul tempo nelle varie regioni da un altro modulo intermedio, ed un ulteriore modulo, ad un livello superiore, provvederà all'integrazione dei dati che gli verranno forniti dai *mediatori* (o *facilitatori*) sottostanti.
In questo quadro, dal 1992, si inserisce il progetto di ricerca I^3, fondato e sponsorizzato dall'ARPA, agenzia che fa capo al Dipartimento di Difesa americano [5]. I^3 si focalizza sul livello intermedio della partizione sopra descritta, livello che media tra gli utilizzatori e le sorgenti. All'interno di questo livello staranno diversi moduli tra i quali i più importanti sono:

- *facilitator* e *mediator* (le differenze tra i due sono flebili ed ancora ambigue in letteratura), che ricercano le fonti "interessanti" e combinano i dati da esse ricevuti;
- *query processor*, che riformulano le query aumentando le probabilità di successo di quest'ultime;
- *data miner*, che analizzano i dati per estrarre informazioni intensionali implicite.

Oltre alla architettura di riferimento, muovendosi questo progetto in un campo di ricerca particolarmente giovane e in evoluzione, è riportato in Appendice A il glossario, a cui rifarsi per termini che risultino ambigui o poco chiari.

1.1 Architettura di riferimento per sistemi I^3

L'architettura di riferimento presentata in questo paragrafo è stata tratta dal sito web [5], e rappresenta una sommaria categorizzazione dei principi e dei servizi che possono e devono essere usati nella realizzazione di un integratore *intelligente* di informazioni derivanti da fonti eterogenee. Alla base del progetto I^3 stanno infatti due ipotesi:

- la cosiddetta "autostrada delle informazioni" è oggigiorno incredibilmente vasta e, conseguentemente, sta per diventare una risorsa di informazioni utilizzabile poco efficientemente;
- le fonti di informazioni ed i sistemi informativi sono spesso semanticamente correlati tra di loro, ma non in una forma semplice né premeditata. Di conseguenza, il processo di integrazione di informazioni può risultare molto complesso.

In questo ambito, l'obiettivo del programma I^3 è di ridurre considerevolmente il tempo necessario per la realizzazione di un integratore di informazioni, raccogliendo e "strutturando" le soluzioni fino ad ora prevalenti nel campo della ricerca. Da sottolineare, prima di passare alla descrizione dell'architettura di riferimento, che questa architettura non implica alcuna soluzione implementativa, bensì vuole rappresentare alcuni dei servizi che deve includere un qualunque integratore di informazioni, e le interconnessioni tra questi servizi. Inoltre, è opportuno rimarcare che non sarà necessario, ed anzi è improbabile, che ciascun sistema che si prefigge di integrare informazioni (o servizi, o applicazioni) comprenda l' intero insieme di funzionalità che verranno descritte, bensì usufruirà esclusivamente delle funzionalità necessarie ad un determinato compito.

1.1.1 A cosa serve la tecnologia I^3 e quali problemi deve risolvere

Vi è un immenso spettro di applicazioni che si prestano naturalmente come campi applicativi per queste nuove tecnologie, tra le quali:

- pianificazione e supporto della logistica;

- sistemi informativi nel campo sanitario;
- sistemi informativi nel campo manifatturiero;
- sistemi bancari internazionali;
- ricerche di mercato.

Naturalmente, essendo questa riportata una architettura che pretende di essere il più generale possibile, ed essendo la casistica dei campi applicativi così vasta, sarà possibile identificare, al di là di un insieme di servizi di base, funzionalità più adatte ad una determinata applicazione e funzionalità specifiche di un altro ambiente. Ad esempio, un integratore che vuole interagire con sistemi di basi di dati "classici", come possono essere considerati i sistemi basati sui file, quelli relazionali, i DB ad oggetti, necessiterà di un pacchetto base di servizi molto differenti da un sistema cosiddetto "multimediale", che vuole integrare suoni, immagini ...
Così come possono essere differenti gli obiettivi di un sistema I^3, saranno differenti pure i problemi che si troverà ad affrontare. Tra questi, possono essere identificati:

- *la grande differenza tra le fonti di informazione*:
  - le fonti informative sono semanticamente differenti, e si possono individuare dei livelli di differenze semantiche [6];
  - le informazioni possono essere memorizzate utilizzando differenti formati, come possono essere file, DB relazionali, DB ad oggetti;
  - possono essere diversi gli schemi, i vocabolari usati, le ontologie su cui questi si basano, anche quando le fonti condividono significative relazioni semantiche;
  - può variare inoltre la natura stessa delle informazioni, includendo testi, immagini, audio, media digitali;
  - infine, può variare il modo in cui si accede a queste sorgenti: interfacce utente, linguaggi di interrogazione, protocolli e meccanismi di transazione;
- *la semantica complessa ed a volte nascosta delle fonti*: molto spesso, la chiave per l'uso delle informazioni di vecchi sistemi sono i programmi applicativi su di essi sviluppati, senza i quali può essere molto difficile dedurre la semantica che si voleva esprimere, specialmente se si ha a che fare con sistemi molto vasti e quasi impossibili da interpretare se visti solo dall'esterno;

- *l'esigenza di creare applicazioni in grado di interfacciarsi con porzioni diverse delle fonti di informazione*: molto spesso, non è sempre possibile avere a disposizione l'intera sorgente di informazione, bensì una sua parte selezionata che può variare nel tempo;

- *il grande numero di fonti da integrare*: con il moltiplicarsi delle informazioni, il numero stesso delle fonti da integrare per una applicazione, ad esempio nel campo sanitario, è aumentato considerevolmente, e decine di fonti devono essere accedute in modo coordinato;

- *il bisogno di realizzare moduli I^3 riusabili*: benché questo possa essere considerato uno dei compiti più difficili nella realizzazione di un integratore, è importante realizzare non un sistema ad-hoc, bensì un'applicazione i cui moduli possano facilmente essere riutilizzati in altre applicazioni, secondo i moderni principi di riusabilità del software. In questo caso, l'abilità di costruire valide funzioni di libreria può considerevolmente diminuire i tempi e le difficoltà di realizzazione di un sistema informativo che si basa su più fonti differenti.

Passiamo ora ad analizzare l'architettura vera e propria di un sistema I^3, riportata in Figura 1.1. L'architettura di riferimento dà grande rilevanza ai Servizi di Coordinamento. Questi servizi giocano infatti due ruoli: come prima cosa, possono localizzare altri servizi I^3 e fonti di informazioni che possono essere utilizzati per costruire il sistema stesso; secondariamente, sono responsabili di individuare ed invocare a run-time gli altri servizi necessari a dare risposta ad una specifica richiesta di dati.
Sono comunque in totale cinque le famiglie di servizi che possono essere identificati in questa architettura: importanti sono i due assi della figura, orizzontale e verticale, che sottolineano i differenti compiti dei servizi I^3.
Se percorriamo l'asse verticale, si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i servizi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti, che sono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passati ai servizi di Coordinamento che ne avevano fatto richiesta. L'asse orizzontale mette invece in risalto il rapporto tra i servizi di Coordinamento e quelli di Amministrazione, ai quali spetta infatti il compito di mantenere informazioni sulle capacità delle varie sorgenti (che tipo di dati possono fornire ed in quale modo devono essere interrogate). Funzionalità di supporto, che verranno descritte successivamente, sono invece fornite dai Servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti.
Analizziamone in dettaglio funzionalità e problematiche affrontate.

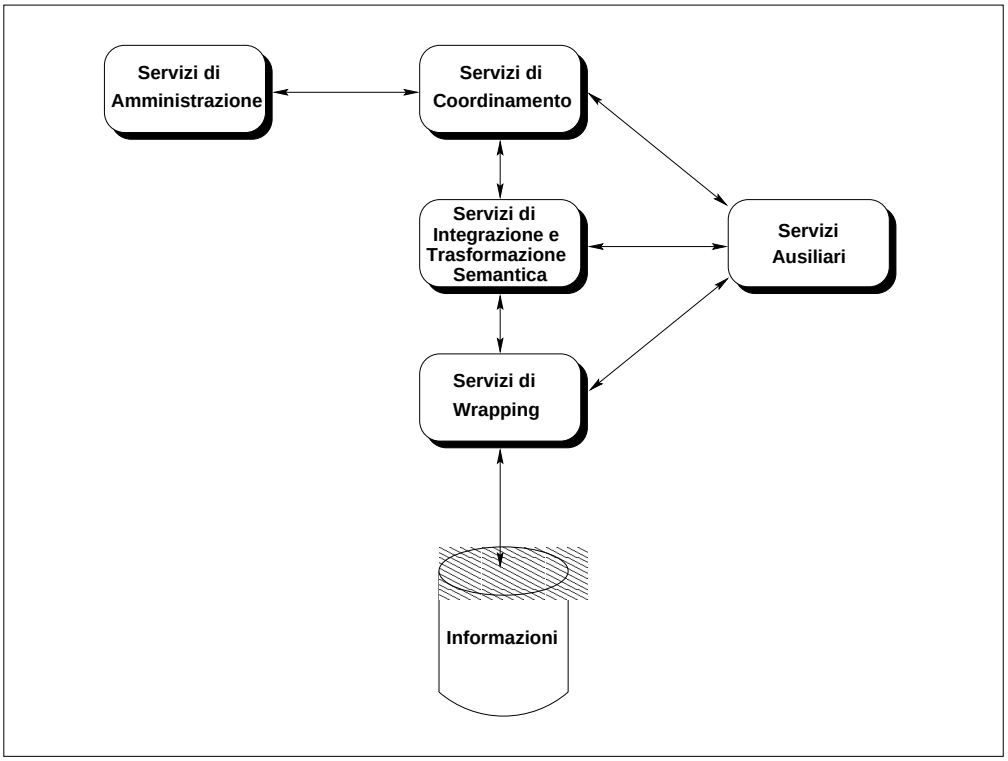


Figura 1.1: Diagramma dei servizi I^3

1.1.2 Servizi di Coordinamento

I servizi di Coordinamento sono quei servizi di alto livello che permettono l'individuazione delle sorgenti di dati *interessanti*, ovvero che probabilmente possono dare risposta ad una determinata richiesta dell'utente. A seconda delle possibilità dell'integratore che si vuole realizzare, vanno dalla selezione dinamica delle sorgenti (o brokering, per Integratori Intelligenti) al semplice *Matchmaking*, in cui il mappaggio tra informazioni integrate e locali è realizzato manualmente ed una volta per tutte. Vediamo alcuni esempi.

1. **Facilitation e Brokering Services**: l'utente manda una richiesta al sistema e questo usa un deposito di metadati per ritrovare il modulo che può trattare la richiesta direttamente. I moduli interessati da questa richiesta potranno essere uno solo alla volta (nel qual caso si parla di Brokering) o più di uno (e in questo secondo caso si tratta di facilitatori e mediatori, attraverso i quali a partire da una richiesta ne viene generata più di una da inviare singolarmente a differenti moduli che gestiscono sorgenti distinte, e reintegrando poi le risposte in modo da presentarle all'utente come se fossero

state ricavate da un'unica fonte). In questo ultimo caso, in cui una query può essere decomposta in un insieme di sottoquery, si farà uso di servizi di Query Decomposition e di tecniche di Inferenza (mutuate dall'Intelligenza Artificiale) per una determinazione dinamica delle sorgenti da interrogare, a seconda delle condizioni poste nell'interrogazione.
I vantaggi che questi servizi di Coordinamento portano, consistono nel fatto che non è richiesta all'utente del sistema una conoscenza del contenuto delle diverse sorgenti, dandogli l'illusione di interagire con un sistema omogeneo che gestisce direttamente la sua richiesta. E' quindi esonerato dal conoscere i domini con i quali i vari moduli I^3 hanno a che fare, ottenendone una considerabile diminuzione di complessità di interazione col sistema.

2. **Matchmaking**: il sistema è configurato manualmente da un operatore all'inizio, e da questo punto in poi tutte le richieste saranno trattate allo stesso modo. Sono definiti gli anelli di collegamento tra tutti i moduli del sistema, e nessuna ottimizzazione è fatta a tempo di esecuzione.

1.1.3 Servizi di Amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti *utili*, per determinare le loro capacità, e per creare ed interpretare i TEMPLATE. I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task. Sono quindi utilizzati dai sistemi meno "intelligenti", e consentono all'operatore di predefinire le azioni da eseguire a seguito di una determinata richiesta, limitando al minimo le possibilità di decisione del sistema.
In alternativa a questi metodi dei Template, sono utilizzate le **Yellow Pages**: servizi di directory che mantengono le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata). Consultando queste Yellow Pages, il mediatore sarà in grado di spedire alla giusta sorgente la richiesta di informazioni, ed eventualmente di rimpiazzare questa sorgente con una equivalente nel caso non fosse disponibile. Fanno parte di questa categoria di servizi il Browsing: permette all'utente di "navigare" attraverso le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste. Il servizio si basa sulla premessa che queste descrizioni siano fornite esplicitamente tramite un linguaggio dichiarativo leggibile e comprensibile dall'utente. Potrebbe fornirsi a sua volta dei servizi Trasformazione del Vocabolario e dell'Ontologia, come pure di Integrazione Semantica. Da citare sono pure i servizi di Iterative Query Formulation: aiutano l'utente a rilassare o meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

1.1.4 Servizi di Integrazione e Trasformazione Semantica

Questi servizi supportano le manipolazioni semantiche necessarie per l'integrazione e la trasformazione delle informazioni. Il tipico input per questi servizi saranno una o più sorgenti di dati, e l'output sarà la "vista" integrata o trasformata di queste informazioni. Tra questi servizi si distinguono quelli relativi alla trasformazione degli schemi (ovvero di tutto ciò che va sotto il nome di *metadati*) e quelli relativi alla trasformazione dei dati stessi. Sono spesso indicati come servizi di Mediazione, essendo tipici dei moduli mediatori.

1. Servizi di **integrazione degli schemi**. Supportano la trasformazione e l'integrazione degli schemi e delle conoscenze derivanti da fonti di dati eterogenee. Fanno parte di essi i servizi di trasformazione dei vocaboli e dell'ontologia, usati per arrivare alla definizione di un'ontologia unica che combini gli aspetti comuni alle singole ontologie usate nelle diverse fonti. Queste operazioni sono molto utili quando devono essere scambiate informazioni derivanti da ambienti differenti, dove molto probabilmente non si condivideva un'unica ontologia. Fondamentale, per creare questo insieme di vocaboli condivisi, è la fase di individuazione dei concetti presenti in diverse fonti, e la riconciliazione delle diversità presenti sia nelle strutture, sia nei significati dei dati.

2. Servizi di **integrazione delle informazioni**. Provvedono alla traduzione dei termini da un contesto all'altro, ovvero dall'ontologia di partenza a quella di destinazione. Possono inoltre occuparsi di uniformare la "granularità" dei dati (come possono essere le discrepanze nelle unità di misura, o le discrepanze temporali).

3. Servizi di **supporto al processo di integrazione**. Sono utilizzati nel momento in cui una query è scomposta in molte subquery, da inviare a fonti differenti, ed i loro risultati devono essere integrati. Comprendono inoltre tecniche di *caching*, per supportare la materializzazione delle viste (problematica molto comune nei sistemi che vanno sotto il nome di datawarehouse).

1.1.5 Servizi di Wrapping

Sono utilizzati per fare sì che le fonti di informazioni aderiscano ad uno standard, che può essere interno o proveniente dal mondo esterno con cui il sistema vuole interfacciarsi. Si comportano come traduttori dai sistemi locali ai servizi di alto livello dell'integratore. In particolare, sono due gli obiettivi che si prefiggono:

1. permettere ai servizi di coordinamento e di mediazione di manipolare in modo uniforme il maggior numero di sorgenti locali, anche se queste non erano state esplicitamente pensate come facenti parte del sistema di integrazione;

2. essere il più riusabili possibile. Per fare ciò, dovrebbero fornire interfacce che seguano gli standard più diffusi (e tra questi, si potrebbe citare il linguaggio SQL come linguaggio di interrogazione di basi di dati, e CORBA come protocollo di scambio di oggetti). Questo permetterebbe alle sorgenti estratte da questi wrapper ”universali” di essere accedute dal numero maggiore possibile di moduli mediatori.

In pratica, compito di un wrapper è modificare l'interfaccia, i dati ed il comportamento di una sorgente, per facilitarne la comunicazione con il mondo esterno. Il vero obiettivo è quindi standardizzare il processo di wrapping delle sorgenti, permettendo la creazione di una libreria di fonti accessibili; inoltre, il processo stesso di realizzazione di un wrapper dovrebbe essere standardizzato, in modo da poter essere riutilizzato per altre fonti.

1.1.6 Servizi Ausiliari

Aumentano le funzionalità degli altri servizi descritti precedentemente: sono prevalentemente utilizzati dai moduli che agiscono direttamente sulle informazioni. Vanno dai semplici servizi di monitoraggio del sistema (un utente vuole avere un segnale nel momento in cui avviene un determinato evento in un database, e conseguenti azioni devono essere attuate), ai servizi di propagazione degli aggiornamenti e di ottimizzazione.

1.2 Il mediatore

L'obiettivo del progetto di ricerca di cui la tesi fa parte è la progettazione e realizzazione di un **mediatore**, ovvero del modulo intermedio dell'architettura precedentemente descritta, che si pone tra l'utente e le sorgenti di informazioni. Secondo la definizione proposta da Wiederhold in [7] ”un mediatore è 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.”
Compiti di un mediatore sono allora:

- assicurare un servizio stabile, anche quando cambiano le risorse;

- amministrare e risolvere le eterogeneità delle diverse fonti;

- integrare le informazioni ricavate da più risorse;
- presentare all'utente le informazioni attraverso un modello scelto dall'utente stesso.

La prima ipotesi che è stata fatta, per restringere il campo applicativo del sistema da progettare (e di conseguenza per restringere il campo dei problemi a cui dare risposta) è di avere a che fare, per il momento, esclusivamente con sorgenti di dati testuali strutturati, come possono essere basi di dati relazionali, ad oggetti e file di testo. L'approccio architetturale scelto è stato quello *classico*, che consta principalmente di 3 livelli:

1. 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;
2. mediatore: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio di interrogazione) comune a tutte le fonti;
3. wrapper: ogni wrapper gestisce una singola sorgente, convertendo le richieste del mediatore in una forma comprensibile dalla sorgente, e le informazioni da essa estratte nel modello usato dal mediatore.

Facendo riferimento ai servizi descritti nelle sezioni precedenti, l'architettura del mediatore che si è progettato è riportata in Figura 1.2. In particolare, in questa tesi, sono state esaminati

- **servizi di Integrazione e Trasformazione Semantica**: saranno forniti dal mediatore servizi che facilitino l'integrazione sia degli schemi che delle informazioni.

Parallelamente a questa impostazione architetturale inoltre, il nostro progetto si vuole distaccare dall'approccio *strutturale*, cioé sintattico, tuttora dominante tra i sistemi presenti sul mercato. Questo approccio, è caratterizzato dal fatto di usare un self-describing model per rappresentare gli oggetti da integrare, limitando l'uso delle informazioni semantiche a delle regole predefinite dall'operatore. In pratica, il sistema non conosce a priori la semantica di un oggetto che va a recuperare da una sorgente (e dunque di questa non possiede alcuno schema descrittivo) bensì è l'oggetto stesso che, attraverso delle etichette, si autodescrive, specificando tutte le volte, per ogni suo singolo campo, il significato che ad esso è associato. Questo approccio porta quindi ad un insieme di vantaggi, tra i quali possiamo identificare:

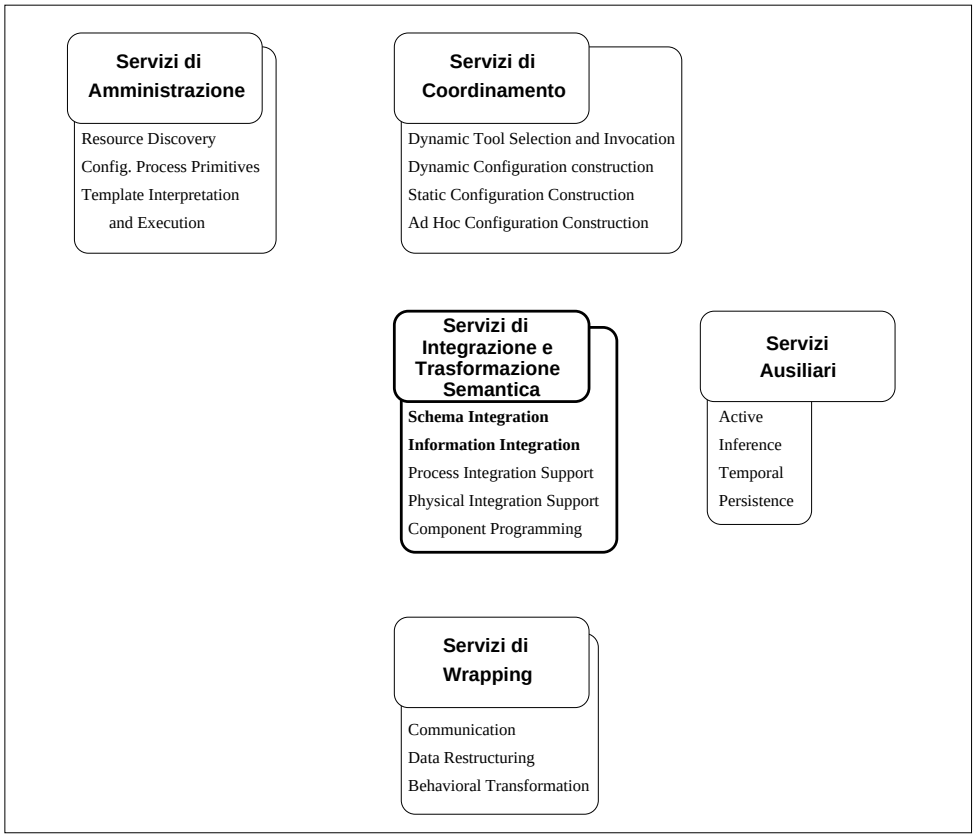


Figura 1.2: Servizi I^3 presenti nel mediatore

- la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo: il mediatore non si basa infatti su una descrizione predefinita degli schemi delle sorgenti, bensì sulla descrizione che ogni singolo oggetto fa di sè. Oggetti simili provenienti dalla stessa sorgente possono quindi avere strutture differenti, cosa invece non ammessa in un ambiente tradizionale object-oriented;

- per trattare in modo omogeneo dati che descrivono lo stesso concetto, o che hanno concetti in comune, ci si basa sulla definizione manuale di rule, che permettono di identificare i termini (e dunque i concetti) che devono essere condivisi da più oggetti.

Altri progetti, e tra questi il nostro, seguono invece un approccio definito *semantico*, che è caratterizzato da questi punti:

- il mediatore deve conoscere, per ogni sorgente, lo schema concettuale (metadati);
- informazioni semantiche sono codificate in questi schemi;
- deve essere disponibile un modello comune per descrivere le informazioni da condividere (e dunque per descrivere anche i metadati);
- deve essere possibile una integrazione (parziale o totale) delle sorgenti di dati.

In questo modo, sfruttando in modo appropriato le informazioni semantiche che necessariamente ogni schema sottintende, il mediatore può individuare concetti comuni a più sorgenti e relazioni che li legano.

1.2.1 Problematiche da affrontare

Pur avendo a disposizione gli schemi concettuali delle varie sorgenti, non è certamente un compito banale individuare i concetti comuni ad esse, le relazioni che possono legarli, né tantomeno è banale realizzare una loro coerente integrazione. Mettendo da parte per un attimo le differenze dei sistemi fisici (alle quali dovrebbero pensare i moduli wrapper) i problemi che si è dovuto risolvere, o con i quali occorre giungere a compromessi, sono (a livello di mediazione, ovvero di integrazione delle informazioni) essenzialmente di due tipi:

1. problemi ontologici;
2. problemi semantici.

Vediamoli più in dettaglio.

1.2.2 Problemi ontologici

Come riportato nel glossario A, per ontologia si intende, in questo ambito, "l'insieme dei termini e delle relazioni usate in un dominio, che denotano concetti ed oggetti". Con ontologia quindi ci si riferisce a quell'insieme di termini che, in un particolare dominio applicativo, denotano in modo univoco una particolare conoscenza e fra i quali non esiste ambiguità poiché sono condivisi dall'intera comunità di utenti del dominio applicativo stesso. Non è certamente l'obiettivo né di questo paragrafo, né della tesi in generale, dare una descrizione esaustiva di cosa si intenda per ontologia e dei problemi che essa comporta (ancorché ristretti al campo dell'integrazione delle informazioni), ma mi limito a riportare una semplice classificazione delle ontologie (mutuata da Guarino [8, 9]), per inquadrare

l'ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate) sono essenzialmente quattro:

1. *top-level ontology*: descrivono concetti molto generali come spazio, tempo, evento, azione…, che sono quindi indipendenti da un particolare problema o dominio: si considera ragionevole, almeno in teoria, che anche comunità separate di utenti condividano la stessa top-level ontology;

2. *domain e task ontology*: descrivono, rispettivamente, il vocabolario relativo a un generico dominio (come può essere un dominio medico, o automobilistico) o a un generico obiettivo (come la diagnostica, o le vendite), dando una specializzazione dei termini introdotti nelle top-level ontology;

3. *application ontology*: descrivono concetti che dipendono sia da un particolare dominio che da un particolare obiettivo.

Come ipotesi semplificativa di questo progetto, si è considerato di muoversi all'interno delle domain ontology, ipotizzando quindi che tutte le fonti informative condividano almeno i concetti fondamentali (ed i termini con cui identificarli).

1.2.3 Problemi semantici

Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare, e quindi un insieme di concetti comuni, niente ci dice che i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti, né tantomeno le stesse strutture dati. Poiché infatti le diverse sorgenti sono state progettate e modellate da persone differenti, è molto improbabile che queste persone condividano la stessa "concettualizzazione" del mondo esterno, ovvero non esiste nella realtà una semantica univoca a cui chiunque possa riferirsi.
Se la persona P1 disegna una fonte di informazioni (per esempio DB1) e un'altra persona P2 disegna la stessa fonte DB2, le due basi di dati avranno sicuramente differenze semantiche: per esempio, le coppie sposate possono essere rappresentate in DB1 usando degli oggetti della classe COPPIE, con attributi MARITO e MOGLIE, mentre in DB2 potrebbe esserci una classe PERSONA con un attributo SPOSA.
Come riportato in [6] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma non è l'unica. Le differenze nei sistemi di DBMS possono portare all'uso di differenti modelli per la rappresentazione della porzione di mondo in questione: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate

attraverso un modello relazionale, o ad oggetti.
L'obiettivo dell'integratore, che è fornire un accesso integrato ad un insieme di sorgenti, si traduce allora nel non facile compito di identificare i concetti comuni all'interno di queste sorgenti e risolvere le differenze semantiche che possono essere presenti tra di loro. Possiamo classificare queste contraddizioni semantiche in tre gruppi principali:

1. **eterogeneità tra le classi di oggetti**: benché due classi in due differenti sorgenti rappresentino lo stesso concetto nello stesso contesto, possono usare nomi diversi per gli stessi attributi, per i metodi, oppure avere gli stessi attributi con domini di valori diversi o ancora (dove questo è permesso) avere regole differenti su questi valori;

2. **eterogeneità tra le strutture delle classi**: comprendono le differenze nei criteri di specializzazione, nelle strutture per realizzare una aggregazione, ed anche le *discrepanze schematiche*, quando cioé valori di attributi sono invece parte dei metadati in un altro schema (come può essere l'attributo SESSO in uno schema, presente invece nell'altro implicitamente attraverso la divisione della classe PERSONE in MASCHI e FEMMINE);

3. **eterogeneità nelle istanze delle classi**: ad esempio, l'uso di diverse unità di misura per i domini di un attributo, o la presenza/assenza di valori nulli.

Parallelamente a tutto questo, è però il caso di sottolineare la possibilità di sfruttare adeguatamente queste differenze semantiche per arricchire il nostro sistema: analizzando a fondo queste differenze, e le loro motivazioni, si può arrivare al cosiddetto **arricchimento semantico**, ovvero all'aggiungere esplicitamente ai dati tutte quelle informazioni che erano originariamente presenti solo come metadati negli schemi, dunque in un formato non interrogabile.

1.3 Soluzione proposta

Sulla base di tutte le problematiche precedentemente esposte, si inserisce il progetto di un sistema intelligente di Integrazione delle Informazioni denominato MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources). Contributo innovativo del progetto, rispetto ad altri similari, è sicuramente la fase di analisi ed integrazione degli schemi sorgenti, realizzata in modo semi-automatico, e che rappresenta l'argomento di questa tesi.
In particolare é stato studiato un procedimento che, usando le relazioni terminologiche definite precedentemente, valuta automaticamente il livello di affinitá tra

classi appartenenti o no alla stessa sorgente.
La modifica introdotta rende piú completo il processo di analisi e integrazione, in quanto fornisce uno strumento basilare per il processo di generazione dello schema globale.
Per la realizzazione di tale componente sono stati utilizzati delle tecniche di Analisi degli Schemi eterogenei, che verranno descritte nel Capitolo 2.
Per quanto riguarda il progetto é stato realizzato un prototipo software, denominato **ESCA** (**E**valuation of **S**chema **C**lasses **A**ffinity), che realizza tutti i passaggi che portano alla valutazione della similaritá di pezzi di informazione rappresentati in differenti modi oppure che hanno differenti significati o scopi all'interno delle diverse sorgenti [17].

Capitolo 2

Gli strumenti utilizzati

L'obiettivo della presente tesi é arrivare alla definizione di un modulo che, all'interno del mediatore MOMIS, analizzi i coefficienti di affinitá tra classi appartenenti a sorgenti eterogenee. A tal fine sono state utilizzate tecniche di Integrazione di Schemi, sviluppate presso l'Universitá di Milano e che verranno descritte in seguito, le quali sfruttano un predefinito dizionario semantico.

In oltre si é pensato di fare una breve descrizione di moduli che fanno uso di tecniche di Intelligenza Artificiale basate sulla logica descrittiva **OCDL**, che vanno sotto il nome di ODB-Tools, sviluppati presso il Dipartimento di Ingegneria dell'Universitá di Modena ed utilizzati durante lo sviluppo di MOMIS, con lo scopo di dare una descrizione piú completa di tale progetto.

2.1 ODB-Tools

ODB-Tools è un sistema per l'acquisizione e la verifica di consistenza di schemi di basi di dati e per l'ottimizzazione semantica di interrogazioni nelle basi di dati orientate agli oggetti (OODB). È stato sviluppato presso il Dipartimento di Scienze dell'Ingegneria dell'Università di Modena [10, 11]. L'ambiente teorico su cui è basato ODB-Tools include due elementi fondamentali:

1. **OCDL**(**O**bject **C**onstraint **D**escription **L**ogics), proposto come formalismo comune per esprimere descrizioni di classi, vincoli di integrità, ed interrogazioni e dotato di tecniche di inferenza basate sul calcolo della sussunzione introdotte per le *Logiche Descrittive* nell'ambito dell'Intelligenza Artificiale;

2. **espansione semantica** di un tipo, realizzata attraverso l'algoritmo di sussunzione.

2.1.1 OCDL: un Formalismo per Oggetti Complessi e Vincoli di Integrità

Riportiamo in questa sezione la sintassi del linguaggio **OCDL**, nonché una breve descrizione dei concetti di *espansione semantica* e *relazione di sussunzione*.
Nel corso della descrizione, si farà inoltre riferimento ad un esempio esplicativo, di un dominio `Magazzino`, di cui riportiamo la descrizione in sintassi ODMG-93 nella tabella 2.1, e le regole di integrità su di esso definite in tabella 2.2.

L'esempio considerato è relativo alla struttura organizzativa di una società: i materiali (`Material`) sono descritti da un nome, un rischio e da un insieme di caratteristiche; gli `SMaterial` sono un sottoinsieme dei `Material`. I `DMaterial` (materiali "pericolosi") sono un sottoinsieme dei `Material`, caratterizzati da un rischio compreso tra 15 e 100. I manager hanno un nome, un salario compreso tra 40K e 100K dollari e un livello compreso tra 1 e 15. I `TManager` sono manager che hanno un livello compreso tra 8 e 12. I magazzini (`Storage`) sono descritti da una categoria, sono diretti (`managed_by`) da un manager e contengono (`stock`) un insieme di articoli (`item`) che sono dei materiali, per ciascuno dei quali è indicata la quantità presente; è inoltre riportato il rischio massimo (`maxrisk`) ammissibile per i materiali conservati. Gli `SStorage` sono un sottoinsieme dei magazzini. Infine vi sono i magazzini "pericolosi" (`DStorage`) che sono un sottoinsieme degli `Storage` caratterizzati dallo stoccaggio di soli materiali "pericolosi" (`DMaterial`).

Schema e istanza del database

Sia $\mathbf{D}$ l'insieme infinito numerabile dei valori atomici (che saranno indicati con $d_1, d_2, \ldots$), e.g., l'unione dell'insieme degli interi, delle stringhe e dei booleani. Sia $\mathbf{B}$ l'insieme di designatori di tipi atomici, con $\mathbf{B}$ = {`integer`, `string`, `boolean`, `real`, i_1–j_1, i_2–j_2, $\ldots$, $d_1, d_2, \ldots$}, dove i d_k indicano tutti gli elementi di `integer` $\cup$ `string` $\cup$ `boolean` e dove gli i_k–j_k indicano tutti i possibili intervalli di interi (i_k può essere $-\infty$ per denotare il minimo elemento di `integer` e j_k può essere $+\infty$ per denotare il massimo elemento di `integer`).

Sia $\mathbf{A}$ un insieme numerabile di *attributi* (denotati da $a_1, a_2, \ldots$) e $\mathcal{O}$ un insieme numerabile di *identificatori di oggetti* (denotati da $o, o', \ldots$) disgiunti da $\mathbf{D}$. Si definisce l'insieme $\mathcal{V}(\mathcal{O})$ dei *valori su* $\mathcal{O}$ (denotati da v, v') come segue (assumendo $p \geq 0$ e $a_i \neq a_j$ per $i \neq j$):

$$v \rightarrow d \mid o \mid \{v_1, \ldots, v_p\} \mid [a_1 : v_1, \ldots, a_p : v_p]$$

Gli identificatori di oggetti sono associati a valori tramite una *funzione totale* δ da $\mathcal{O}$ a $\mathcal{V}(\mathcal{O})$; in genere si dice che il valore $\delta(o)$ è lo *stato* dell'oggetto identificato dall'oid o.
Sia $\mathbf{N}$ l'insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che $\mathbf{A}$, $\mathbf{B}$,

Classi dello Schema

```
interface Material
{
   attribute string  name;
   attribute integer risk;
   attribute string  code;
   attribute set <string> feature;
};
interface DMaterial: Material
{
   attribute range  {15, 100} risk;
};
interface SMaterial: Material{ };
interface Storage
{
   attribute string category;
   attribute Manager managed_by;
   attribute set < struct {
         attribute Material item;
         attribute range {10, 300} qty; } > stock;
   attribute integer maxrisk;
};
interface Manager
{
   attribute string  name;
   attribute range  {40K, 100K} salary;
   attribute range  {1, 15} level;
};
interface TManager: Manager
{
   attribute range  {8, 12} level;
};
interface SStorage: Storage{ };
interface DStorage: Storage
{
   attribute set < struct {
         attribute DMaterial item; } > stock;
};
```

Tabella 2.1: Schema del dominio `Magazzino` in sintassi ODMG–93

rule R1 **for all** X **in** `Material` (X.`risk` $\geq$ 10)
then X **in** `SMaterial`
rule R2 **for all** X **in** `Storage` (
for all X1 **in** X.`stock` (X1.`item` **in** `SMaterial`))
then X **in** `SStorage`
rule R3 **for all** X **in** `Storage` (X.`managed_by.level` $\geq$ 6 **and**
X.`managed_by.level` $\leq$ 12)
then X.`category` = "A2"
rule R4 **for all** X **in** `Storage` (X.`category` = "A2" **and**
exists X1 **in** X.`stock` (X1.`item.risk` $\geq$ 10))
then X.`managed_by` **in** `SMaterial`)

Tabella 2.2: Regole di integrità sul dominio `Magazzino`

e $\mathbf{N}$ siano a due a due disgiunti. $\mathbf{N}$ è partizionato in tre insiemi $\mathbf{C}$, $\mathbf{V}$ e $\mathbf{T}$, dove $\mathbf{C}$ consiste di nomi per *tipi-classe base* ($C, C' \ldots$), $\mathbf{V}$ consiste di nomi per *tipi-classe virtuali* ($V, V' \ldots$), e $\mathbf{T}$ consiste di nomi per *tipi-valori* ($t, t', \ldots$). Un *path* p è una sequenza di elementi $p = e_1 \,.\, e_2 \,.\, \ldots \,.\, e_n$, con $e_i \in \mathbf{A} \cup \{\triangle, \forall, \exists\}$. Con ϵ si indica il path vuoto.
$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$[1] indica l'insieme di tutte le *descrizioni di tipo finite* (S, S', ...), dette brevemente *tipi*, su di un dato $\mathbf{A}, \mathbf{B}, \mathbf{N}$, ottenuto in accordo con la seguente regola sintattica:

$$S \;\rightarrow\; \top \mid B \mid N \mid [a_1 : S_1, \ldots, a_k : S_k] \mid \forall\{S\} \mid \exists\{S\} \mid \triangle S \mid S \sqcap S' \mid (p : S)$$

$\top$ denota il *tipo universale* e rappresenta tutti i valori; $[\,]$ denota il costruttore di tupla. $\forall\{S\}$ corrisponde al comune costruttore di insieme e rappresenta un insieme i cui elementi sono *tutti* dello stesso tipo S. Invece, il costruttore $\exists\{S\}$ denota un insieme in cui *almeno* un elemento è di tipo S. Il costrutto $\sqcap$ indica la *congiunzione*, mentre $\triangle$ è il costruttore di oggetto. Il tipo $(p{:}\, S)$ è detto *tipo path* e rappresenta una notazione abbreviata per i tipi ottenuti con gli altri costruttori.

Dato un dato sistema di tipi $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, uno *schema* σ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ è una funzione totale da $\mathbf{N}$ a $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, che associa ai nomi di tipi la loro descrizione. Diremo che un nome di tipo N *eredita* da un altro nome di tipo N', denotato con $N \prec_\sigma N'$, se $\sigma(N) = N' \sqcap S$. Si richiede che la relazione di ereditarietà sia priva di cicli, i.e., la chiusura transitiva di $\prec_\sigma$, denotata $\prec$, sia un ordine parziale stretto.

Dato un dato sistema di tipi $\mathbf{S}$, una *regole di integrità* R è espressa nella forma $R = S^a \rightarrow S^c$, dove S^a e S^c rappresentano rispettivamente l'antecedente e il conseguente della regola R, con $S^a, S^c \in \mathbf{S}$. Una regola R esprime il seguente vincolo: per tutti gli oggetti v, se v è di tipo S^a allora v deve essere di tipo S^c. Con $\mathbf{R}$ si denota un insieme finito di regole.

[1] In seguito, scriveremo $\mathbf{S}$ in luogo di $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ quando i componenti sono ovvi dal contesto.

$$
\begin{array}{l}
\mathbf{C} = \{\texttt{Material}, \texttt{SMaterial}, \texttt{DMaterial}, \texttt{Storage}, \texttt{SStorage}, \texttt{DStorage}, \texttt{Manager}, \\
\quad \texttt{TManager}\}, \\
\mathbf{V} = \emptyset, \\
\mathbf{T} = \emptyset
\end{array}
$$

$$
\sigma \left\{ \begin{array}{lcl}
\sigma(\texttt{Material}) & = & \triangle[\texttt{name}\colon \texttt{string}, \texttt{risk}\colon \texttt{integer}, \texttt{code}\colon \texttt{string}, \texttt{feature}\colon \{\texttt{string}\}] \\
\sigma(\texttt{SMaterial}) & = & \texttt{Material} \\
\sigma(\texttt{DMaterial}) & = & \texttt{Material} \sqcap \triangle[\texttt{risk}\colon 15 \div 100] \\
\sigma(\texttt{Storage}) & = & \triangle[\texttt{managed\_by}\colon \texttt{Manager}, \texttt{category}\colon \texttt{string}, \texttt{maxrisk}\colon \texttt{integer}, \\
 & & \quad \texttt{stock}\colon \{[\texttt{item}\colon \texttt{Material}, \texttt{qty}\colon 10 \div 300]\}] \\
\sigma(\texttt{SStorage}) & = & \texttt{Storage} \\
\sigma(\texttt{DStorage}) & = & \texttt{Storage} \sqcap \triangle[\texttt{stock}\colon \{[\texttt{item}\colon \texttt{DMaterial}]\}] \\
\sigma(\texttt{Manager}) & = & \triangle[\texttt{name}\colon \texttt{string}, \texttt{salary}\colon 40K \div 100K, \texttt{level}\colon 1 \div 15] \\
\sigma(\texttt{TManager}) & = & \texttt{Manager} \sqcap \triangle[\texttt{level}\colon 8 \div 12]
\end{array} \right.
$$

$$
\mathbf{R} \left\{ \begin{array}{ll}
R_1\colon & \texttt{Material} \sqcap (\triangle.\texttt{risk}\colon 10 \div \infty) \rightarrow \texttt{SMaterial} \\
R_2\colon & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\forall.\texttt{item}\colon \texttt{SMaterial}) \rightarrow \texttt{SStorage} \\
R_3\colon & \texttt{Storage} \sqcap (\triangle.\texttt{managed\_by}.\triangle\, \texttt{level}\colon 6 \div 12) \rightarrow (\triangle.\texttt{category}\colon \texttt{''A2''}) \\
R_4\colon & \texttt{Storage} \sqcap (\triangle.\texttt{category}\colon \texttt{''A2''}) \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk}\colon 10 \div \infty) \\
 & \qquad\qquad \rightarrow \triangle[\texttt{managed\_by}\colon \texttt{TManager}]
\end{array} \right.
$$

Tabella 2.3: Schema con Regole di Integrità in linguaggio OCDL

Uno *schema con regole* è una coppia $(\sigma, \mathbf{R})$, dove σ è uno schema e $\mathbf{R}$ un insieme di regole. Ad esempio, il dominio Magazzino rappresentato in Tabella 2.1 ed esteso in 2.2 con l'aggiunta di alcune regole di integrità, è descritto in OCDL tramite lo schema con regole mostrato in Tabella 2.3.

La *funzione interpretazione* $\mathcal{I}$ è una funzione da $\mathbf{S}$ a $2^{\mathcal{V}(\mathcal{O})}$ tale che: $\mathcal{I}[\top] = \mathcal{V}(\mathcal{O})$, $\mathcal{I}[B] = \mathcal{I}_{\mathbf{B}}[B]$[2] , $\mathcal{I}[C] \subseteq \mathcal{O}$, $\mathcal{I}[V] \subseteq \mathcal{O}$, $\mathcal{I}[t] \subseteq \mathcal{V}(\mathcal{O}) - \mathcal{O}$. L'interpretazione è estesa agli altri tipi come segue:

$$
\begin{array}{rcl}
\mathcal{I}[[a_1\colon S_1, \ldots, a_p\colon S_p]] & = & \left\{ [a_1\colon v_1, \ldots, a_q\colon v_q] \,\middle|\, p \leq q, v_i \in \mathcal{I}[S_i], 1 \leq i \leq p, \right. \\
 & & \qquad \left. v_j \in \mathcal{V}(\mathcal{O}), p{+}1 \leq j \leq q \right\} \\
\mathcal{I}[\forall\{S\}] & = & \left\{ \{v_1, \ldots, v_p\} \,\middle|\, v_i \in \mathcal{I}[S], 1 \leq i \leq p \right\} \\
\mathcal{I}[\exists\{S\}] & = & \left\{ \{v_1, \ldots, v_p\} \,\middle|\, \exists i, 1 \leq i \leq p, v_i \in \mathcal{I}[S] \right\} \\
\mathcal{I}[\triangle S] & = & \left\{ o \in \mathcal{O} \,\middle|\, \delta(o) \in \mathcal{I}[S] \right\} \\
\mathcal{I}[S \sqcap S'] & = & \mathcal{I}[S] \cap \mathcal{I}[S']
\end{array}
$$

[2]Assumendo $\mathcal{I}_{\mathbf{B}}$ funzione di *interpretazione standard* da $\mathbf{B}$ a $2^{\mathbf{B}}$ tale che per ogni $d \in \mathbf{D}$: $\mathcal{I}_{\mathbf{B}}[d] = \{d\}$.

Per i tipi cammino abbiamo $\mathcal{I}[(p\colon S)] = \mathcal{I}[(e\colon (p'\colon S))]$ se $p = e.p'$ dove

$$\mathcal{I}[(\epsilon\colon S)] = \mathcal{I}[S],\ \mathcal{I}[(a\colon S)] = \mathcal{I}[[a\colon S]],\ \mathcal{I}[(\triangle\colon S)] = \mathcal{I}[\triangle S],$$

$$\mathcal{I}[(\forall\colon S)] = \mathcal{I}[\forall\{S\}],\ \mathcal{I}[(\exists\colon S)] = \mathcal{I}[\exists\{S\}]$$

Quindi il tipo path è un'abbreviazione per un altro tipo: ad esempio, il tipo path $(\triangle.\texttt{stock}.\forall.\texttt{item}\colon \texttt{SMaterial})$ è equivalente a $\triangle[\texttt{stock}\colon \forall\{[\texttt{item}\colon \texttt{SMaterial}]\}]$.

Si introduce ora la nozione di istanza legale di uno schema con regole come una interpretazione nella quale un valore istanziato in un nome di tipo ha una descrizione corrispondente a quella del nome di tipo stesso e dove sono valide le relazioni di inclusioni stabilite tramite le regole.

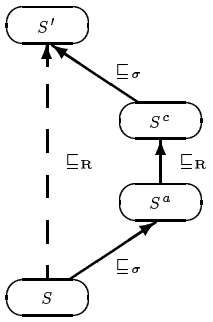
Definizione 1 (Istanza Legale) *Una funzione di interpretazione $\mathcal{I}$ è una* istanza legale *di uno schema con regole $(\sigma, \mathbf{R})$ sse l'insieme $\mathcal{O}$ è* finito *e per ogni $C \in \mathbf{C}, V \in \mathbf{V}, t \in \mathbf{T}, R \in \mathbf{R}$: $\mathcal{I}[C] \subseteq \mathcal{I}[\sigma(C)]$, $\mathcal{I}[t] = \mathcal{I}[\sigma(t)]$, $\mathcal{I}[V] = \mathcal{I}[\sigma(V)]$, $\mathcal{I}[S^a] \subseteq \mathcal{I}[S^c]$.*

Si noti come, in una istanza legale $\mathcal{I}$, l'interpretazione di un nome di classe base *è contenuta* nell'interpretazione della sua descrizione, mentre per un nome di classe virtuale, come per un nome di tipo-valore, l'interpretazione *coincide* con l'interpretazione della sua descrizione. In altri termini, mentre l'interpretazione di una classe base è fornita dall'utente, l'interpretazione di una classe virtuale è calcolata sulla base della sua descrizione. Pertanto, in particolare, le classi virtuali possono essere utilizzate per rappresentare interrogazioni effettuate sulla base di dati. In presenza di classi virtuali cicliche questa computazione non è univoca: fissata un'interpretazione per le classi base, una classe virtuale ciclica può avere più di una interpretazione possibile. Tra tutte queste interpretazioni, noi selezioniamo come istanza legale quella *più grande*, cioè adottiamo una semantica di *massimo punto fisso*. Le definizioni formali di tale semantica e le motivazioni della scelta sono discusse in [12]; in questa sede si vuole soltanto evidenziare il fatto che il modello ammette delle classi virtuali cicliche e quindi il metodo di ottimizzazione proposto è applicabile anche alle query cicliche o ricorsive [13].

Sussunzione ed Espansione Semantica di un tipo

Introduciamo la relazione di sussunzione in uno schema con regole.

Definizione 2 (Sussunzione) *Dato uno schema con regole $(\sigma, \mathbf{R})$, la* relazione di sussunzione *rispetto a $(\sigma, \mathbf{R})$, scritta $S \sqsubseteq_{\mathrm{R}} S'$ per ogni coppia di tipi $S, S' \in \mathbf{S}$, è data da: $S \sqsubseteq_{\mathrm{R}} S'$sse $\mathcal{I}[S] \subseteq \mathcal{I}[S']$ per tutte le istanze legali $\mathcal{I}$ di $(\sigma, \mathbf{R})$.*



$$S = \texttt{Manager} \sqcap (\triangle.\texttt{level}: 5 \div 8)$$
$$S' = (\triangle.\texttt{salary}: 30 \div 70)$$
$$R_1 = \texttt{Manager} \sqcap (\triangle.\texttt{level}: 5 \div 10) \rightarrow (\triangle.\texttt{salary}: 40 \div 60)$$
$$S \sqsubseteq_R S', \;\; S \not\sqsubseteq_\sigma S'$$

Figura 2.1: Relazione di sussunzione dovuta alle regole

Segue immediatamente che $\sqsubseteq_R$ è un preordine (i.e., transitivo e riflessivo ma antisimmetrico) che induce una *relazione di equivalenza* $\simeq_R$ sui tipi: $S \simeq_R S'$ sse $S \sqsubseteq_R S'$ e $S' \sqsubseteq_R S$. Diciamo, inoltre, che un tipo S è *inconsistente* sse $S \simeq_R \bot$, cioè per ciascun dominio l'interpretazione del tipo è sempre vuota.

È importante notare che la relazione di sussunzione rispetto al solo schema σ, cioè considerando $\mathbf{R} = \emptyset$, denotata con $\sqsubseteq_\sigma$, è simile alle relazioni di *subtyping* o *refinement* tra tipi definite nei *CODMs* [14, 15]. Questa relazione può essere calcolata attraverso una comparazione sintattica sui tipi; per il nostro modello tale l'algoritmo è stato presentato in [12].

È immediato verificare che, per ogni S, S', se $S \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_R S'$. Comunque, il viceversa, in generale, non vale, in quanto le regole stabiliscono delle relazioni di inclusione tra le interpretazioni dei tipi che fanno sorgere *nuove* relazioni di sussunzione. Intuitivamente, come viene mostrato nell'esempio di Figura 2.1, se $S \sqsubseteq_\sigma S^a$ e $S^c \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_R S'$.

La relazione esistente tra $\sqsubseteq_R$ e $\sqsubseteq_\sigma$ può essere espressa formalmente attraverso la nozione di *espansione semantica*.

Definizione 3 (Espansione Semantica) *Dato uno schema con regole* $(\sigma, \mathbf{R})$ *e un tipo* $S \in \mathbf{S}$*, l'*espansione semantica *di* S *rispetto a* $\mathbf{R}$*, EXP(S), è un tipo di* $\mathbf{S}$ *tale che:*

1. $EXP(S) \simeq_R S$*;*
2. *per ogni* $S' \in \mathbf{S}$ *tale che* $S' \simeq_R S$ *si ha* $EXP(S) \sqsubseteq_\sigma S'$*.*

In altri termini, EXP(S) è il tipo più specializzato (rispetto alla relazione $\sqsubseteq_\sigma$) tra tutti i tipi $\simeq_R$–equivalenti al tipo S.

L'espressione $EXP(S)$ permette di esprimere la relazione esistente tra $\sqsubseteq_R$ e $\sqsubseteq_\sigma$: per ogni $S, S' \in \mathbf{S}$ si ha $S \sqsubseteq_R S'$ se e solo se $EXP(S) \sqsubseteq_\sigma S'$. Questo significa che, dopo aver determinato l'espansione semantica, anche la relazione

di sussunzione nello schema con regole può essere calcolata tramite l'algoritmo presentato in [12].

È facile verificare che, per ogni $S \in \mathbf{S}$ e per ogni $R \in \mathbf{R}$, se $S \sqsubseteq_\sigma (p : S^a)$ allora $S \sqcap (p{:}\, S^c) \simeq_{\mathrm{R}} S$. Questa trasformazione di S in $S \sqcap (p{:}\, S^c)$ è la base del calcolo della $EXP(S)$: essa viene effettuata iterativamente, tenendo conto che l'applicazione di una regola può portare all'applicazione di altre regole. Per individuare tutte le possibili trasformazioni di un tipo implicate da uno schema con regole $(\sigma, \mathbf{R})$, si definisce la funzione totale $\Gamma : \mathbf{S} \longrightarrow \mathbf{S}$, come segue:

$$\Gamma(S) \quad = \quad \begin{cases} S \sqcap (p : S^c) & \text{se esistono } R \text{ e } p \text{ tali che } S \sqsubseteq_\sigma (p{:}\, S^a) \text{ e } S \not\sqsubseteq_\sigma (p{:}\, S^c) \\ S & \text{altrimenti} \end{cases}$$

e poniamo $\tilde{\Gamma} = \Gamma^i$, dove i è il più piccolo intero tale che $\Gamma^i = \Gamma^{i+1}$. L'esistenza di i è garantita dal fatto che il numero di regole è finito e una regola non può essere applicata più di una volta con lo stesso cammino ($S \not\sqsubseteq_\sigma (p{:}\, S^c)$). Si può dimostrare che, per ogni $S \in \mathbf{S}$, EXP(S) è effettivamente calcolabile tramite $\tilde{\Gamma}(S)$.

Esempio applicativo

Riprendiamo l'esempio descritto all'inizio di questo capitolo (in Sezione 2.1.1, e vediamo a quali vantaggi pratici può portare l'uso della logica **OCDL**.
L'attività di inferenza iniziale da parte del sistema, effettuauta da OCDL-Designer, consiste nel determinare l'espansione semantica di ogni classe dello schema. Questo permette di rilevare relazioni *isa* non esplicitate nella definizione dell classi stesse. Ad esempio, nel calcolo dell'espansione semantica della classe `DMaterial` si rileva che tale classe è sussunta (implica) l'antecedente della regola R1: congiungendone quindi il conseguente si determina che `DMaterial` è una specializzazione di `SMaterial`. Nel caso in cui vengano applicate più regole durante l'espansione semantica di una classe, le relazioni *isa* ricavate sono meno ovvie di quella appena descritta. Ad esempio, nel calcolo dell'espansione semantica della classe `DStorage` viene applicata prima la regola R1 e successivamente la regola R2, concludendo che `DStorage` è una specializzazione di `SStorage`.

L'altra componente del sistema, ODB-QOptimizer, si occupa invece dell'ottimizzazione semantica delle interrogazioni a run-time, anch'essa basata sull'espansione semantica, in modo da specializzare le classi interessate dall'interrogazione. Prendiamo ad esempio la seguente query, espressa in *OQL*, che vuole selezionare dallo schema tutti i magazzini che contengono materiali che hanno tutti rischio maggiore o uguale a 15.

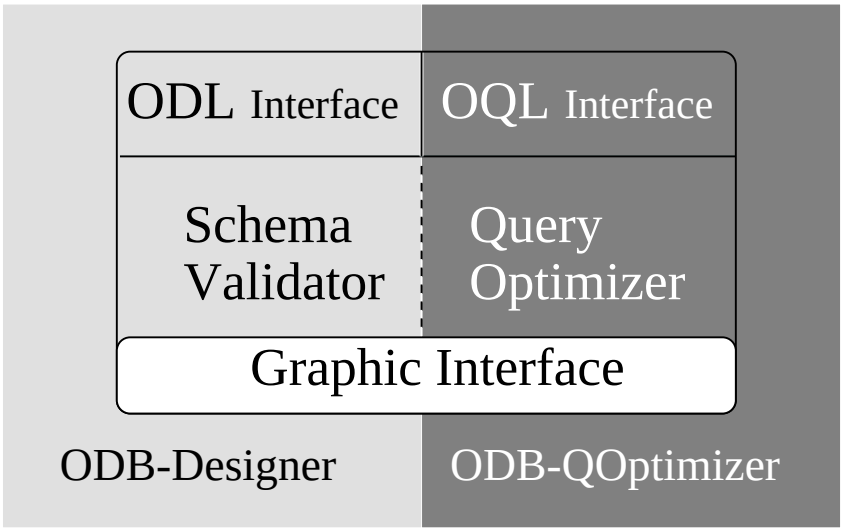


Figura 2.2: ODB-Tools

```
select  * from Storage S
where   for all X in S.stock:( X.item in Material and
                               X.item.risk ≥ 15 )
```

L'espansione semantica di quest'interrogazione, ottenuta trasformando la query stessa in un classe virtuale **OCDL** ed applicando prima la regola R1 e successivamente la regola R2, porta alla seguente interrogazione equivalente:

```
select  * from SStorage S
where   for all X in S.stock:( X.item in SMaterial and
                               X.item.risk ≥ 15 )
```

L'esempio mostra un'effettiva ottimizzazione della query, indipendente da un qualsiasi modello di costo: sostituendo `Storage` con `SStorage` e `Material` con `SMaterial` si riduce l'insieme di oggetti da controllare per individuare il risultato dell'interrogazione.

2.1.2 Architettura degli ODB-Tools

Tutti gli strumenti software che si basano sull'uso della logica descrittiva **OCDL**, e che sono già stati sviluppati presso il Dipartimento di Scienze dell'Ingegneria dell'Università di Modena, sono visibili ed utilizzabili attraverso il sito web *http://www.sparc20.dsi.unimo.it*.
ODB-Tools, la cui architettura è mostrata in Figura 2.2, fornisce i seguenti servizi:

- *validazione di schemi* (ODB-Designer): l'utente può inserire la descrizione di uno schema di una base di dati, usando il linguaggio ODL, e il sistema realizza automaticamente la validazione dello schema (verificando che non esistano classi incoerenti, ovvero che non possono essere popolate da alcun oggetto) e la sua riclassificazione (determinando eventuali relazioni di specializzazione non esplicitate dallo schema stesso). Al suo interno è quindi presente un'interfaccia tra ODL e **OCDL**;

- *ottimizzazione semantica delle interrogazioni* (ODB-QOptimizer): l'utente può inserire una query, in OQL, posta su uno schema predefinito, e questa viene automaticamente riformulata attraverso il procedimento precedentemente descritto. Fa uso di un'interfaccia di traduzione da OQL a **OCDL** e viceversa.

2.2 Tecniche di Integrazione di Schemi

Di seguito verrá fatta una descrizione delle tecniche di integrazione degli schemi, strumenti sui quali si basa lo studio fatto in questa tesi.
Il problema principale che bisogna affrontare nella fase di analisi ed integrazione degli schemi é che una stessa informazione, in sorgenti eterogenee, puó essere rappresentata in modi diversi, o avere significati e scopi differenti.
In tal caso si parla di eterogeneitá semantica delle sorgenti da integrare.

L'idea base, per una soluzione a tale problema, é la costruzione di un *dizionario semantico* all'interno di un dominio che si vuole integrare per rendere piú facile sia la fase di studio della similaritá dei concetti comuni, sia quella di reperimento delle informazioni.
Sono state, quindi, sviluppate tecniche che supportano la definizione e l'organizzazione del *dizionario semantico*, che vedremo in seguito, mediante l'analisi degli schemi sorgenti.

2.2.1 Architettura del Dizionario Semantico

Nel dizionario semantico sono definite delle *Gerarchie di Concetti* dove al livello piú alto ci sono o concetti piú generali rispetto a quelli dei livelli inferiori oppure sono composti da essi, come mostrato in Fig. 2.3.

Ad ogni concetto di livello piú basso della gerarchia (chiamato concetto *bottom*) é associato un *cluster* di entitá semanticamente simili appartenenti peró a schemi differenti.
I concetti sono connessi tra loro da links (collegamenti), i quali possono essere di tre tipi:

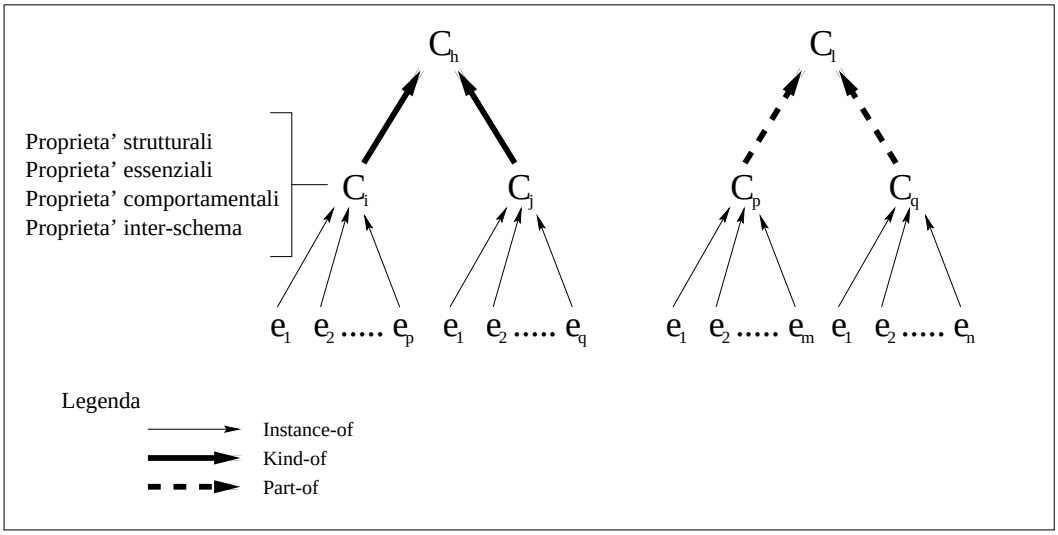


Figura 2.3: Gerarchia di concetti nel dizionario semantico

- **Instance-of**: collegamenti tra un concetto bottom C_k e ogni entitá e_i appartenente al claster associato a C_k.
- **Kind-of**: collegamenti tra coppie di concetti per rappresentare una relazione del tipo *is-a*. Cioé, una connessione Kind-of da C_h a C_k sta a significare che C_h é un subconcetto di C_k (ad esempio `Public Organization` **Kind-of** `Organization`).
- **Part-of**: collegamenti che rappresentano delle relazioni di *aggregazione* tra coppie di concetti. Un link Part-of da C_h a C_k denota che C_h é un componente di C_k (ad esempio `Unit` **Part-of** `Organization`).

In aggiunta a questi links, per i concetti bottom sono mantenuti anche un insieme di *proprietá* che servono a descrivere tali concetti. In particolare, per un concetto C_k, sono memorizzati i seguenti insiemi di proprietá:

- PROPRIETÁ STRUTTURALI: é un insieme di proprietá ognuna delle quali specifica un attributo di C_k. Esse sono definite come l'unione delle proprietá strutturali delle istanze e_i del concetto C_k.
- PROPRIETÁ ESSENZIALI: é un sottoinsieme delle proprietá strutturali, il quale specifica gli attributi essenziali per caratterizzare un concetto e le sue istanze e_i rispetto ad altri concetti presenti nel dizionario. Possono essere identificate esaminando gli attributi comuni alle diverse istanze e_i di un concetto C_k.

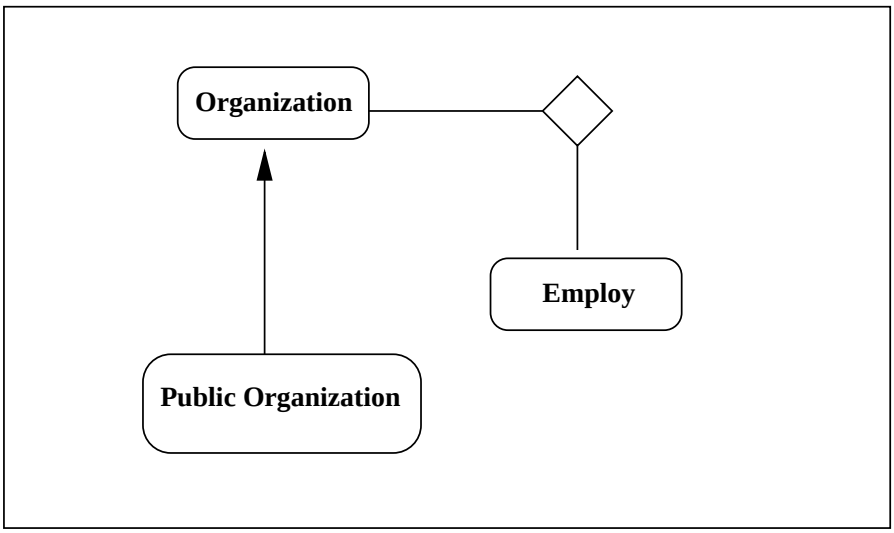


Figura 2.4: Esempio di schema concettuale

- PROPRIETÁ COMPORTAMENTALI: é un insieme di proprietá, dove ognuna di esse specifica una operazione che puó essere fatta. Tali proprietá, per ogni concetto C_k, sono ottenute dall'unione delle proprietá comportamentali delle sue istanze e_i.

- PROPRIETÁ INTER-SCHEMA: ogni proprietá di questo insieme specifica mutue relazioni tra le entitá di un concetto. In particolare, se $EXT(e_i)$ é l'insieme di istanze di un entitá e_i in un determinato database, date due entitá e_i e e_j appartenenti ad un cluster C_k, si possono avere quattro tipi di proprietá inter-schema:

  1. e_i EQUIVALENT e_j: sse $EXT(e_i) \equiv EXT(e_j)$;
  2. e_i CONTAINED e_j: sse $EXT(e_i) \subset EXT(e_j)$;
  3. e_i OVERLAPPING e_j: sse $EXT(e_i) \cap EXT(e_j) \neq \emptyset$ e e_i CONTAINED e_j non è verificato;
  4. e_i DISJOINTNESS e_j: sse $EXT(e_i) \cap EXT(e_j) = \emptyset$

2.2.2 Costruzione del Dizionario Semantico

La costruzione di un dizionario semantico é basata sull'analisi di un insieme di schemi concettuali delle sorgenti da integrare, cercando di limitare il piú possibile l'intervento dell'utente. Nel seguito verrá usato il modello E/R per rappresentare uno schema concettuale. Riportiamo in figura 2.4 lo schema usato come esempio

di riferimento in questo capitolo.
In particolare, si distinguono tre fasi:

1. Analisi degli schemi
2. Identificazione di entitá semanticamente simili
3. Definizione delle gerarchie di concetti

Analisi degli Schemi

L'obiettivo di questa fase é l'indentificazione di *relazioni terminologiche* tra nomi di entitá appartenenti a schemi differenti allo scopo di valutarne la loro similaritá semantica. In particolare sono prese in considerazione:

- relazioni derivate dall'analisi dei nomi assegnati agli elementi degli schemi, come per esempio i sinonimi i quali sono considerati un importante indicatore della similaritá semantica.
- relazioni derivate dall'analisi dei contesti delle entitá negli schemi ER, con riferimento ad attributi, gerarchie di generalizzazione, e relazioni che caratterizzano una entitá in un dato schema. Infatti, le entitá semanticamente simili sono generalmente caratterizzate dalla presenza di elementi comuni nei loro contesti, sia in termini di attributi che di entitá connesse.

Per poter rappresentare e codificare tali informazioni, facendo riferimento per gli esempi allo schema E/R di Figura 2.4, sono state introdotte le seguenti relazioni terminologiche binarie tra coppie di termini:

- SYN (synonyms term): definita tra nomi che sono considerati sinonimi, in quanto possono essere rimpiazzati tra loro in tutti gli schemi senza che cambi il loro significato (per esempio, `Worker` SYN `Employee`). La relazione SYN puó essere identificata contando sulla conoscenza di un dominio generico sottoforma di un thesaurus esistente, e sulla conoscenza di un dominio specifico sottoforma di regole e convenzioni adottate nella fase di assegnazione dei nomi a elementi degli schemi analizzati.
- BT (Broader Terms): definita tra due nomi tali che il primo ha un significato piú generale del secondo. La Relazione BT é ottenuta analizzando le gerarchie di generalizzazione di schemi concettuali. In particolare, si definisce una relazione (n_i BT n_j) tra due nomi n_i e n_j se una entitá e_i con nome n_i esiste in qualche schema, il quale é una generalizzazione di una entitá e_j con nome n_j (ad esempio, `Organization` BT `Public Organization`).

- RT (Related Terms): definita tra nomi che sono generalmente usati nello stesso contesto. In particolare, é definita una relazione (n_i RT n_j) tra due nomi n_i e n_j in uno dei seguenti casi:
  - un'entitá e_i con nome n_i ha un attributo nello stesso schema il cui nome é n_j;
  - un entitá e_i con nome n_i eín associazione con una entitá e_j con nome n_j (ad esempio, `Organization` RT `Employee`).

Tutte le relazioni scoperte con l'analisi degli schemi sono propriamente immagazzinati in un Thesaurus, che diventa in questo modo la base di conoscenza terminologica dell'intero sistema.

Identificazione di entitá semanticamente simili

In questa fase viene presentato il modo con cui vengono usate le relazioni terminologiche per valutare la similaritá semantica (o affinitá) tra entitá appartenenti a schemi differenti.
A tale scopo é assegnato, per ogni tipo di relazione, un valore compreso tra zero e uno (chiamato `forza` e indicato con la lettera σ) che sta ad indicare l'importanza ed il peso che essa ha nella valutazione della similaritá semantica, per cui piú sará stringente la relazione tra due termini diversi piú alto sará il valore associato. In particolare, é stato assegnato alla relazione SYN il peso piú alto, poiché essa indica una affinitá piú stretta rispetto alle altre relazioni, cosí come per $\sigma_{BT} \geq \sigma_{RT}$.
Poiché nel thesaurus ogni coppia di nomi di entitá possono essere connessi da due tipi di relazioni terminologiche: quelle *esplicite*, che sono memorizzate nel thesaurus e quelle *implicite*, che possono essere derivate da quelle esplicite e corrispondono ai percorsi di relazioni terminologiche tra nomi, il coefficiente di similaritá semantica é valutato come una combinazione dei pesi delle relazioni verificate per i nomi di entitá considerati (una descrizione piú dettaglata dell'intero procedimento verrá data nel capitolo 4).

Definizione delle gerarchie di concetti

I concetti nel dizionario semantico sono definiti come rappresentativi di cluster di entitá di databases locali che descivono dati semanticamente simili [16]. I coefficienti di similaritá semantica possono essere usati per definire le gerarchie di concetti, seguendo un approccio *basato sull'integrazione* oppure mediante un approccio *basato sulla scoperta*.
Nel primo caso, gli schemi concettuali dei databases sono considerati contemporaneamente e propriamente integrati in gerarchie di concetti. Per cui, i coefficienti

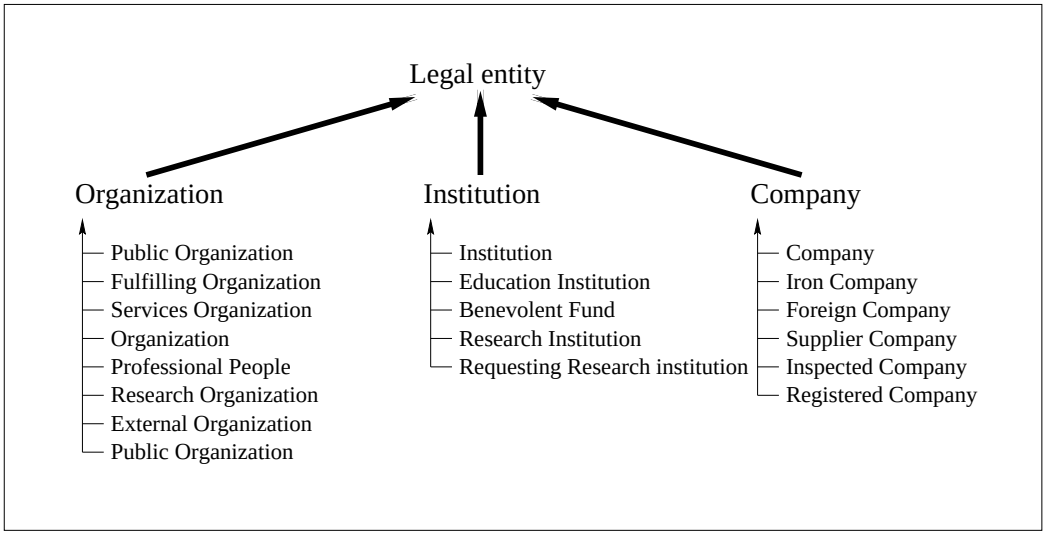


Figura 2.5: Esempio di gerarchia di concetti

di affinitá sono calcolati (e memorizzati in una matrice) per tutte le possibili coppie di entitá che devono essere considerate, sfruttando sia le affinitá tra nomi delle entitá sia le affinitá tra i nomi degli attributi di queste. Sulla base di questa matrice quadrata attraverso un algoritmo iterativo vengono riorganizzate in cluster le entitá che hanno tra di loro un livello di affinitá piú alto. Un esempio dell'intero processo di integrazione é riportato in Figura 2.5: un insieme di entità, estratte da database diversi della Pubblica Amministrazione, sono stati riorganizzati, dopo aver computato i loro coefficienti di affinità, all'interno di tre cluster, ognuno dei quali è quindi ora rappresentativo di un insieme definito e limitato di entità locali. Da notare che, una volta identificati i cluster, è possibile riorganizzarli in una struttura simile ad un vero e proprio schema concettuale, attraverso la definizione di ulteriori anelli di collegamento tra concetti *bottom* per definire relazioni di specializzazione e di aggregazione, arrivando in questo modo alla creazione di una gerarchia di concetti (in Figura 2.5 i cluster ottenuti sono definiti specializzazioni del concetto `Legal Entity`).
Nel secondo caso invece, i concetti gerarchici sono scoperti e quindi definiti in modo incrementale considerando uno schema alla volta.

Capitolo 3

Il progetto MOMIS

Il progetto MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) è nato da una collaborazione tra il gruppo di lavoro della Professoressa Sonia Bergamaschi e lo staff della Dottoressa Silvana Castano (del Dipartimento di Scienze dell'Informazione dell'Università di Milano) con lo scopo di sviluppare un sistema di integrazione di sorgenti eterogenee di informazione. In particolare, con questa tesi, si è cercato di arricchire la fase di analisi degli schemi fornendo un modulo software che definisce il livello di affinitá tra le classi.
In questo capitolo viene fatta una descrizione del mediatore MOMIS al fine di rendere piú chiara la collocazione della presente tesi all'interno di tale progetto.

3.1 Introduzione

Come é stato visto nel capitolo introduttivo, l'aumento di informazioni disponibili (sia sulle rete Internet, che all'interno di una azienda) ha evidenziato la necessitá di avere a disposizione dei sistemi che non solo facilitino il loro reperimento (in quanto questo puo' avvenire con dei motori di ricerca), ma che diano anche una visione integrata di queste, in modo da eliminare incongruenze e ridondanze necessariamente presenti tra di loro. In genere le architetture di tali sistemi di integrazione di dati sono basate sul concetto di *mediatore*, un modulo il cui compito é quello di reperire ed integrare informazioni presenti in una molteplicitá di basi eterogenee di dati [7]. In letteratura sono già stati presentati diversi sistemi diretti all'integrazione di database convenzionali, come pure applicabili all'integrazione di dati semi-strutturati. Di questi, secondo quanto esposto in [18], si può realizzare una categorizzazione sulla base del diverso tipo di approccio utilizzato, distinguendoli in *semantici* e *strutturali*.
Per quanto riguarda l'approccio *strutturale* esso é caratterizzato dai seguenti punti:

- per trattare tutti i singoli oggetti presenti nel sistema é utilizzato un mod-

ello comune dei dati di tipo *self-describing*, rendendo inutili gli schemi concettuali delle sorgenti.

- l'utilizzo di un linguaggio self-describing facilita l'integrazione soprattutto di dati semi-strutturati.
- le uniche informazioni semantiche sono inserite manualmente da un operatore attraverso l'utilizzo di linguaggi descrittivi
- l'assenza di uno schema globale non permette, soprattutto in caso di database di grandi dimensioni, una ottimizzazione semantica delle interrogazioni
- l'intero onere dell'integrazione é a carico del progettista del mediatore, in quanto sono eseguibili solo le interrogazioni predefinite dall'operatore.

Altri progetti, e fra questi MOMIS, seguono invece un approccio che si può definire *semantico*, e che è caratterizzato da diverse peculiarità:

- per ogni sorgente sono a disposizione dei metadati, codificati nello schema concettuale;
- nello schema concettuale sono pure presenti le informazioni semantiche, che possono essere utilmente sfruttate sia nella fase di integrazione delle sorgenti, sia in quella di ottimizzazione delle interrogazioni;
- deve essere presente nel sistema, per poter descrivere in modo uniforme tutte le informazioni che devono essere condivise, un modello di dati comune;
- è realizzata effettivamente una unificazione (parziale o totale) degli schemi concettuali (in modo semi-automatico), per arrivare alla definizione di uno schema globale.

Per MOMIS si é stabilito di ricorrere, per la rappresentazione dello schema globale, all'adozione del modello ad oggetti standard ODMG-93 [20], esteso per le problematiche di integrazione. Diverse sono le motivazioni che possono portare alla adozione di un approccio di questo tipo, unito all'utilizzo di un modello ad oggetti:

- la presenza di una schema globale permette all'utente di formulare qualsiasi interrogazione che sia consistente con lo schema
- la possibilitá, usando le informazioni semantiche comprese nello schema globale, di una eventuale ottimizzazione di queste interrogazioni;

- L'uso delle primitive di generalizzazione e di aggregazione, proprie dei modelli ad oggetti, permette la riorganizzazione delle conoscenze estensionali;

- l'adozione di una semantica *type as a set* per gli schemi permette di controllarne la consistenza, facendo riferimento alle loro descrizioni;

- notevoli sforzi sono stati realizzati per lo sviluppo di standard rivolti agli oggetti: CORBA [19] per lo scambio di oggetti attraverso sistemi diversi; ODMG-93 (e con esso i modelli ODM e ODL per la descrizione degli schemi, e OQL come linguaggio di interrogazione) per lo sviluppo di object-oriented database;

- l'adozione di una semantica di *mondo aperto* permette di poter utilizzare tale ambiente anche in presenza di dati semi-strutturati: in tal caso gli oggetti di una classe condividono una struttura minima comune (che è quindi la descrizione della classe stessa), ma possono avere ulteriori proprietà non esplicitamente comprese nella struttura della classe di appartenenza.

Per la descrizione degli schemi delle sorgenti e del Mediatore sono stati proposti in MOMIS un modello di dati comune e un linguaggio di definizione comune (ODL_{I^3}) il quale verrá descritto nel prossimo capitolo. Come vedremo ODL_{I^3} costituisce un estensione del corrispondente linguaggio di definizione ODL proposto dal gruppo di standardizzazione ODMG-93. Inoltre, la logica descrittiva **OCDL**(vedi Sezione 2.1.1) è utilizzata come linguaggio *kernel* del sistema, sia per automatizzare la fase di integrazione, sia per la fase di ottimizzazione delle interrogazioni, attraverso l'ausilio degli ODB-Tools.

L'approccio proposto in MOMIS é costituito principalmente di due fasi:

1. **unificazione degli schemi:** è la fase presentata in [4] ed è quella studiata ed approfondita nel Capitolo 4. In questa fase l'uso della logica descrittiva **OCDL**, insieme alle tecniche di clustering riportate nella Sezione 2.2 ed in [16], permettono la realizzazione di una processo semi-automatico di integrazione degli schemi, fino a pervenire alla definizione di uno schema globale, direttamente interrogabile dall'utente, che rappresenti l'unione di tutti gli schemi locali, rimuovendone differenze e ripetizioni;

2. **querry processing:** in questa fase a partire da una interrogazione dell'utente posta sullo *schema globale*, si giunge alla definizione di un insieme di interrogazione indirizzate alle varie sorgenti, ed alla presentazione di un'unica risposta.

3.2 Architettura

L'architettura del sistema MOMIS é mostrata in Fig. 3.1, come si puó notare é stata mantenuta la struttura classica di sistema I^3. Essa puo' essere suddivisa in quattro livelli, che sono:

1. ***sorgenti***: sono al livello piú basso e rappresentano le fonti di informazione che devono essere integrate dal sistema. Possono essere sia dei database tradizionali (ad oggetti, o basati sul modello relazionale), che semplici file system, mentre é in via di sviluppo una estensione del linguaggio di definizione ODL_{I^3} per il supporto di dati semi-strutturati;

2. ***wrapper***: rappresentano l'interfaccia tra il mediatore e le varie sorgenti locali di dati, perció devono essere sviluppati appositamente per il tipo di sorgente che andranno ad interfacciare. I Wrapper hanno una doppia funzione:

   - nella fase di integrazione, essi sono responsabili della descrizione degli schemi delle sorgenti nel linguaggio ODL_{I^3};
   - nella fase di querry processing, devono tradurre le query ricevute dal mediatore (espressa nel linguaggio comune di interrogazione OQL_{I^3}) in interrogazioni comprensibili dalla sorgente con cui ognuno di essi opera. Inoltre deve presentare al mediatore, attraverso il modello comune di dati utilizzato dal sistema, i dati ricevuti in risposta alla query presentata;

3. ***mediatore***: é un modulo software che combina, integra e ridefinisce gli schemi ODL_{I^3} ricevuti dai wrappers. Puó essere definito il cuore del sistema, ed é costituito da diversi sottomoduli quali:

   - *Global Schema Builder*: é il modulo di integrazione degli schemi locali. Partendo dalle descrizioni delle sorgenti espresse mediante il linguaggio descrittivo ODL_{I^3}, genera un unico schema globale da presentare all'utente. Questa fase di integrazione, realizzata in modo semi-automatico con l'interazione del progettista del sistema, utilizza gli ODB-Tools, le tecniche di clustering e quelle di fusione delle gerarchie;
   - *Query Manager*: é il modulo che gestisce le interrogazioni. Provvede a gestire la query dell'utente, prima deducendone da essa un insieme di sottoquery da spedire alle fonti locali, poi *ricomponendo* le informazioni da esse ottenute, ed inoltre ha anche il compito di

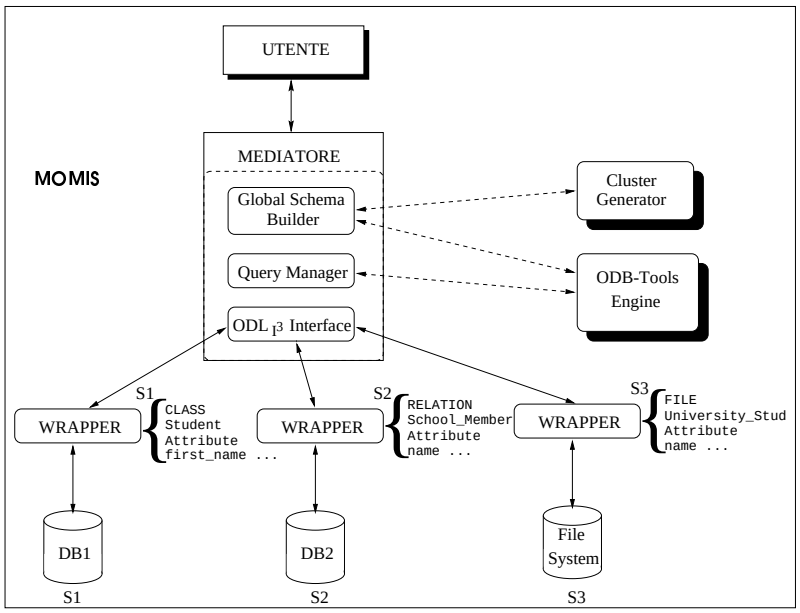


Figura 3.1: Architettura del sistema I^3 MOMIS

provvedere, utilizzando gli ODB-Tools, all'ottimizzazione semantica delle interrogazioni;

- *ODL_{I^3} interface*: é l'interfaccia che il mediatore usa per interfacciarsi con i wrapper in modo uniforme, utilizzando il modello comune. In questo modo l'onere delle traduzioni (dei dati, delle interrogazioni e delle descrizioni degli schemi) é lasciato ai rispettivi wrappers;

4. ***utente***: é colui che interagisce con il mediatore e al quale viene presentato lo schema globale, nel cui interno sono rappresentati tutti i concetti che possono essere interrogati. In pratica per l'utente é come se si trovasse di fronte ad un unico database da interrogare in modo tradizionale, mentre le sorgenti locali restano completamente trasparenti.

Utilizzando questa architettura, ci si é preposti con MOMIS la realizzazione di un sistema di mediazione che contribuisca a realizzare un *accesso integrato* alle sorgenti di informazione al fine di permettere all'utente di porre una singola query ed ottenere un unica risposta unificata.

Per realizzare l'integrazione degli schemi, sono state arricchite con un componente *intelligente* le tecniche di integrazione basate sul clustering, ampliandole con fasi automatiche realizzate sfruttando gli ODB-Tools, in modo da diminuire il più possibile l'apporto manuale del progettista del sistema. Inoltre queste tecniche di clustering sono state anche rifinite, al fine di ampliare quella che prima (vedi

Sezione 2.2) era una semplice analisi terminologica di nomi di classe e attributi, e che adesso é diventata una analisi di nomi e di domini dei tipi che devono essere integrati.

3.3 Aspetti generali dell'approccio

In questa sezione viene data una breve descrizione dell'approccio all'integrazione intelligente di schemi, al fine di avere una visione generale del comportamento di MOMIS. Per una descrizione piú dettagliata si rimanda al prossimo capitolo, dove con l'aiuto di un esempio di riferimento verranno esaminati tutte le fasi di tale approccio.

L'approccio *semantico* utilizzato si articola nei seguenti passi:

- *Generazione di un Thesaurus comune*: l'obiettivo di questa fase é la costruzione di un thesaurus di *relazioni terminologiche* riferite a termini di classi appartenenti a sorgenti diverse. In pratica si cerca di ottenere, attraverso l'utilizzo degli ODB-Tools, in modo semi-automatico (in quanto é richiesta l'interazione col progettista) nuove relazioni di sinonimia e corrispondenza tra i termini (intendendo sia nomi di classi che di attributi) delle diverse sorgenti. Questo passo é realizzato dal modulo $\mathbf{SIM}_1$ e rappresenta il file di input del modulo **ESCA** realizzato durante lo sviluppo di questa tesi;

- *Analisi delle Affinitá delle classi*: in questa fase le relazioni terminologiche memorizzate nel thesaurus sono utilizzate per valutare l'*affinitá* tra classi di una stessa sorgente o di sorgenti diverse.
  L'affinitá tra due classi é stabilita attraverso appropriati *coefficienti di affinitá,* che prendono in considerazione sia i nomi delle classi che i loro attributi (considerando per ogni attributo sia il nome che il dominio). Il calcolo dei coefficienti di affinitá é realizzato dal modulo software **ESCA**;

- *Generazione dei clusters*: utilizzando i coefficienti di affinitá calcolati nella fase precedente, le classi tra loro affini vengono raggruppati in cluster, usando delle tecniche di clusterizzazione gerarchica. L'obiettivo di questa fase é di identificare le classi che devono essere integrate, in quanto descrivono le informazioni identiche o semanticamente simili;

- *Generazione dello schema globale del mediatore*: l'unione di classi affini porta alla creazione dello schema globale del mediatore. Per ogni claster é definita una classe *globale* che é rappresentativa di tutte le classe appartenti ad esso, ed é caratterizzata dall'unione degli attributi di ogni singola

classe. In questo modo lo schema globale sará composto da tutte queste classi globali e sará, inoltre, la base per porre le query sui dati delle sorgenti.

Una volta che lo schema globale del mediatore é stato costruito (da tener presente che questa operazione deve essere eseguita solo in fase di inizializzazione del sistema o di acquisizione di una nuova sorgente da integrare), il sistema puó essere interrogato; l'utente puó porre le proprie query senza doversi preoccupare delle sorgenti dove verranno recuperate le informazioni. Inoltre, l'utente sarà pure esonerato dal conoscere i diversi linguaggi di interrogazione locali delle fonti integrate: è compito del Query Manager ottimizzare le query ricevute e spedirle alle sorgenti che da queste devono essere interrogate (durante la fase di Query Reformulation, che descriveró nella prossima sezione).

3.4 Query Reformulation

In questa sezione verrà descritto il processo che, sfruttando le informazioni memorizzate nelle mapping table (struttra dati tabellare che contiene la definizione di una classe globale in ODL_{I^3} e le informazioni in esse contenute, vedi Sezione 4.6), realizza la Query Reformulation. Scopo di questo processo, che verrà attivato ogniqualvolta l'utente pone una interrogazione sullo schema globale (esprimendola dunque in termini di classi e attributi globali), è arrivare alla definizione automatica delle interrogazioni che devono essere spedite alle diverse fonti di informazione per dare risposta alla query originale.
In accordo con altri approcci *semantici*, questo processo consiste di tre fasi distinte:

- *ottimizzazione semantica*
- *formulazione del query plan*
- *ottimizzazione basata sulla conoscenza estensionale*

Ottimizzazione Semantica

In questa fase il mediatore MOMIS opera sulla query dell'utente sfruttando le tecniche di ottimizzazione semantica supportate dagli ODB-Tools, al fine di ridurre il costo del piano di accesso che sará generato. La query é rimpiazzata da una nuova la quale contiene ogni possibile restrizione che non é presente nella query originale ma che é logicamente implicata da essa stessa e dalle regole definite sullo schema, per poter poi sfruttare eventuali indici ausiliari presenti nelle sorgenti locali.
Per questa fase di ottimizzazione sono necessarie le regole di integritá definite

sullo schema globale; anche se questa ipotesi puó sembrare molto forte (infatti le regole sono definite dal progettista, il quale deve essere a conoscenza di condizioni valide su tutte le estensioni di tutte le classi appartenenti al cluster da cui deriva una determinata classe globale) non é improbabile che in determinati domini applicativi (facendo riferimento a databases appartenenti a domini fortemente regolamentati) la definizione di queste regole sia possibile.

Formulazione del Query Plan

Una volta che il sistema MOMIS ha prodotto la query eventualmente ottimizzata, deve essere costruito un piano di accesso, il quale consiste nel formulare un insieme di subquery da fornire alle sorgenti d'informazione locale, per andare a recuperare le informazioni da dare in risposta all'utente; mentre si rimanda alla Sezione 3.5 per la presentazione della fase di riunificazione dei dati.

Si supponga che la query sia stata posta facendo riferimento ad una classe globale, alla quale corrisponderà in memoria una mapping table: per ogni classe appartenente al cluster da cui la classe globale ha avuto origine, deve essere formulata una apposita interrogazione, che ne utilizzi la terminologia adeguata. In particolare, il sistema deve riformulare la query di partenza, per ogni classe, esprimendola utilizzando i termini della classe locale che si sta considerando. A questo scopo é stato inoltre pensato un algoritmo di *controllo ed eliminazione*, la cui funzione è eliminare dalla lista di classi da interrogare quelle nelle quali si è già sicuri, basandosi sulle informazioni memorizzate nella mapping table, di non trovare dati *utili* o comunque *verificabili*.
Per quanto riguarda le fonti che fornirebbero dati non *utili*, ci si basa sui valori di default della mapping table: se il valore definito di default per un attributo in una classe locale è diverso dal valore specificato nella query, la classe non sarà interrogata.
Per le sorgenti, invece, che fornirebbero dati non *verificabili* ci si basa sui valori nulli presenti nella mapping table: in particolare, se è stata specificata una condizione su un attributo globale per il quale non esiste un attributo locale corrispettivo nella classe che si sta analizzando, si è scelto di non procedere all'interrogazione della classe, non potendo essere verificata quella condizione. È stata quindi fatta una scelta drastica, per mantenere la sicurezza di ricevere solamente dati completamente verificati, ma sarebbero comunque possibili scelte differenti: tra queste, si potrebbe ad esempio ipotizzare l'esistenza di un livello di *credibilità* delle risposte, che l'utente dovrebbe scegliere, che esprima la percentuale delle condizioni che si vogliono siano assolutamente verificate dai dati ricevuti in risposta all'interrogazione.

L'algoritmo di controllo ed eliminazione è riportato in Figura 3.2. Sup-

Algoritmo di Controllo ed Eliminazione
/* **Input:** Cluster Cl_j e k classi appartenenti al cluster */

begin procedure:
for each classe $c_h \in Cl_j$ **do**
for each fattore boolean **do**
if (attributo globale ∈ fattore boolean) corrisponde a:
1. valore nullo in c_h: nessuna query locale viene generata per c_h;
2. valore di default in c_h: **if** valore specificato nel fattore boolean factor
e' diverso da quello di default
then nessuna query locale viene generata per c_h;
end procedure.

Figura 3.2: Algoritmo di Controllo ed Eliminazione

poniamo che il cluster corrispondente alla classe globale interessata dalla interrogazione sia Cl_j: per ogni classe c_h appartenente al cluster sono controllati nella mapping table tutti i corrispondenti locali degli attributi globali sui quali, nella query, sono state poste condizioni in **and** (denominate fattori boolean). Nel caso in cui, dopo la fase di controllo ed eliminazione, la query sia da spedire a quella classe, tutti i termini globali sono naturalmente tradotti nei corrispondenti termini locali, al fine di rendere *comprensibile* dalla sorgente la nuova query generata.

Ottimizzazione basata sulla Conoscenza Estensionale

Fino ad ora all'interno di MOMIS sono state analizzate informazioni inerenti le intensioni, ovvero le strutture, delle classi che devono essere integrate (siano esse informazioni semantiche o terminologiche). Potrebbero invece essere sfruttate anche eventuali informazioni riguardanti le estensioni di queste classi, fino ad ora escluse dalla trattazione principalmente per due motivi, infatti esse potrebbero essere fornite in due modi:

- manuale
- automatico

Nel primo caso il progettista dovrebbe controllare fisicamente tutte le estensioni di tutte le classi coinvolte nel sistema, per vedere se vi sono tra loro dati duplicati, o in qualche modo correlati; il che comporterebbe un enorme mole di lavoro.
Nel secondo caso, invece, bisognorebbe ideare un algoritmo di controllo e confronto dei dati memorizzati nelle varie sorgenti (si pensi ad esempio che

questo confronto dovrebbe essere riattivato successivamente ad ogni operazione di aggiornamento, o di inserimento,...), il che non è comunque un compito banale.

Vi sono però casi in cui, magari grazie a conoscenze a priori (può essere il caso di più database derivati da un'unica sorgente di partenza) o a un confronto tra i progettisti stessi delle sorgenti locali, si può supporre che sia possibile definire relazioni che intercorrono tra le estensioni delle classi che sono state integrate dal mediatore.
Per questi casi, sono state definite quattro tipologie di proprietà *inter-schema* atte a esprimere mutue relazioni esistenti tra le estensioni.
In particolare, sia $EXT(c)$ l'insieme delle istanze (*estensione*) della classe $c \in Cl_i$. Esempi di proprietà interschema che possono essere definite tra due classi c e c' a livello estensionale sono i seguenti:

1. EQUIVALENZA: due classi c e c' sono equivalenti a livello estensionale, denotato da $c \equiv_{ext} c'$, se e solo se $EXT(c) \equiv EXT(c')$;

2. CONTENIMENTO: due classi c e c' sono contenute l'una nell'altra a livello estensionale, denotato da $c \subset_{ext} c'$, se e solo se $EXT(c) \subset EXT(c')$;

3. INTERSEZIONE: due classi c e c' sono intersecate a livello estensionale, denotato da $c \cap_{ext} c'$, se e solo se $EXT(c) \cap EXT(c') \neq \emptyset$ e $c \subset_{ext} c^{'}$ non è verificato;

4. DISGIUNZIONE: due classi c e c' sono disgiunte a livello estensionale, denotato da $c \emptyset_{ext} c'$, se e solo se $EXT(c) \cap EXT(c') = \emptyset$.

Supponiamo di avere, per esempio, due classi `School_Member` e `University_Student` appartenenti rispettivamente a due sorgenti diverse S_1 e S_2 ed inserite in un dato cluster Cl, e che il progettista sia in grado di definire, a livello estensionale, la seguente proprietà:
`School_Member` $\subset_{ext}$ `University_Student`
valida tra le classi appartenenti al cluster.

Il sistema potrebbe sicuramente utilizzare questa conoscenza durante la fase di Query Processing, per minimizzare il numero di accesso ai dati.
Infatti, MOMIS, attraverso i passi appena descritti, arriverebbe alla formulazione di due query locali, da spedire rispettivamente alle classi `School_Member` and `University_Student` nelle sorgenti S_1 e S_2. Utilizzando però la conoscenza espressa dalla proprietà interschema definita dal progettista tra queste due classi, si può dedurre l'inutilità di spedire questa query alla classe `School_Member` in quanto, relativamente agli attributi specificati, i dati sono inclusi nell'insieme dei dati recuperabili da `University_Student`. Il risultato sarà dunque la generazione (e spedizione) dell'unica query locale alla classe `University_Student`.

3.5 Unificazione dei dati

Una volta ricevuti i dati richiesti dalle sorgenti, in risposta alle varie query locali a loro spedite, si pone il problema di come riorganizzare queste informazioni per presentarle all'utente.

Posto in altri termini, il problema è il riconoscimento automatico di tutte le informazioni che appartengono ad uno stesso *oggetto globale*. Dal punto di vista teorico, la soluzione ideale sarebbe la presenza di una *chiave universale*, o di un *oid* comune, condivisa da tutte le sorgenti (o almeno da tutte le sorgenti che fanno parte del sistema) che individui univocamente al loro interno tutti gli oggetti. È però impensabile che questo sia realmente realizzabile, dal momento che, anche nel migliore dei casi, questo identificatore sarebbe comunque valido e unico solo all'interno di un contesto limitato (si può per esempio prendere il codice fiscale, che perde però significato all'esterno di una determinata nazione...). Un'altra soluzione potrebbe essere il riunificare le risposte ricevute non sulla base di un oid universale, ma sulla base delle chiavi locali (quando dichiarate) delle singole sorgenti, ma il problema rimane in altri termini:

- supponendo che esista per esempio una chiave `nome` in tutte le tabelle interrogate, niente ci assicura che persone con nomi uguali siano effettivamente la stessa persona in contesti diversi;
- supponendo invece che come chiave interna delle tabelle si utilizzi un codice (pensiamo ad un esempio magazzino), il problema è più complicato: è molto probabile che si usino codici diversi per identificare lo stesso concetto, pur se il dominio degli attributi `codice` è uguale in tutte le sorgenti.

È quindi un problema aperto, la cui unica soluzione, fino a questo momento, sembra essere lo spostare questo tipo di decisione al progettista rimettendo a lui questa responsabilità. È allora ragionevole pensare che, congiuntamente alla definizione di proprietà interschema (in cui magari si definisce che due sorgenti hanno gli stessi dati, o dati parzialmente duplicati tra loro), sia compito del progettista indicare esplicitamente il campo (o l'insieme di campi) che dovrà essere utilizzato dal sistema per riunire le informazioni che fanno riferimento ad una unica entità.

Capitolo 4

Approccio Semantico all'Integrazione di Informazioni

L'approccio semantico seguito in MOMIS per l'integrazione di informazioni, consiste in una fase di *analisi ed estrazione* ed in un'altra di *unificazione*. La fase di analisi ed estrazione riguarda la costruzione di un Thesaurus comune di relazioni terminologiche basato sulle descrizioni degli schemi in ODL_{I^3}, e la formazione di clusters (usando delle tecniche di clusterizzazione) di classi ODL_{I^3} che descrivono informazioni simili in schemi differenti. La fase di unificazione, invece, costruisce, dall'integrazione di classi ODL_{I^3} contenute in un dato cluster, uno schema globale integrato per le sorgenti analizzate. L'approccio, come visto nella Sezione 3.3, si articola nei seguenti passi:

- *generazione di un Thesaurus comune*: l'obiettivo di questa fase è la costruzione di un Thesaurus di *relazioni terminologiche*, che si riferiscono a termini di classi appartenenti a sorgenti diverse;

- *analisi delle affinità delle classi*: le relazioni terminologiche memorizzate nel Thesaurus sono utilizzate per dare una valutazione delle *affinità* tra le classi di una stessa sorgente, o di sorgenti diverse. L'affinitá tra due classi é stabilita attraverso appropriati *coefficienti di affinitá*;

- *generazione dei clusters*: sulla base dei coefficienti di affinità, i concetti *più affini* tra loro vengono raggruppati all'interno di un *cluster*, attraverso l'uso di un algoritmo iterativo;

- *generazione dello schema globale del mediatore*: l'organizzazione delle classi all'interno di clusters porta direttamente alla creazione dello schema globale del mediatore. Per ogni cluster ottenuto viene definita una *classe globale*, che sarà rappresentativa di tutte le classi appartenenti al cluster.

Nel seguito di questo capitolo, saranno approfonditamente analizzati tutti i passi del processo che porta alla costruzione dello Schema Globale, cosí come presentati in [2, 3, 4].

4.1 Il linguaggio ODL$_{I^3}$

Il linguaggio ODL$_{I^3}$ è il linguaggio di definizione attraverso il quale i wrapper comunicano al mediatore (ed in particolare al Global Schema Builder) le descrizioni delle sorgenti da loro servite. Punto di partenza per la definizione di questo linguaggio, è stato l'attenersi alle raccomandazioni della proposta di standardizzazione per linguaggi di mediazione [21], risultato del lavoro di un workshop I^3. Parallelamente a questa proposta (secondo la quale i diversi sistemi di mediazione avrebbero potuto supportare sorgenti con modelli complessi, come quelli ad oggetti, e sorgenti molto più semplici, come file di strutture), si è cercato di discostarsi il meno possibile dal linguaggio ODL, a sua volta proposto dal gruppo di standardizzazione ODMG-93.
È stato quindi definito il linguaggio ODL$_{I^3}$ come una estensione allo standard ODL, per raggiungere i seguenti scopi:

- si ipotizzi che il nostro sistema di mediazione debba integrare database relazionali, ad oggetti, e file system;
- dichiarare relazioni terminologiche fra nomi di classi ed attributi;
- descrivere tutte le fonti di informazioni in maniera uniforme, come depositi di oggetti, e quindi utilizzare il concetto di *classe*, indipendentemente dal modello originale utilizzato. Sarà compito del wrapper provvedere alla traduzione dal modello originale di descrizione all'ODL$_{I^3}$, aggiungendo eventualmente in questa descrizione tutte le informazioni necessarie al mediatore (il nome e il tipo della sorgente,...);
- supportare la dichiarazione di regole di integrità (*if then rule*), siano esse definite sugli schemi locali (e magari da questi ricevute), siano esse riferite allo Schema Globale, e quindi inserite dal progettista del mediatore;
- supportare la dichiarazione di regole di mediazione, o *mapping rule*, utilizzate per meglio specificare l'accoppiamento tra i concetti globali e i concetti locali originali;
- consentire l'utilizzo di una *semantica di mondo aperto*, che permette alle classi descritte di cambiare formato (magari aggiungendo attributi agli oggetti) senza necessariamente cambiarne la descrizione;

- tradurre il linguaggio automaticamente nella logica descrittiva OCDL, e quindi utilizzarne le capacità nei controlli di consistenza nonché nell'ottimizzazione semantica delle interrogazioni.

Utilizzando questo linguaggio, sono fornite al mediatore le descrizioni di tutte le classi da integrare: da sottolineare che le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema, e quindi interrogabili. Non è quindi detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente, bensì ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa. Sia $S = \{S_1, S_2, \ldots, S_N\}$ un insieme di schemi di N sorgenti eterogenee che devono essere integrate. Come richiesto dall'ODL$_{I^3}$, ogni schema sorgente S_i è composto da un insieme di *classi*: una classe $c_{ji} \in S_i$ è caratterizzata da un nome e da un insieme di attributi, $c_{ji} = \langle n_{c_{ji}}, A(c_{ji})\rangle$. A sua volta, ogni attributo $a_h \in A(c_{ji})$, con $h = 1, \ldots, n$, è definito da una coppia $a_h = \langle n_h, d_h\rangle$, dove n_h è il nome e d_h è il dominio associato ad a_h.
Mostriamo di seguito un esempio di descrizioni di classi ODL$_{I^3}$:

```
interface School_Member
( source relational University
  extent School_Member
  key name )
{ attribute string first_name;
  attribute string last_name;
  attribute string faculty;
  attribute integer year; };
```

```
interface Student : CS_Person
( source object Computer_Science
  extent Students )
{ attribute integer year
  attribute set<Course> takes;
  attribute string rank; };
```

La nuova definizione aggiunge, dopo il nome della classe, le informazioni sul tipo e sul nome della sorgente di origine, sulla sua estensione, sugli attributi chiave, ed eventualmente sulle foreign key. Nell'esempio sono riportate una classe `Student` derivata da una sorgente ad oggetti (`Computer_Science`) ed una classe `School_Member` proveniente da una sorgente relazionale, rappresentata quindi originariamente da una tabella di tuple, con attributo chiave `name`.
La sintassi del linguaggio ODL$_{I^3}$ è riportata, in BNF, nell'Appendice B.

4.2 Esempio di riferimento

Questo esempio verrá utilizzato nel seguito per meglio chiarire tutti i passaggi effettuati durante il processo di integrazione, e fa riferimento alle definizioni degli schemi delle sorgenti presentati nell'Appendice C in ODL$_{I^3}$. Per semplicitá esso é rappresemtato in modo schematico in Figura 4.1.

L'esempio si riferisce ad una realtà universitaria: le sorgenti da integrare sono tre. La prima, `University` (S_1), è un database di tipo relazionale,

Sorgente University (S_1)

```
Research_Staff(first_name,last_name,relation,email,
               dept_code,section_code)
School_Member(first_name,last_name,faculty,year)
Department(dept_name,dept_code,budget,dept_area)
Section(section_name,section_code,length,room_code)
Room(room_code,seats_number,notes)
```

Sorgente Computer_Science (S_2)

```
CS_Person(name)
Professor:CS_Person(title,belongs_to:Division,rank)
Student:CS_Person(year,takes:set⟨Course⟩,rank)
Division(description,address:Location,fund,sector,employee_nr)
Location(city,street,number,county)
Course(course_name,taught_by:Professor)
```

Sorgente Tax_Position (S_3)

```
University_Student(name,student_code,faculty_name,tax_fee)
```

Figura 4.1: Esempio di riferimento

che contiene informazioni sullo staff e sugli studenti di una determinata università. È composta da cinque tabelle: `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Per ogni professore (presente nella tabella `Research_Staff`), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key `dept_code`), sul suo indirizzo di posta elettronica (`email`), e sul corso da lui tenuto (`section_code`). Per il corso, viene memorizzata pure l'aula (`Room`) dove questo si svolge, mentre del dipartimento sono descritti, oltre al nome (`dept_name`) ed al codice (`dept_code`), il budget (`budget`) che ha a disposizione e l'area (`dept_area`) a cui appartiene, sia essa Scientifica, Economica,... Per gli studenti presenti nella tabella `School_Member` sono invece mantenuti il nome (nella coppia `first_name` e `last_name`), la facoltà di appartenenza (`faculty`) e l'anno di corso (`year`). La sorgente `Computer_Science` (S_2) contiene invece informazioni sulle persone afferenti a questa facoltà, ed è un database ad oggetti. Sono presenti sei classi: `CS_Person`, `Professor`, `Student`, `Division`, `Location` e `Course`. I dati mantenuti sono comunque abbastanza simili a quelli della sorgente S_1: per quanto riguarda i professori, sono memorizzati il titolo (`title`), e la divisione di appartenenza (`belongs_to`), che a sua volta fa parte di un dipartimento (e ne può quindi essere considerata una specializzazione); per gli studenti sono memorizzati i corsi seguiti (`takes`) e l'anno di corso (`year`). Il corso ha poi un attributo

complesso che lo lega al professore che ne è titolare (`taught_by`), mentre per la divisione si tiene l'indirizzo (`address`), i fondi (`fund`) e il numero di impiegati (`employee_nr`).
È presente inoltre una terza sorgente, `Tax_Position` (S_3), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (`tax_fee`). In questo caso (S_3), non si tratta di un database ma di un file system, che contiene quindi semplici tracciati record.
Si è inoltre ipotizzato che, per identificare in modo univoco all'interno del mediatore un nome (sia esso di classe, sia esso di attributo), sia rispettivamente necessaria la coppia *nome_sorgente.nome_classe* e *nome_sorgente.nome_classe.nome_attributo*. Per semplicità nel nostro esempio, in assenza di ambiguità, sarà usato, per identificare un concetto, solo il nome locale (quando esso è univoco, cioé presente in un'unica sorgente, o quando, pur appartenendo a più sorgenti, denota in tutte lo stesso concetto). Si parlerà inoltre genericamente di *termine* t_i indicando con esso un nome di classe o di attributo.

4.3 Generazione del Thesaurus comune

Lo scopo di questa fase è la costruzione di un Thesaurus di *relazioni terminologiche* che rappresenti la conoscenza a disposizione sulle classi da integrare (ovvero sui nomi delle classi, sugli attributi) e che sarà la base per il calcolo di affinità tra le classi stesse. A questo fine, si possono definire, all'interno del Thesaurus, le seguenti tipologie di relazioni:

- SYN (synonym-of): definita tra due termini t_i e t_j, con $t_i \neq t_j$, che sono considerati sinonimi, ovvero che possono essere interscambiati nelle sorgenti, identificando lo stesso concetto del mondo reale. Un esempio di relazione SYN nel nostro caso è $\langle$`Section` SYN `Course`$\rangle$.

- BT (broader-term): definita tra due termini t_i e t_j tali che t_i ha un significato più ampio, più generale di t_j. Un caso di BT, nell'esempio universitario, può essere $\langle$`CS_Person` BT `Student`$\rangle$.

- NT (narrower-term): concettualmente è la stessa relazione espressa con una BT, intesa dall'altro punto di vista, dunque t_i BT $t_j \rightarrow t_j$ NT t_i. Lo stesso esempio potrebbe infatti essere $\langle$`Student` NT `CS_Person`$\rangle$.

- RT (related-term): definita tra due termini t_i e t_j che sono generalmente usati nello stesso contesto, come il caso in cui t_i risulta in associazione con t_j. Per esempio, possiamo avere la seguente $\langle$`Student` RT `Course`$\rangle$.

La scoperta di relazioni terminologiche presenti all'interno degli schemi è un processo semi-automatico, caratterizzato dalla interazione tra il progettista del sistema e gli ODB-Tools. Lo sforzo fatto in questa fase è stato diretto a limitare il più possibile l'intervento dell'operatore, al fine di aumentare la porzione di definizione del Thesaurus realizzabile in modo realmente automatico. L'intero processo che porta, partendo dalle descrizioni degli schemi in ODL_{I^3}, alla definizione del Thesaurus comune è riportato in Figura 4.2, e si articola in quattro passi.

1. **Estrazione automatica di relazioni dagli schemi ODL_{I^3}**
   Sfruttando le informazioni semantiche presenti negli schemi strutturati (sia basati sui modelli ad oggetti, sia relazionali) può essere identificato in modo automatico un insieme di relazioni terminologiche. In particolare, durante questa preliminare fase di analisi, si è in grado di estrarre relazioni BT/NT tra classi derivate dalle gerarchie di generalizzazione, e relazioni RT ottenute dalle gerarchie di aggregazione. Inoltre, sono estratte altre relazioni RT dagli schemi sorgenti per rappresentare l'aggregazione tra una classe ed ognuno dei sui attributi.
   Nel caso di schemi relazionali, le relazioni RT sono estratte anche dalle foreign keys.

   Esempio 1 Si considerino le sorgenti S_1 e S_2 riportate in Figura 4.1. Le relazioni terminologiche automaticamente estratte sono le seguenti:

   ⟨`Professor` NT `CS_Person`⟩
   ⟨`Student` NT `CS_Person`⟩
   ⟨`Professor` RT `Division`⟩
   ⟨`Student` RT `Course`⟩
   ⟨`Division` RT `Location`⟩
   ⟨`Course` RT `Professor`⟩
   ⟨`Research_Staff` RT `Department`⟩
   ⟨`Research_Staff` RT `Section`⟩
   ⟨`Section` RT `Room`⟩

2. **Revisione/Integrazione delle relazioni**
   Interagendo con il modulo, il progettista deve inserire tutte le relazioni terminologiche che non sono state estratte nel passo precedente, ma che devono comunque essere presenti per pervenire ad una esatta integrazione delle sorgenti. In particolare, possono essere inserite relazioni che coinvolgono classi appartenenti a schemi diversi, come pure relazioni che si riferiscono a nomi di attributi. Da sottolineare che, durante questa fase,

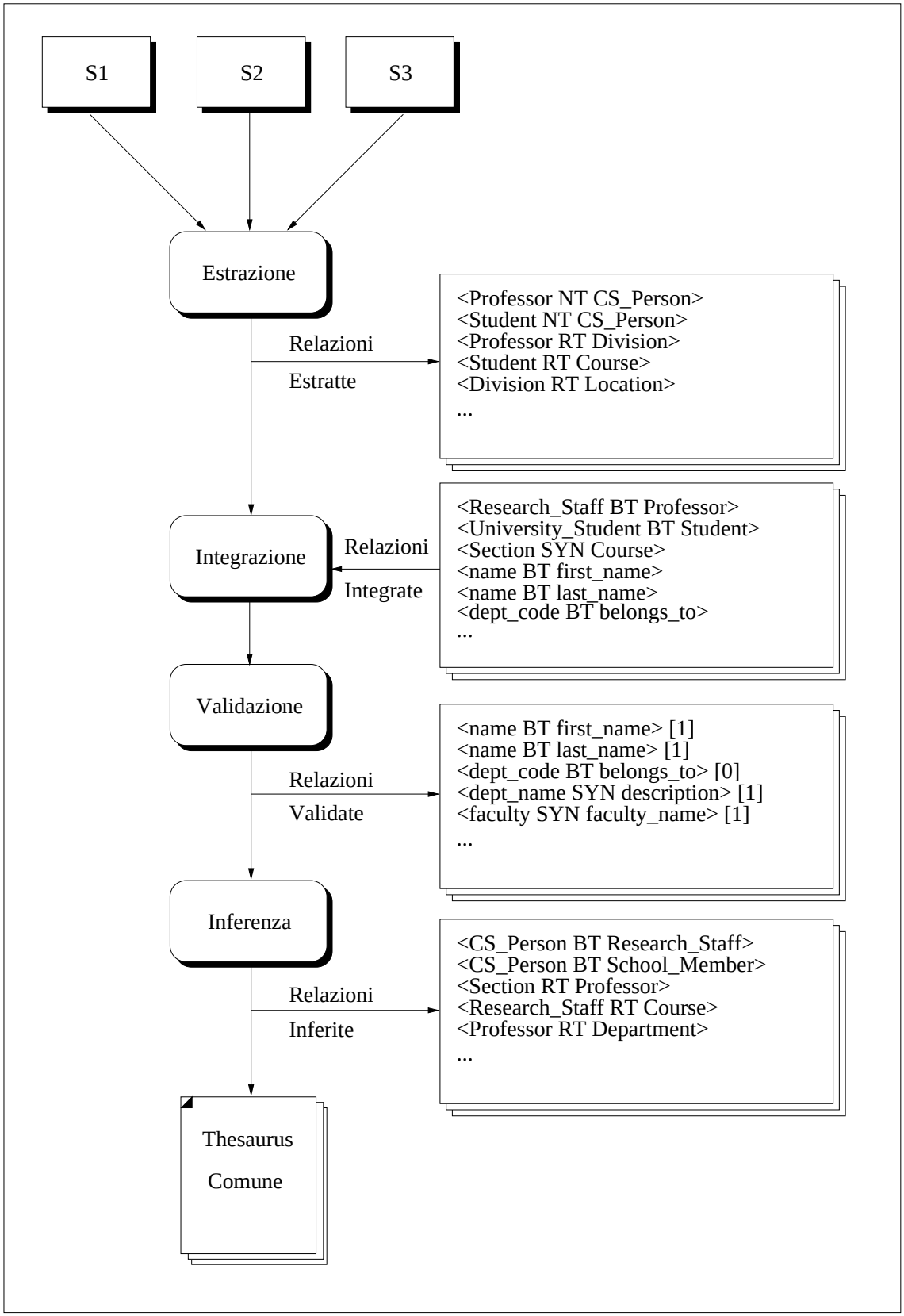


Figura 4.2: Il processo di generazione del Thesaurus comune

tutte le relazioni inserite dal progettista hanno carattere esclusivamente *terminologico*, a priori quindi da qualunque considerazione sulle estensioni. È quindi una fase che, nei prossimi sviluppi del progetto, potrebbe facilmente essere coadiuvata dall'uso di un dizionario in linea che evidenzi eventuali termini sinonimi, o correlati tra loro.

Esempio 2 Nelle sorgenti prese ad esempio, il progettista provvederà ad inserire le seguenti relazioni, che coinvolgono sia nomi di classi, sia nomi di attributi:

⟨`Research_Staff` BT `Professor`⟩
⟨`School_Member` BT `Student`⟩
⟨`University_Student` BT `Student`⟩
⟨`Department` BT `Division`⟩
⟨`Section` SYN `Course`⟩
⟨`name` BT `first_name`⟩
⟨`name` BT `last_name`⟩
⟨`dept_code` BT `belongs_to`⟩
⟨`dept_name` SYN `description`⟩
⟨`section_name` SYN `course_name`⟩
⟨`faculty` SYN `faculty_name`⟩
⟨`fund` SYN `budget`⟩
⟨`dept_area` SYN `sector`⟩

3. **Validazione delle relazioni**
   Come è stato precedentemente sottolineato, le relazioni inserite del progettista si basano esclusivamente sull'analisi dei termini che identificano un concetto. È stata quindi inserita a questo livello una fase di *validazione* delle relazioni definite tra attributi, che verifichi se anche i domini di questi attributi possano essere compatibili con il tipo di relazione definita tra loro. In questo modo le relazioni terminologiche sono distinte in *valide* ed *invalide*. In particolare, siano $a_t = \langle n_t, d_t \rangle$ e $a_q = \langle n_q, d_q \rangle$ i due attributi coinvolti in una relazione, utilizzando gli ODB-Tools, verranno eseguiti i seguenti controlli:

   - $\langle n_t$ SYN $n_q \rangle$: la relazione è considerata *valida* se d_t e d_q sono equivalenti, o se uno tra i due è più specializzato dell'altro;
   - $\langle n_t$ BT $n_q \rangle$: la relazione è considerata *valida* se d_t contiene o è equivalente a d_q;
   - $\langle n_t$ NT $n_q \rangle$: la relazione è considerata *valida* se d_t è equivalente o è contenuto in d_q.

In questa fase, tutti i controlli di equivalenza o di contenimento tra i tipi dei domini sono realizzati utilizzando l'algoritmo di sussunzione: ad esempio, d_t contiene d_q se il tipo del secondo è sussunto dal tipo del primo.

Esempio 3 Facendo riferimento al Thesaurus definito nell'esempio 2, si riporta di seguito l'output di questa fase: ad ogni relazione, definita dal progettista, che coinvolga due attributi viene affiancato un flag di controllo che riporta l'esito della validazione. Il valore [1] denota le relazioni valide e [0] quelle invalide.

⟨`name` BT `first_name`⟩	[1]
⟨`name` BT `last_name`⟩	[1]
⟨`dept_code` BT `belongs_to`⟩	[0]
⟨`dept_name` SYN `description`⟩	[1]
⟨`section_name` SYN `course_name`⟩	[1]
⟨`faculty` SYN `faculty_name`⟩	[1]
⟨`fund` SYN `budget`⟩	[1]
⟨`dept_area` SYN `sector`⟩	[1]

4. **Deduzione automatica di nuove relazioni**
   Sulla base delle relazioni inserite dal progettista, attraverso tecniche di inferenza, viene generato un insieme di nuove relazioni, che contribuiscano ad *accoppiare* sempre più le classi che, pur appartenendo a schemi diversi, condividono strutture simili (ovvero insiemi di attributi semanticamente simili). Per fare questo, viene generato, per ogni sorgente locale, un nuovo schema (partendo naturalmente dalla descrizione originale) che tenga in qualche modo conto di tutte le informazioni che il progettista ha inserito. In particolare, sono eseguite le seguenti modifiche:

   - classi: sono modificate in modo da conformarsi alle informazioni aggiunte attraverso le relazioni. Sono quindi aggiunte relazioni di ereditarietà negli schemi (per rappresentare BT o NT non presenti originariamente), relazioni di aggregazione (per rappresentare RT tra classi);
   - attributi: tutti gli attributi coinvolti in relazioni validate (che cioé abbiano superato positivamente la fase di controllo dei domini) sono riorganizzati in gerarchie (strutturate sul modello di alberi gerarchici). Per ogni attributo sono mantenuti gli attributi equivalenti (identificati da nomi definiti sinonimi), quelli più specializzati e quelli più generali. Al termine di questa riorganizzazione in gerarchie di termini, ogni attributo è sostituito con il termine più generale della gerarchia di appartenenza.

Relazioni Esplicite (Step 1,2)	Relazioni Inferite (Step 4)
⟨CS_Person BT Student⟩	⟨CS_Person BT Research_Staff⟩
⟨CS_Person BT Professor⟩	⟨CS_Person BT School_Member⟩
⟨School_Member BT Student⟩	⟨Section RT Professor⟩
⟨Research_Staff BT Professor⟩	⟨Research_Staff RT Course⟩
⟨Section SYN Course⟩	⟨Professor RT Department⟩
⟨Department BT Division⟩	⟨Professor RT Section⟩
⟨Student RT Course⟩	⟨Course RT Room⟩
⟨Course RT Professor⟩	⟨Student RT Section⟩
⟨Research_Staff RT Department⟩	⟨CS_Person BT University_Student⟩
⟨Research_Staff RT Section⟩	
⟨Professor RT Division⟩	
⟨Division RT Location⟩	
⟨University_Student BT Student⟩	
⟨Section RT Room⟩	

Figura 4.3: Relazioni Esplicite ed Inferite

Una volta ottenuti i nuovi schemi modificati, sono riutilizzati gli ODB-Tools per inferire ulteriori relazioni presenti in questi schemi, che concorrano alla formazione del Thesaurus comune.

Esempio 4 In Figura 4.3 sono riportate tutte le nuove relazioni inferite, messe a confronto con quelle definite esplicitamente negli schemi (includendo in quelle definite esplicitamente sia quelle inserite manualmente dal progettista, sia quelle dedotte automaticamente durante la fase preliminare di analisi). In Figura 4.4 sono invece riportate tutte le classi di tutte le sorgenti, così come risultano riorganizzate alla fine della fase di generazione del Thesaurus comune, facendo quindi riferimento alle classi modificate. È quindi una rappresentazione grafica delle relazioni che sussistono tra queste: le linee tratteggiate mettono in evidenza le relazioni inferite, le linee unite rappresentano invece quelle esplicite; le linee dotate di frecce rappresentano inoltre relazioni di generalizzazione, mentre quelle in cui le frecce sono assenti denotano le relazioni di aggregazione.

4.4 Analisi di Affinità delle classi ODL$_{I^3}$

In questa sezione, verrá presentato un approccio per valutare il livello di affinitá tra classi ODL$_{I^3}$ che descrivono lo stesso pezzo di informazione in schemi differenti. L'approccio é basato sulle relazioni terminologiche immagazzinate nel

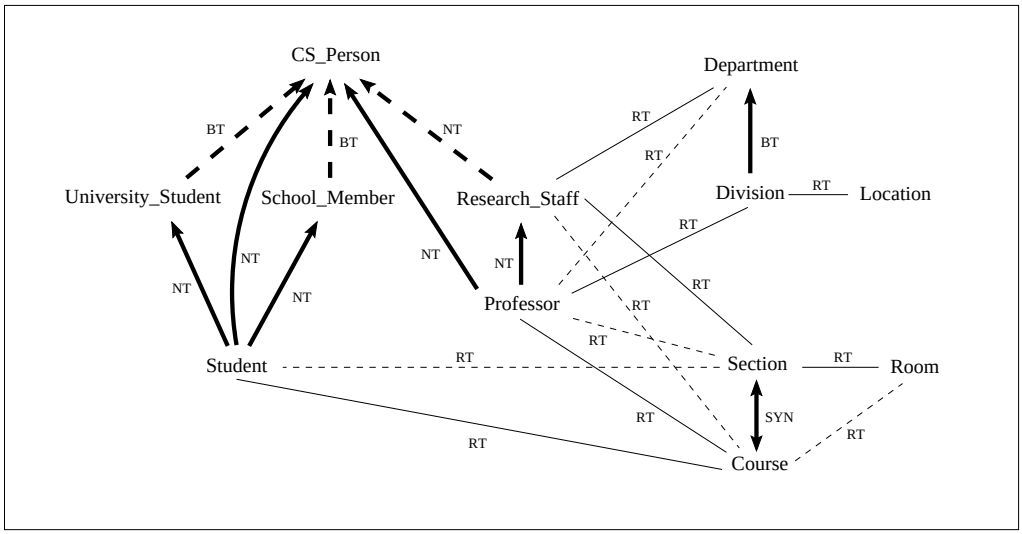


Figura 4.4: Thesaurus comune per le sorgenti $\mathbf{S_1}$, $\mathbf{S_2}$, e $\mathbf{S_3}$.

Thesaurus Comune e sul confronto dei nomi e degli attributi della classi.
Altri approcci proposti in letteratura si basano, invece, sull'analisi terminologica degli elementi dello schema e sulla definizione di corrispondenza tra attributi per identificare e risolvere l'inconsistenza semantica tra le diverse sorgenti.
Nel nostro approccio le classi ODL$_{I^3}$ sono analizzate e raffrontate attraverso i *coefficienti di affinità*, che ci permettono di determinare il loro livello di *similarità*. In particolare, il livello di affinitá di due classi c_{ji} e c_{hk} apparteneti rispettivamente alle sorgenti S_i S_k, é definito analizzando le relazioni che esistono tra i loro nomi (attraverso il *Name Affinity Coefficient*) e tra i loro attributi (per mezzo dello *Structural affinity Coefficient*), per arrivare ad un valore globale denominato *Global Affinity Coefficient*.
A questo scopo, il Thesaurus viene organizzato in una struttura simile alle Associative Networks [22], dove i nodi (ciascuno dei quali rappresenta genericamente un termine, sia esso il nome di una classe o il nome di un attributo) sono uniti attraverso relazioni terminologiche. A loro volta, tutte le relazioni presenti in questa rete sono percorribili in entrambi i sensi (dunque anche le BT e NT): due termini sono quindi affini se esiste un percorso che li unisce, formato da qualsivoglia relazioni. Per dare una valutazione numerica della affinità tra due termini, a ogni tipo di relazione viene associato un peso (denominato *strength* e denotato da $\sigma_{\Re}$), che sarà tanto maggiore quanto più questo tipo di relazione contribuisce a legare due termini (sarà quindi $\sigma_{syn} \geq \sigma_{bt/nt} \geq \sigma_{rt}$). In questa sezione si userà $\sigma_{ij_{\Re}}$ per denotare il peso della relazione terminologica $\Re$ definita tra i termini t_i e t_j. Nel nostro esempio si è adottato $\sigma_{syn} = 1$, $\sigma_{bt} = \sigma_{nt} = 0.8$ e $\sigma_{rt} = 0.5$. Quindi il

livello di affinitá di due termini dipende dalla lunghezza del percorso, dal tipo di relazioni terminologiche coinvolte in tale percorso e dal loro peso, in accordo con la *funzione di affinitá* di seguito definita.

Definizione 4 (Funzione di Affinità) Presi due termini, t_i e t_j, possono essere presenti nel Thesaurus zero o più cammini che li uniscono, formati da relazioni. Ad ognuno di questi cammini corrisponde naturalmente un valore, dato dal prodotto dei pesi delle relazioni in esso coinvolte. La Funzione di Affinità $A_{thes}(t_i, t_j)$ tra due termini, t_i e t_j, restituisce il valore maggiore tra questi, corrispondente al cammino più *stringente*, che unisce questi termini (che non sempre coincide col cammino più breve), definito come segue:

$$A_{thes}(t_i, t_j) = \begin{cases} 1 & \text{se } t_i = t_j \\ \sigma_{i1_{\Re}} \cdot \sigma_{12_{\Re}} \cdot \ldots \cdot \sigma_{(k-1)j_{\Re}} & \text{se } t_i \rightarrow^k t_j \\ 0 & \text{in tutti gli altri casi} \end{cases}$$

dove la notazione $t_i \rightarrow^k t_j$ denota appunto il più *stringente* tra questi cammini di lunghezza k, con $k \geq 1$, tra t_i e t_j nel Thesaurus.

Il livello di Affinità tra termini dipende quindi dalla lunghezza del cammino che li unisce, ma pure dal tipo delle relazioni coinvolte in questo cammino (e quindi dal loro peso). Per ogni coppia di termini, sarà necessariamente $A_{thes} \in [0, 1]$. La Affinità tra due temini sarà 0 se non esiste alcun cammino che li unisca, 1 se i due termini coincidono.

Esempio 5 Si consideri il Thesaurus illustrato in Figura 4.4. Si analizzino due cammini tra le classi `Student` e `Professor`, ovvero:

- `Student` $\rightarrow^{nt}$ `CS_Person` $\rightarrow^{nt}$ `Research_Staff` $\rightarrow^{nt}$ `Professor`
- `Student` $\rightarrow^{rt}$ `Section` $\rightarrow^{rt}$ `Professor`

Da questo si deduce che $A_{thes}(\texttt{Student}, \texttt{Professor}) = 0.8 \cdot 0.8 \cdot 0.8 = 0.512$ e non 0.25 che é il cammino piú breve.

Definizione 5 (Termini Affini) Due termini t_i, t_j si dicono *affini*, e si denotano con $t_i \sim t_j$, se la loro Funzione di Affinità restituisce un valore maggiore o uguale ad un predefinito valore di soglia $\alpha > 0$, cioè:

$$t_i \sim t_j \leftrightarrow A_{thes}(t_i, t_j) \geq \alpha$$

Per esempio, si supponga $\alpha = 0.4$.
In questo caso, dal momento che $A_{thes}(\texttt{Student}, \texttt{Professor}) = 0.512$, possiamo concludere che `Student` $\sim$ `Professor`.

4.4.1 Coefficienti di Affinità

In questo paragrafo, vengono date le definizioni dei coefficienti *Name Affinity Coefficient, Structural Affinity Coefficient* e *Global Affinity Coefficient* facendo riferimento a due classi ODL_{I^3} c e c' appartenenti rispettivamente alle sorgenti S e S'.

Definizione 6 (Name Affinity Coefficient) Misura la affinità di due classi calcolata rispetto ai loro nomi. Il *Name Affinity Coefficient* di due classi c e c' denotato da $NA(c,c')$, è la misura della affinità tra i loro nomi, n_c e $n_{c'}$, calcolata come segue:

$$NA(c,c') = \begin{cases} A_{thes}(n_c, n_{c'}) & \text{se } n_c \sim n_{c'} \\ 0 & \text{in tutti gli altri casi} \end{cases}$$

Esempio 6 Si considerino le classi `Student` e `Professor`. Si avrà che $NA(\texttt{Student}, \texttt{Professor}) = 0.512$

Definizione 7 (Structural Affinity coefficient) Lo Structural Affinity Coefficient di due classi c e c', scritto $SA(c,c')$, è la misura dell'affinità dei loro attributi, calcolata come segue:

$$SA(c,c') = \frac{2 \cdot \mid \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), n_t \sim n_q\} \mid}{\mid A(c) \mid + \mid A(c') \mid} \cdot F_c$$

$$F_c = \frac{|\{x \in C \mid flag(x)=1\}|}{|C|}$$

$$\text{C} = \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), \langle a_t \text{ SYN } a_q \rangle \text{ or } \langle a_t \text{ BT } a_q \rangle \text{ or } \langle a_t \text{ NT } a_q \rangle\}$$

dove C è l'insieme delle coppie di attributi validabili (ovvero delle coppie coinvolte in relazioni che possono essere validate attraverso un controllo sui domini) e $flag(x) = 1$ sta per un risultato positivo della suddetta validazione.

Lo Structural Affinity Coefficient è valutato utilizzando la funzione di Dice, e raffinato da un fattore di controllo F_c, e restituisce un valore compreso nell'intervallo [0,1] proporzionale al numero di attributi *affini* tra le classi considerate.
Rispetto alla definizione originaria di SA precedentemente sviluppata (si veda [16]), è stata proposta l'estensione realizzata dal fattore di controllo F_c. Il termine F_c realizza un controllo sui domini degli attributi coinvolti nella relazione

da esaminare (il controllo è quello esposto nel terzo passo della Sezione 4.3), permettendo quindi di non limitare la computazione di questo coefficiente alla sola analisi dei nomi che identificano gli attributi (analisi terminologica) e di estenderla alla considerazione dei domini che caratterizzano questi attributi. In pratica, una relazione che coinvolge attributi viene pesata in modo maggiore o minore nel calcolo del coefficiente a seconda che questa relazione trovi o meno riscontro anche nei tipi dei domini, e non solo nei nomi degli attributi. Il termine F_c va quindi a rifinire il coefficiente SA, moltiplicando la prima parte di questo per un termine compreso tra 0 e 1: in particolare, F_c è il rapporto tra il numero di relazioni validabili memorizzate nel Thesaurus tra attributi delle due classi ($\mid C \mid$), e il numero di queste relazioni che sono state validate.
In questo modo, maggiore sarà il numero di attributi affini tra le due classi e il numero di controlli positivi, più alto risulterà il valore dello *Structural Affinity Coefficient*.

Esempio 7 Si considerino gli schemi delle sorgenti riportati in Figura 4.1. Per quanto riguarda le classi `University_Student` in S_3 e `School_Member` in S_1, abbiamo che:

- $\mid \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), n_t \sim n_q\} \mid = 3$
- A(c) = 4 e A(c') = 4
- C = 3
- $\mid \{x \in C \mid \ flag(x) = 1 \ \} \mid = 3$
- $F_c = 1$

si ha che $SA(\texttt{University\_Student}, \texttt{School\_Member}) = \frac{2 \cdot 3}{4+4} \cdot \frac{3}{3} = 0.75$.

Definizione 8 (Global Affinity Coefficient) Il Global Affinity Coefficient di due classi c e c', denotato da $GA(c, c')$, è la misura della loro affinità calcolata come la somma pesata degli *Name Affinity Coefficient* e *Structural Affinity Coefficient*:

$$GA(c, c') = \begin{cases} w_{NA} \cdot NA(c, c') + w_{SA} \cdot SA(c, c') & \text{se } NA(c, c') \neq 0 \\ 0 & \text{in tutti gli altri casi} \end{cases}$$

dove i pesi w_{NA} e w_{SA}, con $w_{NA}, w_{SA} \in [0, 1]$ e $w_{NA} + w_{SA} = 1$, sono stati introdotti per dare al progettista la possibilità di variare caso per caso l'importanza dovuta ad ognuno dei due coefficienti rispetto all'altro. Nel nostro esempio di riferimento, abbiamo considerato ugualmente rilevanti ai fini dell'integrazione i due coefficienti, ponendo quindi $w_{NA} = w_{SA} = 0.5$.

Procedure Hierarchical Clustering
/* **Input:** K classi da analizzare */

1. Calcola tutte le coppie di Global Affinity coefficients $GA(c, c')$.
2. Crea un cluster per ogni classe.
3. **Repeat**
   Scegli la coppia c_h, c_k di cluster correnti
   con i coefficienti di affinita' maggiori in M, $M[h, k] = max_{i,j} M[i, j]$
   Forma un nuovo cluster unendo c_h, c_k;
   Aggiorna M cancellando le righe e le colonne corrispondenti a c_h e c_k;
   Definisci una nuova riga e una nuova colonna per il nuovo cluster.

until rango di M e' maggiore di 1.
end procedure.

Figura 4.5: Procedura di clustering

Durante il calcolo di questo coefficiente globale, è comunque data implicitamente una maggiore rilevanza al *Name Affinity coefficient*, e quindi ai nomi delle classi stesse: per classi i cui nomi non hanno nulla in comune non è neppure valutata la affinità rispetto ai loro attributi, e conseguentemente il loro GA risulterà nullo.

Esempio 8 Il Global Affinity Coefficient delle classi `University_Student` e `School_Member`, sapendo che $NA(\texttt{University\_Student}, \texttt{School\_Member}) = 0.64$ e tenendo presente l'Esempio 11, è calcolato nel seguente modo:

$$GA(\texttt{University\_Student}, \texttt{School\_Member}) = 0.5 \cdot 0.64 + 0.5 \cdot 0.75 = 0.695$$

4.5 Generazione dei Cluster

Per l'identificazione degli insiemi di classi affini negli schemi considerati sono utilizzate, all'interno del mediatore, tecniche di clustering, attraverso le quali le classi sono automaticamente classificate in gruppi caratterizzati da differenti livelli di affinità, formando un albero.

Tale procedura di clustering (vedi Figura 4.5) funziona in questo modo. Prima di tutto, vengono calcolati i coefficienti di affinitá per tutte le possibile coppie di classi che devono essere analizzate, cioé:

$$K \cdot \frac{(K-1)}{2}$$

dove K é il numero totale di classi da analizzare. Questi coefficienti sono memorizzati in una matrice M di rango K. Ad ogni entrata $M[h, k]$ della matrice corrisponde un coefficiente di affinitá globale $GA()$ riferito alle classi c_h e c_k. La procedura di clustering è iterativa e comincia allocando ogni classe in un singolo cluster: successivamente, ad ogni iterazione, i due cluster tra i quali sussiste il $GA()$ di valore massimo nella matrice M sono uniti. M è così aggiornata dopo ogni operazione di fusione tra cluster, cancellando le righe e le colonne corrispondenti ai cluster unificati, e inserendo una nuova riga ed una nuova colonna che rappresenti il nuovo cluster determinato. Vengono quindi calcolati i coefficienti $GA()$ tra questo cluster aggiunto e tutti quelli già presenti nella matrice: in particolare, viene mantenuto il valore $GA()$ massimo tra i due che erano stati già calcolati tra i cluster rimossi ed il corrispondente cluster col quale si vuole determinare il nuovo valore del coefficiente globale. La procedura termina quando tutte le classi appartengono ad un unico cluster.

L'output di questa fase non è comunque il cluster finale, contenente tutte le classi: ben più importante è l'albero che si è definito attraverso questa procedura di clustering, riportato, sempre relativamente all'esempio universitario, in Figura 4.6. In questo albero, i nodi sono rappresentanti di tutte le classi che sono state analizzate: nodi contigui sono caratterizzati da alta affinità, nodi tra loro molto lontani rappresenteranno invece concetti differenti. In questo modo, scegliendo un valore di soglia di riferimento, si possono formare non un unico bensì un insieme di cluster, all'interno dei quali sono raggruppate tutte le classi (alla prima iterazione) o i cluster (nelle iterazioni successive) tra i quali esiste una affinità (rappresentata dal valore di GA) maggiore del valore soglia predefinito. Come mostrato in figura, abbiamo scelto come soglia il valore 0.5: tutte le classi di partenza delle sorgenti locali S_1, S_2 e S_3 sono state raggruppate iterativamente (attraverso la procedura esposta poco sopra) in cluster finali Cl_i.

4.6 Generazione dello Schema Globale

In questa sezione viene presentato il processo che porta, a partire dai cluster precedentemente determinati, alla definizione dello *Schema Globale* del Mediatore, ovvero della visione dei dati che sarà presentata all'utente in fase di Query Processing. In Figura 4.7 sono mostrati i passi, arricchiti dall'introduzione delle rule estensionali, che portano alla definizione dello Schema Globale.

La prima fase di questo processo viene realizzata automaticamente e genera, per ogni cluster, una *classe_globale*$_i$ rappresentativa di tutte le classi che fanno parte del cluster (ovvero una classe che costituisca una visione unificata di queste classi). Sia Cl_i un cluster determinato nella fase precedente: ad esso viene as-

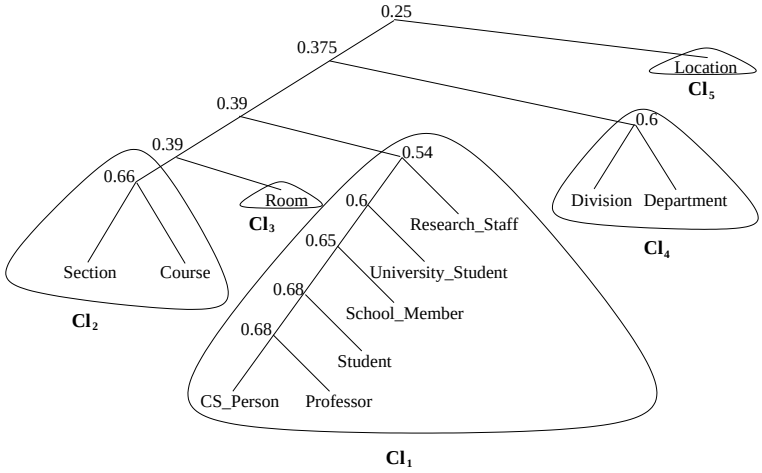


Figura 4.6: Albero di affinità

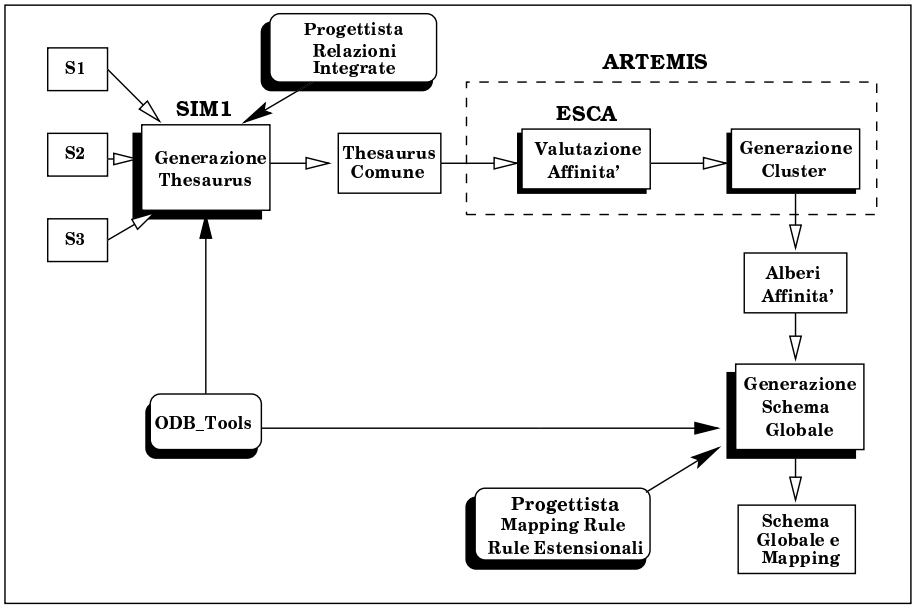


Figura 4.7: Fasi dell'Integrazione

sociata la *classe_globale*$_i$, alle quale corrisponderà quindi un insieme di attributi globali. Il processo di unificazione degli attributi è realizzabile in modo automatico, basandosi sulle relazioni tra attributi memorizzate nel Thesaurus (vedi Sezione 4.3 step 3) e seguendo i seguenti criteri:

- ad ogni *classe_globale*$_i$ è associata l'unione degli attributi di tutte le classi appartenenti al cluster Cl_i dal quale è stata generata;
- all'interno dell'unione degli attributi, per tutti quelli che hanno un'affinitá tra nomi dovuta alla relazione SYN, ne viene riportato solo uno tra essi (rimuovendo quindi tutti gli altri);
- all'interno dell'unione degli attributi sono identificati tutti gli insiemi di termini legati da relazioni di specializzazione (tra i quali erano quindi state definite relazioni di BT e NT) e vengono riorganizzati all'interno di gerarchie: per ognuna di queste gerarchie è mantenuto solamente il termine più generale (che quindi ne sta a capo e ne può essere considerato il rappresentante) mentre sono rimossi tutti gli altri.

Esempio 9 Riferendoci al cluster Cl_1 di Figura 4.6, viene definita automaticamente la seguente classe globale:

```
GC1 = (name, rank, title, dept_code, year, takes, relation,
       email, student_code, tax_fee, section_code, faculty)
```

in cui dall'insieme unione sono stati rimossi i termini `faculty_name` (perché sinonimo di `faculty`), `belongs_to` (perché esiste nell'unione il termine più generale `dept_code`) e la coppia `first_name` e `last_name` (perché più specializzati dell'attributo `name`, che basta quindi a rimpiazzarli entrambi).

Oltre a questa semplice unione ragionata degli attributi, in vista della successiva fase di Query Processing (da realizzare ogniqualvolta un utente sottopone una interrogazione al mediatore), è necessaria una fase di raffinamento delle informazioni presenti nello schema globale già determinato, raffinamento che necessariamente richiederà l'intervento del progettista del sistema. In particolare, MOMIS permette agevolmente di affrontare una vasta casistica di problematiche legate all'interrogazione di un mediatore rifinendo lo schema globale con le seguenti informazioni:

1. nome della *classe_globale*$_i$: il sistema propone un insieme di nomi candidati per la classe globale, sfruttando le relazioni terminologiche memorizzate nel Thesaurus ed i nomi delle classi appartenenti al cluster da cui questa classe globale deriva. Basandosi su questi suggerimenti, il progettista puó decidere

il nome piú adatto per *classe_globale*$_i$. Nel nostro esempio, riferendoci alla classe GC$_1$, il progettista decide di denominarla University_Person, essendo questa classe comprensiva di informazioni riguardanti sia studenti, sia professori di una data università;

2. *mapping* fra gli attributi globali ed i corrispondenti locali: mentre nei casi in cui un attributo globale (per esempio faculty) è correlato ad un singolo attributo locale (ad esempio faculty_name nella classe University_Student di S_3) non vi è bisogno di specificare alcuna informazione, in quanto il mapping è realizzato automaticamente dal sistema, nel caso in cui ad un singolo attributo globale corrisponde un insieme di attributi locali (ed è l'esempio di name che deve essere tradotto nella coppia first_name e last_name nella sorgente S_1) occorre specificare il *tipo* di questa corrispondenza, scegliendo tra le seguenti alternative:

   - corrispondenza in *and*: l'attributo globale corrisponde all'unione, in uno specificato ordine, degli attributi locali ad esso corrispondenti.
   - corrispondenza in *or*: l'attributo globale è equivalente ad ogni singolo attributo locale con cui è messo in corrispondenza, e deve quindi originare più interrogazioni contemporaneamente per quella classe in cui vi è questo tipo di corrispondenza;

3. valori di default: a volte è possibile specificare, per una classe appartenente alla classe globale, valori che in essa sono sempre verificati, relativamente ad un determinato attributo globale (grazie a conoscenze date a priori al progettista, o ad informazioni presenti nel nome della classi locali, come metadati). In questo caso, per agevolare una successiva ottimizzazione delle interrogazioni (che sarà analizzata nel prossimo paragrafo), MOMIS dà la possibilità di specificare esplicitamente questi valori.

   Esempio 10 Nella classe globale GC$_1$, il progettista è sicuramente in grado di specificare che l'attributo globale rank vale sempre Student nella classe School_Member, nonostante questa informazione non sia esplicitamente memorizzata in alcuna struttura, bensì sia presente come metadato nel nome stesso dalla classe. Ci si avvarrà di questa funzionalità di MOMIS per inserire l'informazione;

4. nuovi attributi: viene mantenuta nel sistema la possibilità di inserire a livello globale nuovi attributi, che dovranno però essere manualmente correlati ad attributi locali, o settati con valori di default.

```
interface University_Person
(extent Research_Staffers, School_Members, CS_Person
        Professors, Students, University_Students
 key    name)
{ attribute string name
    mapping_rule (University.Research_Staff.first_name and
                  University.Research_Staff.last_name)
                  (University.School_Member.first_name and
                  University.School_Member.last_name),
                  Computer_Science.CS_Person.name
                  Computer_Science.Professor.name
                  Computer_Science.Student.name
                  Tax_Position.University_Student.name;
 attribute string rank
    mapping_rule University.Research_Staff = 'Professor',
                 University.School_Member = 'Student',
    ... }
```

Figura 4.8: Esempio di classe globale in ODL_{I^3}

Al fine di poter specificare in modo dichiarativo questo insieme di informazioni aggiuntive, è stata proposta in ODL_{I^3} una estensione alla classica definizione di classe attraverso il linguaggio ODL. Un esempio di descrizione di classe globale in ODL_{I^3} è dato nella firgura 4.8, in riferimento alla $\texttt{GC}_1$.

Come si può vedere, per ogni attributo, oltre alla specificazione del tipo e del nome, viene aggiunta una lista di *mapping rule*, attraverso le quali vengono specificate le informazioni su come questo attributo verrà accoppiato con attributi locali, come pure informazioni su valori di default o nulli (nel caso in cui la mapping rule di un attributo per una determinata classe appartenente alla classe globale non sia specificata). Per esempio, riferendoci sempre all'attributo `name`, sono specificati gli attributi che devono essere considerati per ogni classe appartenente al cluster di origine Cl_1. In questo caso, viene definita una corrispondenza di tipo *and* per la classe `University.Research_Staff` (si usa la *dot notation* per evidenziare al sorgente di origine).

In MOMIS, questa definizione di classe, e le informazioni in essa contenute, dà origine ad una struttura dati tabellare, definita *mapping table*, che sarà la base per tutto il processo di Query Processing. Come esempio, sono riportate in Figura 4.9 le mapping table dei cluster Cl_1 e Cl_4 di Figura 4.6, rappresentanti rispettivamente delle classi globali `University_Person` e `Workplace`.

Lo schema globale del Mediatore, quindi, é composto dalle classi globali definite per tutti i cluster dell'albero di affinitá. Una volta che tutte le classi globali

University_Person	**name**	**rank**	**works**	**faculty**
Research_Staff	first_name *and* last_name	'Professor'	dept_code	null
School_Member	first_name *and* last_name	'Student'	null	faculty
CS_Person	name	null	null	'Computer_Scienc
Professor	name	rank	belongs_to	'Computer_Scienc
Student	name	rank	null	'Computer_Scienc
University_Student	name	'Student'	null	faculty_name

Workplace	**name**	**area**	**employee_nr**	**budget**	**...**
Department	dept_name	dept_area	null	budget	...
Division	description	sector	employee_nr	fund	...

Figura 4.9: Mapping table di University_Person e Workplace

sono state definite seguendo questo processo, sono revisionate per controllare la loro mutua consistenza.

Capitolo 5

Il modulo software ESCA

In questo capitolo verrà presentato il modulo software **ESCA**, che definisce automaticamente il grado di affinitá delle classi delle sorgenti da integrare nel progetto MOMIS.

Insieme all'architettura del modulo, verranno presentate le procedure più interessanti utilizzate al suo interno, includendo inoltre la descrizione delle strutture dati e degli algoritmi più significativi realizzati. Il software è stato sviluppato in linguaggio JAVA, utilizzando l'ambiente di sviluppo JDK (Java Development Kit) Version 1.2 Beta 4, su una piattaforma hardware SUN Sparc 20.

5.1 Obiettivi ed Architettura del Modulo

Il modulo ESCA (**E**valuation of **S**chema **C**lass **A**ffinity), si pone come obiettivo la realizzazione della parte iniziale del modulo di integrazione denominato ARTEMIS-Tools, relativa alla fase di integrazione degli schemi delle sorgenti di informazione.
Il modulo software è riportato in Figura 5.1: a partire dal Thesaurus Comune che fornisce le relazioni terminologiche tra nomi di classi e di attributi, e utilizzando gli schemi delle sorgenti che devono essere integrate, forniti in ODL$_{I^3}$, ESCA porta alla valutazione automatica della similaritá delle diverse classi. Dovrà dunque interfacciarsi a valle, con un ulteriore modulo che, applicando le tecniche di clustering definite ed esposte nel Capitolo 4, calcoli i cluster veri e propri di classi che rappresenta la fase finale del processo di integrazione.
In Figura 5.2 é riportato il Data Flow Diagramm del modulo ESCA in cui sono rappresentate le fasi principali del software. In particolare, il processo complessivo si divide in tre passi principali:

1. **estrazione automatica di classi e relativi attributi dagli schemi ODL$_{I^3}$**: chiamata solo all'inizio del modulo, in questa fase

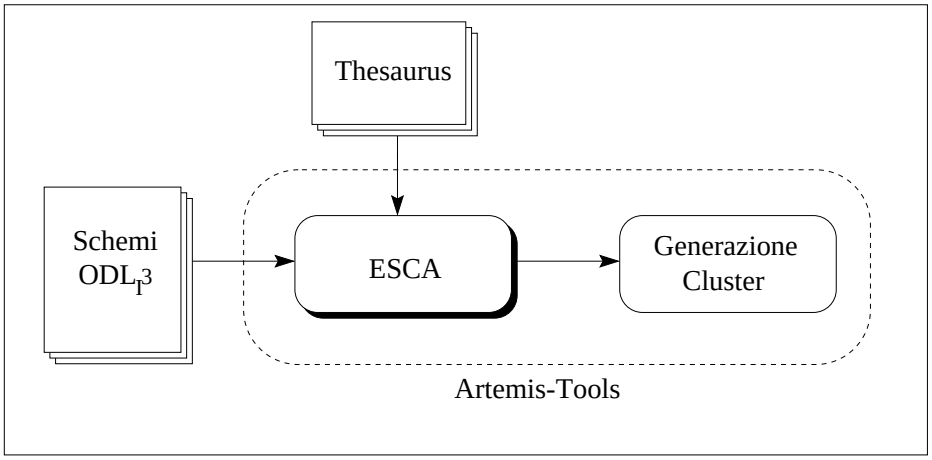


Figura 5.1: Il modulo ESCA

sono derivati, dagli schemi ODL$_{I^3}$, i nomi assoluti delle classi e dei relativi attributi ($nome\_sorgente.nome\_classe$ per le classi e $nome\_sorgente.nome\_classe.nome\_attributo$ per gli attributi);

2. **ricerca del massimo cammino (Max_Path) tra coppie di termini**: questa procedura é chiamata ogni volta che si deve definire l'affinitá tra le classi ma anche tra gli attributi. Il modulo, sulla base delle relazioni terminologiche contenute nel Thesaurus, data una coppia di termini definisce la relativa funzione di affinitá. Tale funzione restituisce il valore maggiore tra due termini t_i e t_j, corrispondente al cammino più *stringente* (detto anche massimo cammino), che unisce questi termini (che non sempre coincide col percorso più breve), definito come segue:

$$A_{thes}(t_i, t_j) = \begin{cases} 1 & \text{se } t_i = t_j \\ \sigma_{i1_{\Re}} \cdot \sigma_{12_{\Re}} \cdot \ldots \cdot \sigma_{(k-1)j_{\Re}} & \text{se } t_i \rightarrow^k t_j \\ 0 & \text{in tutti gli altri casi} \end{cases}$$

dove la notazione $t_i \rightarrow^k t_j$ denota appunto il più *stringente* tra questi cammini di lunghezza k, con $k \geq 1$, tra t_i e t_j nel Thesaurus, mentre con $\sigma_{ij_{\Re}}$ si denota il peso della relazione terminologica $\Re$ definita tra i termini in considerazione.

3. **Valutazione dei Coefficienti di affinitá**: utilizzando il valore della funzione di affinitá calcolata nella fase precedente, e sapendo che due termini t_i e t_j si dicono *affini* (denotato $t_i \sim t_j$) se:

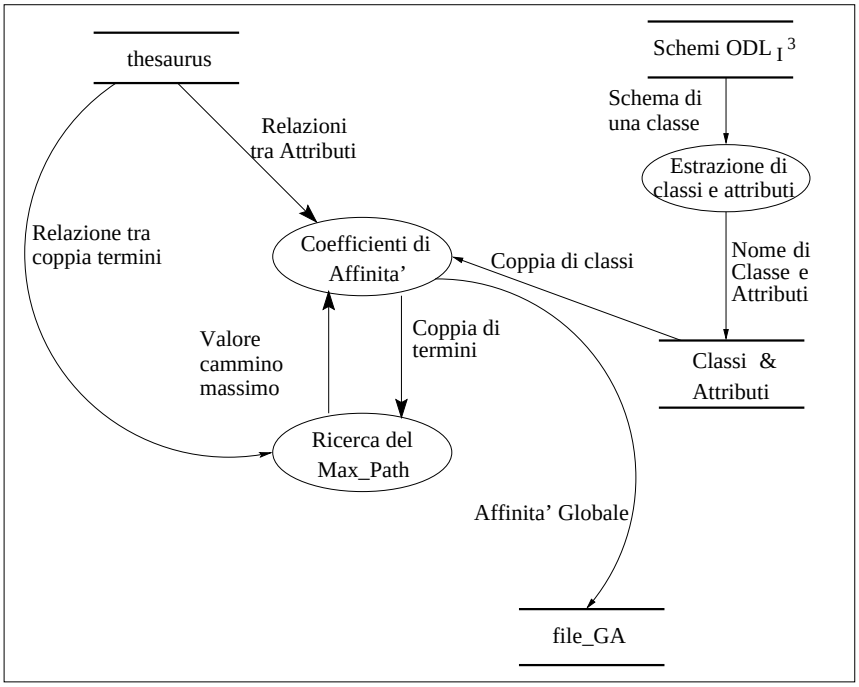


Figura 5.2: DFD del modulo ESCA

$$A_{thes}(t_i, t_j) \geq \alpha$$

il modulo definisce:

- *Name Affinity Coefficient*: il quale misura l'affinitá di due classi calcolata rispetto ai loro nomi come segue:

$$NA(c, c') = \begin{cases} A_{thes}(n_c, n_{c'}) & \text{se } n_c \sim n_{c'} \\ 0 & \text{in tutti gli altri casi} \end{cases}$$

- *Structural Affinity Coefficient*: che é la misura dell'affinitá rispetto agli attributi della coppia di classi in esame definito come segue:

$$SA(c, c') = \frac{2 \cdot \mid \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), n_t \sim n_q\} \mid}{\mid A(c) \mid + \mid A(c') \mid} \cdot F_c$$

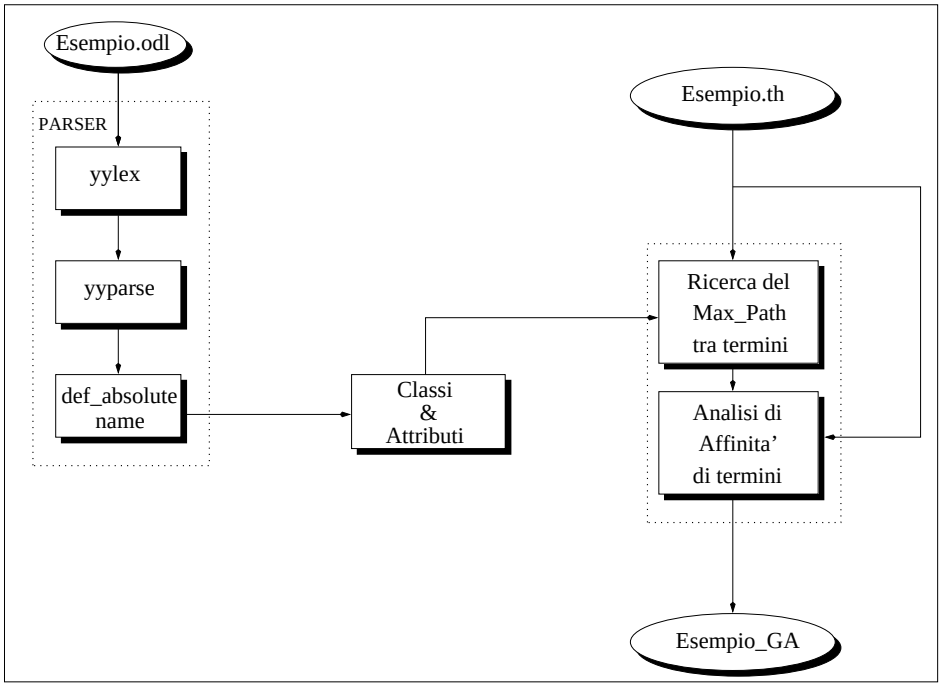


Figura 5.3: Architettura del modulo ESCA

$$F_c = \frac{|\{x \in C |\ flag(x)=1\ \}|}{|C|}$$

$$\text{C} = \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), \langle a_t \text{ SYN } a_q \rangle \text{ or } \langle a_t \text{ BT } a_q \rangle \text{ or } \langle a_t \text{ NT } a_q \rangle\}$$

dove C è l'insieme delle coppie di attributi validabili (ovvero delle coppie coinvolte in relazioni che possono essere validate attraverso un controllo sui domini) e $flag(x) = 1$ sta per un risultato positivo della suddetta validazione.

- *Global Affinity Coefficient*: é la somma pesata degli Name Coefficient Affinity e Structural Coefficient Affinity e rappresenta la misura dell'affinitá globale delle due classi in esame, calcolata come segue:

$$GA(c, c') = \begin{cases} w_{NA} \cdot NA(c, c') + w_{SA} \cdot SA(c, c') & \text{se } NA(c, c') \neq 0 \\ 0 & \text{negli altri casi} \end{cases}$$

Nelle sezioni seguenti saranno analizzate l'architettura, riportata in Figura 5.3, e gli algoritmi piú interessanti del modulo.

5.1.1 Estrazione automatica di classi e relativi attributi dagli schemi ODL$_{I^3}$

Il primo passo di questo processo è l'acquisizione degli schemi delle sorgenti, espressi attraverso il linguaggio ODL$_{I^3}$: per fare questo, è stato necessario utilizzare una estensione, in via di sviluppo in [24], al preesistente parser di ODL, nella quale sono state incluse le modifiche apportate a questo linguaggio descrittivo, per supportare le nuove esigenze di un ambiente di integrazione di informazioni. Il parser è stato realizzato attraverso uno strumento software, BYACC/JAVA (un generatore di parser Java ottenuto dall'estensione del generatore di parser C/C++ "Berkeley v 1.8 YACC", disponibile sul sito internet [23]), che facilita la scrittura di programmi in linguaggio JAVA per l'analisi e l'interpretazione di sequenze di caratteri che costituiscono un dato testo sorgente. In particolare, sono state utilizzate e realizzate le seguenti funzioni:

- **yylex**: effettua l'analisi lessicale del testo sorgente, riconoscendo nelle serie di caratteri ricevuti in input dei predefiniti *token* (ovvero delle espressioni regolari definite dal programmatore);
- **yyparse**: interpreta una sequenza di token e riconosce la sintassi definita nel file di input; in questo modo viene automaticamente realizzato un parser semplicemente definendo la sintassi da riconoscere, a sua volta espressa in una notazione molto simile alla *Bakus-Naur Form* (BNF).
- **def_absolute_name**: in base alla sintassi riconosciuta nel file di input, per ogni nome di classe o attributo definisce il nome assoluto.

Questa fase é stata preceduta da uno studio approfondito del parser in modo tale da riconoscere i punti in cui sono identificati sorgenti, classi e attributi, per poi definire e chiamare all'occorrenza la funzione *def_absolute_name*.

Esempio 11 Si consideri ad esempio la classe `Room`, così descritta in ODL$_{I^3}$:

```
interface Room
 ( source relational University
   extent Room
   key room_code )
{       attribute integer room_code;
        attribute integer seats_number;
        attribute string notes;  };
```

L'estensione ODL che individua il nome della sorgente, della classe e dei suoi attributi é la seguente:

```
.....

Interface:
        ForwardDcl
                    |
        InterfaceDcl
.....
```

fase di individuazione dei nomi delle ***classi***

```
.....

InterfaceDcl:
        INTERFACE
        Identifier

        COLON
        InheritanceSpec
        OptTypePropertyList
                         {def_absolute_name($1.sval,$2.sval);}
        OptPersistenceDcl
        LPAR
        OptInterfaceBodyUnion
        RPAR

                    |

        INTERFACE
        Identifier

        OptTypePropertyList
                        {def_absolute_name($1.sval,$2.sval);}
        OptPersistenceDcl
        LPAR
        OptInterfaceBodyUnion
        RPAR

......
```

fase di individuazione dei nomi delle ***sorgenti***

```
.....

 OptTypePropertyList:
```

```
        /* No type property list */
        |
        TypePropertyList

TypePropertyList:
        LRPAR
        OptSourceSpec
        OptExtentSpec
        OptKeySpec
        OptForKeySpec
        RRPAR

OptSourceSpec:
        /* No extent specifier */
        |
        SOURCE
        SourceType
        Identifier      /* source name */
           {
            def_absolute_name($1.sval,$3.sval);

           }
......
```

fase di individuazione dei nomi degli ***attributi***

```
.....

OptInterfaceBodyUnion:  /* No interface body */
        |
        InterfaceBodyUnion


InterfaceBodyUnion:
        InterfaceBody           /* Singola Interface body */
        |
        InterfaceBody  UNION  InterfaceBodyUnion


InterfaceBody:
        Export
                    |
        Export
        InterfaceBody
```

```
Export:
        TypeDcl       /*  Ignorata la TypeDef propria delle interfacce */
        SEMI          /*  I TypeDef sono riconosciuti solo globali  */
                    |
        ConstDcl      /*  Ignorata la ConstDef propria delle interfacce */
        SEMI          /*  Le costanti sono riconosciute solo globali  */
                    |
        ExceptDcl     /*  Ignorata la gestione delle Exceptions  */
        SEMI
                    |
        AttrDcl       /*  Dichiarazione di attributi  */
        SEMI
                    |
        RelDcl        /*  Dichiarazione di relationships  */
        SEMI
                    |
        OpDcl         /*  Dichiarazione di operations  */
        SEMI
                    |
        error
        SEMI


AttrDcl:
        OptReadonly
        ATTRIBUTE
        OptDomainType
        Identifier              /* Attribute name */
        OptFixedArraySize
        OptMappingRuleDcl
          {
             def_absolute_name($2.sval,$4.sval);
          }

......
```

dove la procedura *def_absolute_name(s1,s2)* é cosí definita:

```
void def_absolute_name(String s1,String s2)
{
   if (s1.compareTo("source")==0)
      source=s2;
   if (s1.compareTo("interface")==0)
```

```
      {
         source_class=source +"."+s2;
         //
         // salvo il nome della classe
         //
         cl=new Classe(source_class);
         classi.addElement(cl);
      }
   if (s1.compareTo("attribute")==0)
     {
         //
         // salvo il nome degli attributi
         //
        tot_name=source_class +"."+s2;
        cl.aggiungiAttributo(tot_name);
     }
}
```

La struttura dati che il parser provvede a memorizzare a seguito del riconoscimento di una classe e dei suoi attributi, sulla quale andranno ad agire tutte le procedure successive é definita come segue:

```
public class Classe {
   String nome;
   Vector attributi=new Vector();

   public int nAttributi() {
      return attributi.size();
   }

   public String toString() {
      return nome;
   }

   Classe (String n){
     nome=n;
   }

  void aggiungiAttributo(String n) {
     Attributo a = new Attributo(n);
     attributi.addElement(a);
  }
}
```

```
public class Attributo {
   String nome;
   public String toString() {
      return nome;
   }

   Attributo(String n) {
      nome = n;
   }
}
```

Dall'esame della classe ROOM (si ricordi che usiamo genericamente il termine classe anche per tabelle relazionali, al fine di uniformare la visione di tutte le entitá integrate) il sistema definisce e memorizza nella struttura dati precedentemente presentata, i seguenti nomi:

⟨University.Room⟩
⟨University.Room.room_code⟩
⟨University.Room.seats_number⟩
⟨University.Room.notes⟩

Lettura del file di Thesaurus

In questa fase é stato studiato un automa, riportato in Figura 5.4, per memorizzare le informazioni del Thesaurus in una struttura dati, al fine di renderle piú manipolabili. In particolare, il modulo riconosce il carattere "," come separatore tra nome1, peso, nome2 e flag di una relazione terminologica. Il carattere ";" é riconosciuto come fine riga, mentre un valore "> 0" definisce la fine del file di Thesaurus. Tale procedura, cosí come quella di *estrazione dei nomi delle classi e degli attributi*, é eseguita solo una volta, al momento della chiamata del modulo ESCA. In Figura 5.5 é rappresentato il modo in cui é stato strutturato il file di Thesaurus, dove **"SX"** é il vettore che racchiude tutti i nomi di sinistra, **"Rel"** contiene tutte relazioni terminologiche, **"DX"** i nomi di destra mentre **"Flag"** contiene il valore dei flag (0 oppure 1) per relazioni tra attributi, *null* per relazioni tra classi. Di seguito é riportata, invece, la struttura dati definita per memorizzare il Thesaurus:

```
public class Relazioni
{
  //
  //parte sinistra del file .th
  //
```

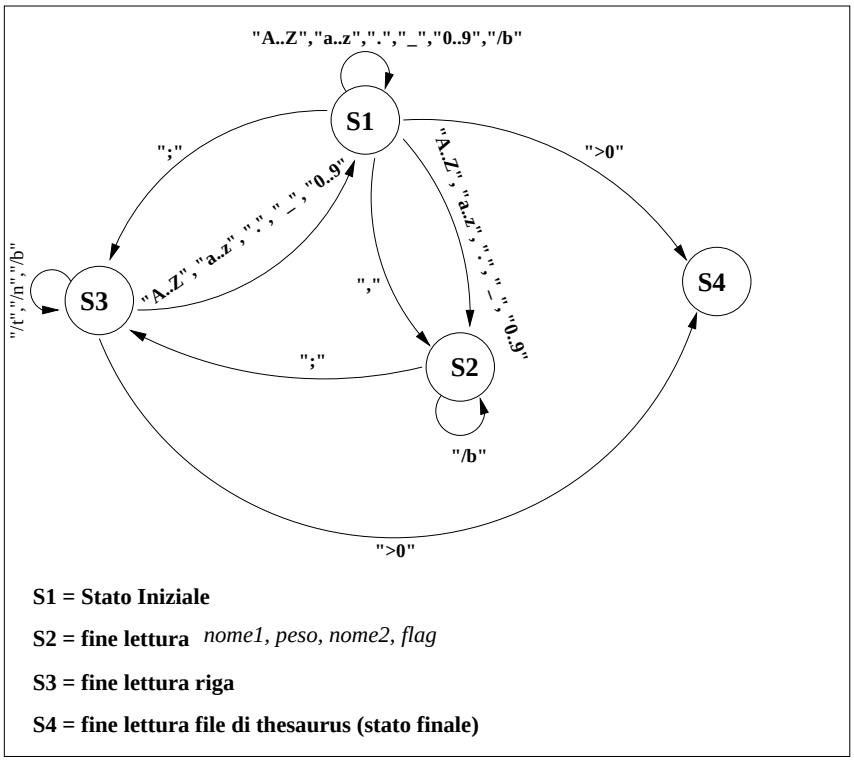


Figura 5.4: Automa di lettura del Thesaurus

```
 public static Vector sx=new Vector();

//
//parte destra del file .th
//

 public static Vector dx=new Vector();

//
//pesi delle relazioni del file .th
//

 public static Vector peso=new Vector();

//
//flag di controllo del dominio degli attributi
//
```

SX	Rel	DX	Flag
Student	nt	CS_Person	, ;
Professor	nt	CS_Person	, ;
Course	rt	Professor	, ;
Research_Staff.name	bt	CS_Person.first_name	, 1;
School_Member.name	bt	CS_Person.first_name	, 1;
Research_Staff.dep_code	bt	Professor.belongs_to	, 0;
...	...	...	...

Figura 5.5: Struttura del Thesaurus

```
   public static Vector flag=new Vector();
}
```

ci si aggiunge, inoltre, un vettore "**Peso**" che associa ad ogni tipo di relazione il valore (BT = NT = 0.8 , SYN = 1, RT = 1).

5.1.2 Ricerca del massimo cammino tra coppie di termini

Compito di questa fase é calcolare, data una coppia di termini, il valore della funzione di affinitá sulla base delle relazioni terminologiche contenute nel Thesaurus, come rappresenta il Data Flow Diagramm di Figura 5.6.

Il modulo si basa su una procedura di ricerca iterativa che esplora tutto il Thesaurus al fine di trovare il commino piú *stringente* che unisce i due termini considerati. Illustriamo il procedimento generale attraverso un esempio.

Esempio 12 Consideriamo, come esempio di riferimento, la Figura 5.7 in cui si é supposto di voler calcolare la funzione di affinitá tra `CS_Person` e `Office`.

Per rendere i tempi di risposta della procedura piú brevi, il modulo é stato arricchito di due fasi, illustrate dall'esempio, che ottimizzano la procedura di ricerca:

- *ordinamento*: prima di visitare il livello inferiore, viene effettuato un ordinamento di tutti i nodi figlio, in base al peso di ogni relazione, al fine di scegliere il nodo con peso maggiore in modo da aumentare la probabilitá di trovare il massimo cammino su quel ramo. Dopo varie prove si é stabilito di utilizzare un ordinamento di tipo QUICKSORT decrescente (in modo tale che il primo ramo scelto é quello con peso maggiore) in quanto i tempi di

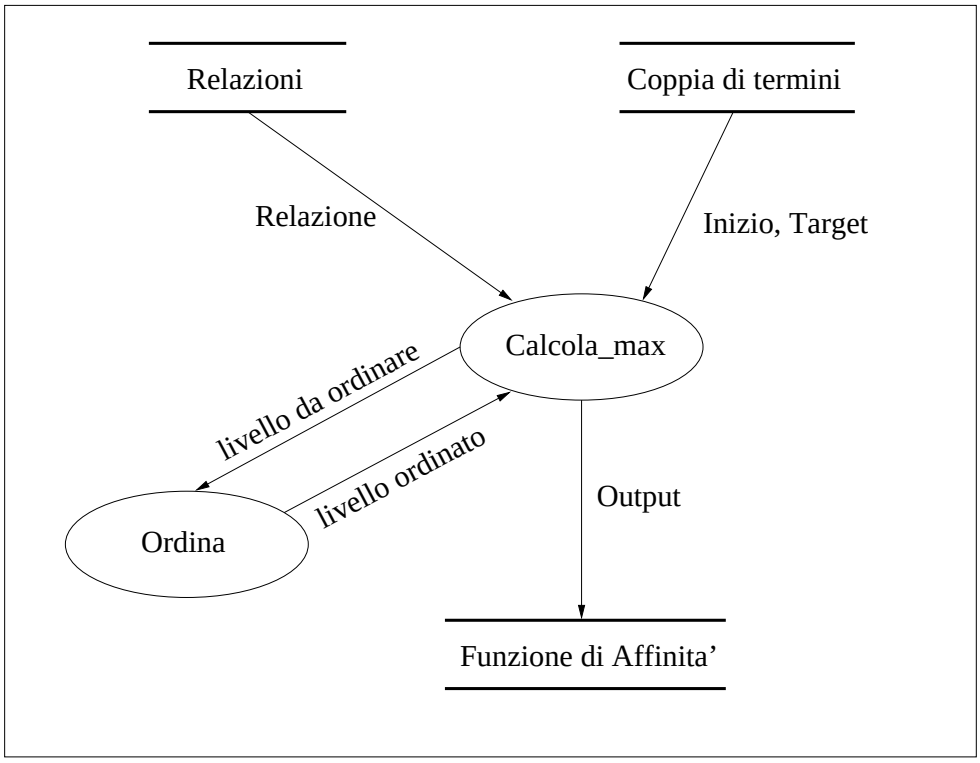


Figura 5.6: DFD della procedura ricerca Max_Path

esecuzione del modulo ESCA sono risultati decisamente inferiori.
Con riferimento all'esempio di Figura 5.7, le relazioni terminologiche presenti nel Thesaurus definiscono un "*albero virtuale*" rappresentato in figura. La procedura di ricerca, essendo i pesi delle relazioni coinvolte nel primo livello uguali, sceglie il primo nodo che trova, `Professor`, mentre al secondo livello invece di visitare il nodo `Section`, a seguito dell'ordinamento sceglie `Research_staff` in quanto BT $>$ RT, trovando, in questo modo, un valore massimo provvisorio di *affinitá* dato da $\rightarrow 0,8 \cdot 0,8 \cdot 0,5 \cdot 0,8 = 0,256$.

- *controllo*: il modulo tiene traccia del massimo relativo ad ogni livello del percorso, in modo da scartare tutti quei rami che, se visitati, porterebbero ad un valore minore del massimo fino a quel momento trovato.

Esempio 13 Riprendiamo l'esempio 12, si supponga che dopo aver trovato il massimo relativo, la procedura risalga l'albero fino a `Professor`, resta da visitare, quindi, `Section`. Il percorso tra `CS_Person` e `Section` vale $0,8 \cdot 0,5 = 0,4$, il quale essendo maggiore del massimo momentaneo (0,256) permette di

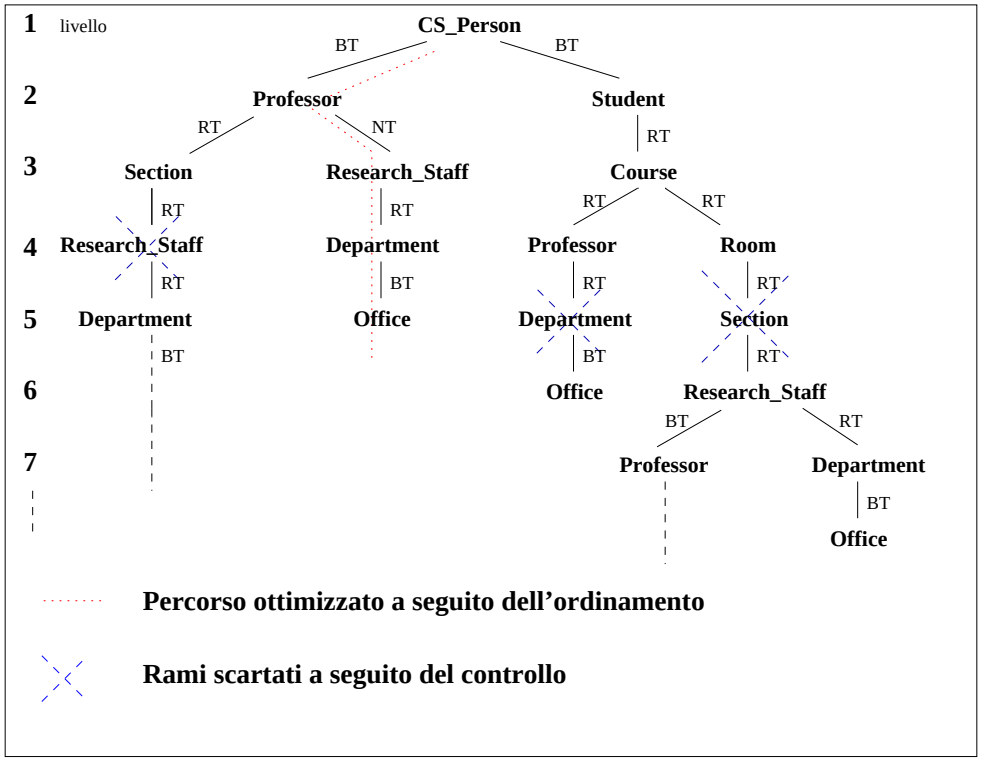


Figura 5.7: Fasi di ottimizzazione della procedura di ricerca

visitare il nodo. Il successivo nodo, `Research_Staff`, peró non é visitato in quanto il massimo sarebbe $0,8 \cdot 0,5 \cdot 0,5 = 0,2$ che é minore di 0,256. Il modulo a questo punto interrompe la ricerca e risale l'albero fino all'altro figlio di `CS_Person`, `Student`, tagliando tutti i rami del nodo `Section`, in quanto essi porterebbero a dei valori del percorso sempre minori.
In questo modo il percorso con valore della funzione di affinitá maggiore, é quello rappresentato in modo tratteggiato.

5.1.3 Valutazione dei Coefficienti di affinitá delle classi

In Figura 5.8 é riportato il Data Flow Diagram della procedura del calcolo di affinitá globale, dal quale emergono le fasi piú importanti per il calcolo dei coefficienti; In particolare per ogni coppia di classi il modulo GA() effettua i seguenti passi:

- per classi i cui nomi coincidono, il modulo assegna direttamente al cammino piú stringente il valore massimo "1"; altrimenti effettua una chiamata alla

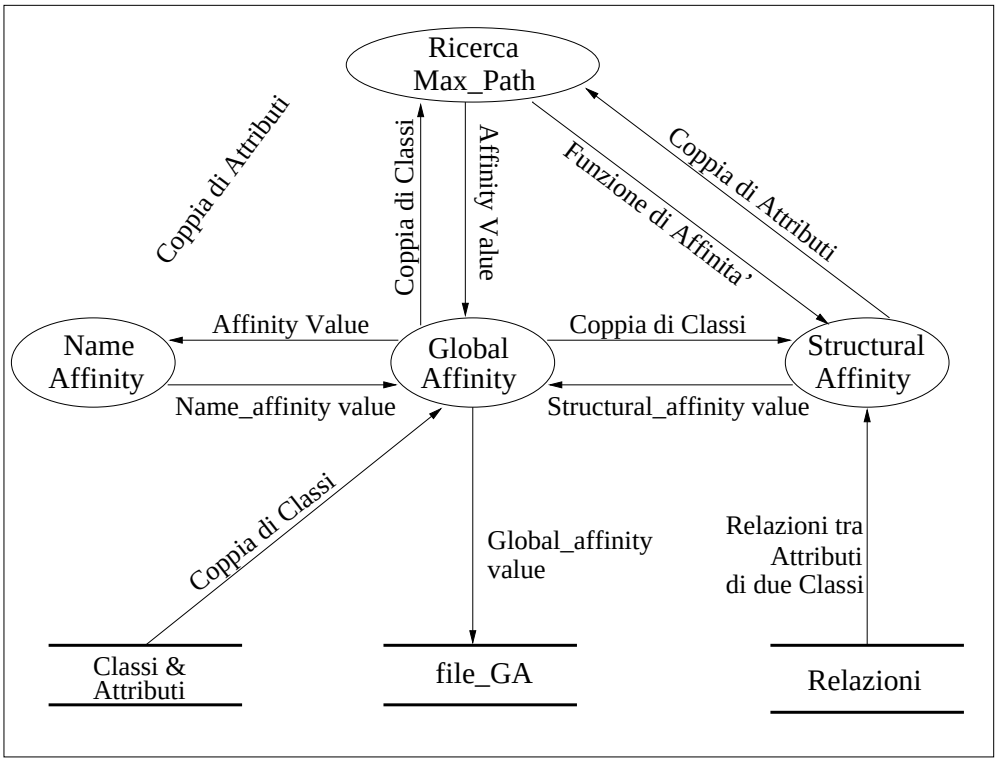


Figura 5.8: DFD della procedura di Valutazione dei Coefficienti di affinitá

procedura *Ricerca*, presentata in Sezione 5.1.2, la quale ritorna il valore del cammino massimo;

- *Name Affinity Coefficient*: il primo passo di questa fase é definire l'affinitá tra i termini considerati, confrontando la funzione di affinitá con un valore di soglia α. In caso affermativo il Name Affinity Coefficient, coincide proprio con tale funzione, in caso contrario é assegnato il valore "0";

- *Structural Affinity Coefficient*: per definire tale valore sono estratte per le due classi in esame:

  – tutte le possibili coppie di attributi delle classi e per ognuna di esse é chiamata la procedura di ricerca del commino piú *stringente*, per poi confrontare il valore della funzione di affinitá ricevuto con il valore di soglia α, in modo da definire il numero di coppie affini;

  – il numero di attributi di ogni classe memorizzato nella struttura dati presentata in Sezione 5.1.1;

  - il numero di relazioni esistenti nel Thesaurus tra gli attributi delle classi in esame e di queste vengono valutate quante sono validate (cioé hanno flag di controllo ”1”);

- *Global Affinity Coefficient*: sulla base dei coefficienti calcolati nei due passi precedenti é estratto il Global Affinity Coefficient ponendo w_{NA} = w_{SA} = 0.5. Per classi i cui nomi non hanno nulla in comune il modulo non valuta l’affinitá rispetto ai loro attributi, assengnando al GA() il valore nullo.

Il compito del modulo **ESCA** termina con la scrittura di un file di output (con appendice **_GA**) in cui saranno memorizzati, per ogni coppia di classi, i Global Coefficient per essere successivamente utilizzati nella fase di Clustering.

5.2 Esempio di esecuzione

In questa sezione verrá presentato un esempio di funzionamento del modulo ESCA, al fine di mettere in evidenza la funzionalitá e l’esattezza dei calcoli effettuati durante l’esecuzione del software. A tale scopo sono stati appositamente definiti dei file di esempio ”.odl” e ”.th”, in modo da coprire il maggior numero di casi possibili, e si é imposto, inoltre, come valore di soglia α = 0.1.
Si supponga di avere a disposizione i seguenti schemi ODL_{I^3}, relativi alle sorgenti S1, S2, S3:

```
interface Class
 ( source object S1    )
{       attribute string Att1a;
        attribute string Att1b;
        attribute string Att1c;   };


interface Class
 ( source object S2    )
{       attribute string Att2a;
        attribute string Att2b;
        attribute string Att2c;   };

interface Class3
 ( source object S3    )
{       attribute string Att3a;
        attribute string Att3b;
  };
```

e di aver le seguenti relazioni nel file di thesaurus (Relazioni):

```
S1.Class.Att1a  ,bt   ,S2.Class.Att2b           ,0;
S2.Class.Att2b  ,nt   ,S1.Class.Att1b           ,1;
S1.Class.Att1b  ,syn  , S3.Class3.Att3b           ,0 ;
S3.Class3.Att3b  ,rt   ,S2.Class.Att2b           ,1 ;
S3.Class3.Att3b  ,syn   ,S2.Class.Att2a           ,1 ;
S1.Class  ,bt   ,S3.Class3           , ;
S2.Class  ,rt   ,S3.Class3           , ;
```

Dopo aver mandato in esecuzione il modulo ESCA si avrá la seguente schermata di output. Nell'esempio presentato, per ogni coppia di classi l'output prevede tutte le variabili che entrano in gioco nel calcolo dei coefficienti. In particolare, *Name Affinity*, *Affinitá fra attributi e numero di attributi affini*, *Flag di controllo*, *numero di attributi di ogni classe*, *Structural Affinity* e *Global Affinity*:

```
grifa@sparc20:~/tesi/tmp >java esca schema

Affinita' tra le Classi S1.Class S2.Class

Name Affinity  NA 1.0
Attribut Affinity
S1.Class.Att1a S2.Class.Att2a 0.6400000000000001
S1.Class.Att1a S2.Class.Att2b 0.8
S1.Class.Att1b S2.Class.Att2a 1.0
S1.Class.Att1b S2.Class.Att2b 0.8
C 2.0  X 1.0  Fc 0.5  num1_att 3.0  num2_att 3.0
num_att_affini 4.0
Structural Affinity  SA 0.6666666666666666
```

Global Affinity S1.Class S2.Class GA 0.8333333333333333

```
Affinita' tra le Classi S1.Class S3.Class3

Name Affinity  NA 0.8
Attribut Affinity
S1.Class.Att1a S3.Class3.Att3b 0.6400000000000001
S1.Class.Att1b S3.Class3.Att3b 1.0
C 1.0  X 0.0  Fc 0.0  num1_att 3.0  num2_att 2.0
num_att_affini 2.0
Structural Affinity  SA 0.0
```

Global Affinity S1.Class S3.Class3 GA 0.4

```
Affinita' tra le Classi S2.Class S3.Class3
```

```
Name Affinity  NA 0.5
Attribut Affinity
S2.Class.Att2a S3.Class3.Att3b 1.0
S2.Class.Att2b S3.Class3.Att3b 0.8
C 1.0  X 1.0  Fc 1.0  num1_att 3.0  num2_att 2.0
num_att_affini 2.0
Structural Affinity  SA 0.8
```

Global Affinity S2.Class S3.Class3 GA 0.65

Si fa notare che nel Thesaurus é stato definito un percorso ciclico

- `S2.Class2.Att2b` $\rightarrow^{nt}$ `S1.Class1.Att1b` $\rightarrow^{syn}$ `S3.Class3.Att3b` $\rightarrow^{rt}$ `S2.Class2.Att2b`

che potrebbe creare dei problemi quando viene calcolato il cammino *stringente* tra `"S1.Class1.Att1a` e `S2.Class2.Att2a"` in quanto porterebbe il modulo in uno stato di *loop*. Infatti, il cammino effettuato sarebbe

- `S1.Class1.Att1a` $\rightarrow^{bt}$ `S2.Class2.Att2b` $\rightarrow^{nt}$ `S1.Class1.Att1b` $\rightarrow^{syn}$ `S3.Class3.Att3b` $\rightarrow^{rt}$ `S2.Class2.Att2b` ...

invece il modulo non va in *loop*, in quanto la procedura di ricerca per ogni cammino tiene traccia dei nodi percorsi, in modo da evitare di visitare lo stesso nodo (durante lo stesso cammino) piú di una volta, situazione che potrebbe portare ad uno stato di loop.

- **NA**: é il *Name Coefficient Affinity* tra i nomi delle due classi in esame. Come si puó notare nel primo caso, nel quale i nomi delle classi sono identici (*Class*), tale coefficiente assume il valore di '1' pur non essendoci nel Thesaurus nessuna relazione "SYN" tra le classi di *source1* e *source2*. Negli altri due casi, invece, assume il valore della funzione di affinitá, perché esiste una relazione che nega la SYN, cioé BT ed RT dichiarate nel Thesaurus.

- **Fc** (*Flag di controllo*) = **X/C**: dove *C* é il numero di relazioni, presenti nel Thesaurus, tra coppie di attributi (uno appartenente alla classe di Source1 e l'altro a quella di Source2) validabili, mentre *X* rappresenta quante di esse sono valide (cioé che hanno flag '1'). Come si puó notare, per esempio nel primo caso, le relazioni tra attributi della classe di Source1 e quelli della classe di Source2 sono '2', mentre solo una di esse ha flag uguale a '1';

- **SA**: é lo *Structural Coefficient Affinity* che é dato da:

$$SA = \frac{2 \cdot num\_att\_affini}{num1\_att + num2\_att} \cdot F_c$$

- **GA**: é il *Global Coefficient Affinity*, il cui valore é facilmente verificabile tenendo presente che é stato calcolato ponendo w_{NA} = w_{SA} = 0.5.

Di seguito é riportato, invece, il file di output "*schema_GA*" generato dal modulo in base ai file di input presentati precedentemente:

```
::::::::::::::
schema_GA
::::::::::::::
S1.Class S2.Class 0.8333333333333333
S1.Class S3.Class3 0.4
S2.Class S3.Class3 0.65
```

5.3 Istruzioni per l'utente

Il modulo riceve in input un *nome_file* senza estensione, sará poi il software a chiamare, a seconda della necessitá, il file *nome_file.odl* (dello schema ODL$_{I^3}$) o *nome_file.th* (del file di Thesaurus). In particolare, sará quindi necessario che sia il file degli schemi sorgenti forniti in ODL$_{I^3}$ che quello di Thesaurus abbiano lo stesso nome. Il modulo é eseguibile tramite il comando:

```
java esca  nome_file.
```

Conclusioni

Lo studio compiuto in questa tesi ha arricchito il progetto MOMIS di un componente importante per la fase di analisi ed integrazione degli schemi di sorgenti eterogenee di informazioni. L'area di ricerca in cui si inserisce il progetto, l'Integrazione Intelligente di Informazioni (sistemi I^3), é abbastanza recente ma si avvale di tecniche di Intelligenza Artificiale ampiamente consolidate.

Il lavoro svolto si é articolato in diverse fasi prima di giungere al progetto vero e proprio. Inizialmente, infatti, é stato studiato il sistema MOMIS al fine di capire il funzionamento e l'ambiente in cui il progetto si inserisce. Successivamente é stato studiato il linguaggio *Java* con cui si é pensato di realizzare il software, un linguaggio ad oggetti sviluppato da Sun Microsystems che ha ottenuto recentemente un grande successo sia in ambito di ricerca che tra gli sviluppatori di applicazioni. A questo punto é stato possibile iniziare la fase di progetto e realizzazione del modulo software che costituisce la parte principale del lavoro svolto.

Obiettivo di MOMIS é fornire l'accesso a sorgenti di informazioni, con lo scopo di integrare, analizzare, e sintetizzare i dati che esse mettono a disposizione, aumentando il valore complessivo delle informazioni ed eliminando incongruenze e duplicati. In particolare, in questo lavoro sono state presentate tecniche per l'analisi di un largo insieme di schemi concettuali e soprattutto per la valutazione della *similaritá semantica* tra entitá in differenti schemi di dati strutturati, sulla base di relazioni terminologiche fornite in un Thesaurus. Inoltre sono stati definiti e calcolati tutti i coefficienti che definiscono l'affinitá strutturale delle entitá da analizzare. Tale parte é di importanza strategica al fine di integrare ed analizzare le informazioni messe a disposizione.

Nella realizzazione degli obiettivi si é rivelato di fondamentale importanza il componente ARTEMIS-Tool [16, 17], utilizzato in tutta la fase del progetto.

In futuro, per il completamento della fase di analisi ed integrazione degli schemi del sistema MOMIS é necessario, utilizzando i coefficienti di affinitá globale calcolati nella presente tesi, sviluppare la fase di *generazione dei cluster* mediante delle tecniche di clusterizzazione presentate in questa tesi, per poi generare lo *schema globale* del mediatore che sará la base per porre le query sui dati delle sorgenti. Mentre sono in via di sviluppo i processi di elaborazione ed ottimizzazione delle query.

Appendice A

Glossario I^3

Questo glossario ed il vocabolario sul quale si basa sono stati originariamente sviluppati durante l'I^3 Architecture Meeting in Boulder CO, 1994, sponsorizzato dall'ARPA, e rifiniti in un secondo incontro presso l'Universitá di Stanford, nel 1995. Il glossario é strutturato logicamente in diverse sezioni:

- Sezione 1: Architettura
- Sezione 2: Servizi
- Sezione 3: Risorse
- Sezione 4: Ontologie

Nota: poiché la versione originaria del glossario usa una terminologia inglese, in alcuni casi é riportato, a fianco del termine, il corrispettivo inglese, quando la traduzione dal termine originale all'italiano poteva essere ambigua o poco efficace.

A.1 Architettura

- Architettura = insieme di componenti.
- architettura di riferimento = linea guida ed insieme di regole da seguire per l'architettura.
- componente = uno dei blocchi sui quali si basa una applicazione o una configurazione. Incorpora strumenti e conoscenza specifica del dominio.
- applicazione = configurazione persistente o transitoria dei componenti, rivolta a risolvere un problema del cliente, e che puó coprire diversi domini.

- configurazione = istanza particolare di una architettura per una applicazione o un cliente.
- collante (glue) = software o regole che servono per per collegare i componenti o per interoperare attraverso i domini.
- strato = grossolana categorizzazione dei componenti e degli strumenti in una configurazione. L'architettura I^3 distingue tre strati, ognuno dei quali fornisce una diversa categoria di servizi:
  1. Servizi di Coordinamento = coprono le fasi di scoperta delle risorse, distribuzione delle risorse, invocazione, scheduling...
  2. Servizi di Mediazione = coprono la fase di query processing e di trattamento dei risultati, nonché il filtraggio dei dati, la generazione di nuove informazioni, etc.
  3. Servizi di Wrapping = servono per l'utilizzo dei wrappers e degli altri strumenti simili utilizzati per adattarsi a standards di accesso ai dati e alle convenzioni adoperate per la mediazione e per il coordinamento.
- agente = strumento che realizza un servizio, sia per il suo proprietario, sia per un cliente del suo proprietario.
- facilitatore = componente che fornisce i servizi di coordinamento, come pure l'instradamento delle interrogazioni del cliente.
- mediatore = componente che fornisce i servizi di mediazione e che provvede a dare valore aggiunto alle informazioni che sono trasmesse al cliente in risposta ad una interrogazione.
- cliente (customer) = proprietario dell'applicazione che gestisce le interrogazioni, o utente finale, che usufruisce dei servizi.
- risorsa = base di dati accessibile, server ad oggetti, base di conoscenze...
- contenuto = risultato informativo ricavato da una sorgente.
- servizio = funzione fornita da uno strumento in un componente e diretta ad un cliente, direttamente od indirettamente.
- strumento (tool) = programma software che realizza un servizio, tipicamente indipendentemente dal dominio.
- wrapper = strumento utilizzato per accedere alle risorse conosciute, e per tradurre i suoi oggetti.

- regole limitative (constraint rules) = definizione di regole per l'assegnamento di componenti o di protocolli a determinati strati.
- interoperare = combinare sorgenti e domini multipli.
- informazione = dato utile ad un cliente.
- informazione azionabile = informazione che forza il cliente ad iniziare un evento.
- dato = registrazione di un fatto.
- testo = dato, informazione o conoscenza in un formato relativamente non strutturato, basato sui caratteri.
- conoscenza = metadata, relazione tra termini, paradigmi…, utili per trasformare i dati in informazioni.
- dominio = area, argomento, caratterizzato da una semantica interna, per esempio la finanza, o i componenti elettronici…
- metadata = informazione descrittiva relativa ai dati di una risorsa, compresi il dominio, proprietá, le restrizioni, il modello di dati,…
- metaconoscenza = informazione descrittiva relativa alla conoscenza in una risorsa, includendo l'ontologia, la rappresentazione…
- metainformazioni = informazione descritiiva sui servizi, sulle capacitá, sui costi…

A.2 Servizi

- Servizio = funzionalitá fornita da uno o piú componenti, diretta ad un cliente.
- instradamento (routing) = servizio di coordinamento per localizzare ed invocare una risorsa o un servizio di mediazione, o per creare una configurazione. Fa uso di un direttorio.
- scheduling = servizio di coordinamento per determinare l'ordine di invocazione degli accessi e di altri servizi; fa spesso uso dei costi stimati.
- accoppiamento (matchmaking) = servizio che accoppia i sottoscrittori di un servizio ai fornitori.

- intermediazione (brokering) = servizio di coordinamento per localizzare le risorse migliori.
- strumento di configurazione = programma usato nel coordinamento per aiutare a selezionare ed organizzare i componenti in una istanza particolare di una configurazione architetturale.
- servizi di descrizione = metaservizi che informano i clienti sui servizi, risorse...
- direttorio = servizio per localizzare e contattare le risorse disponibili, come le pagine gialle, pagine bianche...
- decomposizione dell'interrogazione (query decomposition) = determina le interrogazioni da spedire alle risorse o ai servizi disponibili.
- riformulazione dell'interrogazione (query reformulation) = programma per ottimizzare o rilassare le interrogazioni, tipicamente fa uso dello scheduling.
- contenuto = risultato prodotto da una risorsa in risposta ad interrogazioni.
- trattamento del contenuto (content processing) = servizio di mediazione che manipola i risultati ottenuti, tipicamente per incrementare il valore delle informazioni.
- trattamento del testo = servizio di mediazione che opera sul testo per ricerca, correzione...
- filtraggio = servizio di mediazione per aumentare la pertinenza delle informazioni ricevute in risposta ad interrogazioni.
- classificazione (ranking) = servizio di mediazione per assegnare dei valori agli oggetti ritrovati.
- spiegazione = servizio di mediazione per presentare i modelli ai clienti.
- amministrazione del modello = servizio di mediazione per permettere al cliente ed al proprietario del mediatore di aggiornare il modello.
- integrazione = servizio di mediazione che combina i contenuti ricevuti da una molteplicitá di risorse, spesso eteorgenee.
- accoppiamento temporale = servizio di mediazione per riconoscere e risolvere differenze nelle unitá di misura temporali utilizzate dalle risorse.

- accoppiamento spaziale = servizio di mediazione per riconoscere e risolvere differenze nelle unitá di misura spaziali utilizzate dalle risorse.
- ragionamento (reasoning) = metodologia usata da alcuni componenti o servizi per realizzare inferenze logiche.
- browsing = servizio per permettere al cliente di spostarsi attraverso le risorse.
- scoperta delle risorse = servizio che ricerca le risorse.
- indicizzazione = creazione di una lista di oggetti (indice) per aumentare la velocitá dei servizi di accesso.
- analisi del contenuto = trattamento degli oggetti testuali per creare informazioni.
- accesso = collegamento agli oggetti nelle risorse per realizzare interrogazioni, analisi o aggiornamenti.
- ottimizzazione = processo di manipolazione o di riorganizzazione delle interrogazioni per ridurne il costo o il tempo di risposta.
- rilassamento = servizio che fornisce un insieme di risposta maggiore rispetto a quello che l'interrogazione voleva selezionare.
- astrazione = servizio per ridurre le dimensioni del contenuto portandolo ad un livello superiore.
- pubblicitá (advertising) = presentazione del modello di una risorsa o del mediatore ad un componente o ad un cliente.
- sottoscrizione = richiesta di un componente o di un cliente di essere informato su un evento.
- controllo (monitoring) = osservazione delle risorse o dei dati virtuali e creazione di impulsi da azionare ogniqualvolta avvenga un cambiamento di stato.
- aggiornamento = trasmissione dei cambiamenti dei dati alle risorse.
- instanziazione del mediatore = popolamento di uno strumento indipendente dal dominio con conoscenze dipendenti da un dominio.
- attivo (activeness) = abilitá di un impulso di reagire ad un evento.

- servizio di transazione = servizio che assicura la consistenza temporale dei contenuti, realizzato attraverso l'amministrazione delle transzioni.
- accertamento dell'impatto = servizio che riporta quali risorse saranno interessate dalle interrogazioni o dagli aggiornamenti.
- stimatore = servizo di basso livello che stima i costi previsti e le prestazioni basandosi su un modello, o su statistiche.
- caching = mantenere le informazioni memorizzate in un livello intermedio per migliorare le prestazioni.
- traduzione = trasformazione dei dati nella forma e nella sintassi richiesta dal ricevente.
- controllo della concorrenza = assicurazione del sincronismo degli aggiornamenti delle risorse, tipicamente assegnato al sistema che amministra le transazioni.

A.3 Risorse

- Risorsa = base di dati accessibile, simulazione, base di conoscenza,. . . comprese le risorse "legacy".
- risorse "legacy" = risorse preesistenti o autonome, non disegnate per interoperare con una architettura generale e flessibile.
- evento = ragione per il cambiamento di stato all'interno di un componente o di una risorsa.
- oggetto = istanza particolare appartenente ad una risorsa, al modello del cliente, o ad un certo strumento.
- valore = contenuto metrico presente nel modello del cliente, come qualitá, rilevanza, costo.
- proprietario = individuo o organizzazione che ha creato, o ha i diritti di un oggetto, e lo puó sfruttare.
- proprietario di un servizio = individuo o organizazzione responsabile di un servizio.
- database = risorsa che comprende un insieme di dati con uno schema descrittivo.

- warehouse = database che contiene o dá accesso a dati selezionati, astratti e integrati da una molteplicitá di sorgenti. Tipicamente ridondante rispetto alle sorgenti di dati.
- base di conoscenza = risorsa comprendente un insieme di conoscenze trattabili in modo automatico, spesso nella forma di regole e di metadata; permettono l'accesso alle risorse.
- simulazione = risorsa in grado di fare proiezioni future sui dati e generare nuove informazioni, basata su un modello.
- amministrazione della transazione = assicurare che la consistenza temporale del database non sia compromessa dagli aggiornamenti.
- impatto della transazione = riporta le risorse che sono state coinvolte in un aggiornamento.
- schema = lista dedlle relazioni, degli attributi e, quando possibile, degli oggetti, delle regole, e dei metadata di un database. Costituisce la base dell'ontologia della risorsa.
- dizionario = lista dei termini, fa parte dell'ontologia.
- modello del database = descrizione formalizzata della risorsa database, che include lo schema.
- interoperabilitá = capacitá di interoperare.
- eterogeneitá = incompatibilitá trovate tra risorse e servizi sviluppati autonamamente, che vanno dalla paittaforma utilizzata, sistema operativo, modello dei dati, alla semantica, ontologia,. . .
- costo = prezzo per fornire un servizio o un accesso ad un oggetto.
- database deduttivo = database in grado di utilizzare regole logiche per trattare i dati.
- regola = affermazione logica, unitá della conoscenza trattabile in modo automatico.
- sistema di amministrazione delle regole = software indipendente dal dominio che raccoglie, seleziona ed agisce sulle regole.
- database attivo = database in grado di reagire a determinati eventi.
- dato virtuale = dato rappresentato attraverso referenze e procedure.

- stato = istanza o versione di una base di dati o informazioni.
- cambiamento di stato = stato successivo ad una azione di aggiornamento, inserimento o cancellazione.
- vista = sottoinsieme di un database, sottoposto a limiti, e ristrutturato.
- server di oggetti = fornisce dati oggetto.
- gerarchia = struttura di un modello che assegna ogni oggetto ad un livello, e definisce per ogni oggetto l'oggetto da cui deriva.
- network = struttura di un modello che fa uso di relazioni relativamente libere tra oggetti.
- ristrutturare = dare una struttura diversa ai dati seguendo un modello differente dall'originale.
- livello = categorizzazione concettuale , dove gli oggetti di un livello inferiore dipendono da un antenato di livello superiore.
- antenato (ancestor) = oggetto di livello superiore, dal quale derivano attributi ereditabili.
- oggetto root = oggetto da cui tutti gli altri derivano, all'interno di una gerarchia.
- datawarehouse = deposito di dati integrati provenienti da una molteplicitá di risorse.
- deposito di metadata = database che contiene metadata o metainformazioni.

A.4 Ontologia

- Ontologia = descrizione particolareggiata di una concettualizzazione, i.e. l'insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti, spesso ambigui tra domini diversi.
- concetto = definisce una astrazione o una aggregazione di oggetti per il cliente.
- semantico = che si riferisce al significato di un termine, espresso come un insieme di relazioni.

- sintattico = che si riferisce al formato di un termine, espresso come un insieme di limitazioni.
- classe = definisce metaconoscenze come metodi, attributi, ereditarietá, per gli oggetti in essa istanziati.
- relazione = collegamento tra termini, come *is-a*, *part-of*,. . .
- ontologia unita (merged) = ontologia creata combinando diverse ontologie, ottenuta mettendole in relazione tra loro (mapping).
- ontologia condivisa = sottoinsieme di diverse ontologie condiviso da una molteplicitá di utenti.
- comparatore di ontologie = strumento per determinare relazioni tra ontologie, utilizzato per determinare le regole necessarie per la loro integrazione.
- mapping tra ontologie = trasformazione dei termini tra le ontologie, attraverso regole di accoppiamento, utilizzato per collegare utenti e risorse.
- regole di accoppiamento (matching rules) = dichiarazioni per definire l'equivalenza tra termini di domini diversi.
- trasformazione dello schema = adattamento dello schema ad un'altra ontologia.
- editing = trattamneto di un testo per assicurarne la conformitá ad una ontologia.
- algebra dell'ontologia = insieme delle operazioni per definire relazioni tra ontologie.
- consistenza temporale = é raggiunta se tutti i dati si riferiscono alla stessa istanza temporale ed utilizzano la stessa granularitá temporale.
- specifico ad un dominio = relativo ad un singolo dominio, presuppone l'assenza di incompatibilitá semantiche.
- indipendente dal dominio = software, strumento o conoscenza globale applicabile ad una molteplicitá di domini.

Appendice B

Il linguaggio descrittivo ODL$_{I^3}$

Si riporta la descrizione in BNF del linguaggio descrittivo ODL$_{I^3}$. Essendo questo una estensione del linguaggio standard ODL, si riportano in questo appendice solo le parti che differiscono dall'ODL originale, rimandando invece a quest'ultimo per le parti in comune.

⟨interface_dcl⟩ : : = ⟨interface_header⟩ {[⟨interface_body⟩]};
⟨interface_header⟩ : : = **interface** ⟨identifier⟩
[⟨inheritance_spec⟩]
[⟨type_property_list⟩]
⟨inheritance_spec⟩ : : = : ⟨scoped_name⟩ [,⟨inheritance_spec⟩]
⟨type_property_list⟩ : : = **(** [⟨source_spec⟩] [⟨extent_spec⟩]
[⟨key_spec⟩] [⟨f_key_spec⟩] **)**
⟨source_spec⟩ : : = **source** ⟨source_type⟩ ⟨source_name⟩
⟨source_type⟩ : : = **relational** | **nfrelational** | **object** | **file**
⟨source_name⟩ : : = ⟨identifier⟩
⟨extent_spec⟩ : : = **extent** ⟨extent_list⟩
⟨extent_list⟩ : : = ⟨string⟩ | ⟨string⟩ **,** ⟨extent_list⟩
⟨key_spec⟩ : : = **key**[**s**] ⟨key_list⟩
⟨f_key_spec⟩ : : = **foreign_key** ⟨f_key_list⟩
...

⟨attr_dcl⟩ ::= **[readonly] attribute**
⟨domain_type⟩ ⟨attribute_name⟩
[⟨fixed_array_size⟩] [⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ ::= **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ ::= ⟨rule⟩ | ⟨rule⟩**,**⟨rule_list⟩
⟨rule⟩ ::= ⟨local_attr_name⟩ | **'**⟨identifier⟩**'**
⟨and_expression⟩ | ⟨or_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and** ⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **and** ⟨and_list⟩
⟨or_expression⟩ ::= **(** ⟨local_attr_name⟩ **or** ⟨or_list⟩ **)**
⟨or_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **or** ⟨or_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩**.**⟨class_name⟩**.**⟨attribute_name⟩
...
⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** | ⟨relationship_dcl⟩**;** ⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
⟨relationship_type⟩ ::= **syn** | **bt** | **nt** | **rt**
...
⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_pre⟩ **then** ⟨rule_post⟩
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** | ⟨rule_body⟩ |
⟨rule_body_list⟩ **and** ⟨rule_body⟩ |
⟨rule_body_list⟩ **and (** ⟨rule_body_list⟩ **)**
⟨rule_body⟩ ::= ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in** ⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
⟨rule_const_op⟩ ::= = | ≥ | ≤ | > | <
⟨rule_cast⟩ ::= **(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**⟨dotted_name⟩
⟨forall⟩ ::= **for all** | **forall**

Appendice C

Esempio in ODL$_{I^3}$

Di seguito é riportata la descrizione, attraverso il linguaggio ODL$_{I^3}$, dell'esempio di riferimento di Sezione 4.2.

```
UNIVERSITY source:

interface Research_Staff                interface School_Member
( source relational University          ( source relational Universit
  extent Research_Staffers                extent School_Members
  keys first_name, last_name              keys first_name, last_name
  foreign_key dept_code, section_code ) { attribute string  first_nam
{ attribute string first_name;            attribute string  last_name
  attribute string last_name;             attribute string  faculty;
  attribute string relation;              attribute integer year; }
  attribute string e_mail;
  attribute integer dept_code;
  attribute integer section_code; };

interface Department                    interface Section
( source relational University          ( source relational Universit
  extent Departments                      extent Sections
  key dept_code )                         key section_code
{ attribute string dept_name;             foreign_key room_code )
  attribute integer dept_code;          { attribute string section_na
  attribute integer budget;               attribute integer section_c
  attribute string dept_area; };          attribute integer length;
                                          attribute integer room_code

interface Room
```

```
( source relational University
  extent Room
  key room_code )
{ attribute integer room_code;
  attribute integer seats_number;
  attribute string notes; };
```

COMPUTER_SCIENCE source:

```
interface CS_Person
( source object Computer_Science
  extent CS_Persons
  key name )
{ attribute string name; };
```

```
interface Professor : CS_Person
( source object Computer_Science
  extent Professors )
{ attribute string title;
  attribute Division belongs_to;
  attribute string rank; };
```

```
interface Student : CS_Person
( source object Computer_Science
  extent Students )
{ attribute integer year;
  attribute set<Course> takes;
  attribute string rank; };
```

```
interface Division
( source object Computer_Science
  extent Divisions
  key description )
{ attribute string description;
  attribute Location address;
  attribute integer fund;
  attribute integer employee_nr;
  attribute string sector; };
```

```
interface Location
( source object Computer_Science
  extent Locations
  keys city, street, county, number)
{ attribute string city;
  attribute string street;
  attribute string county;
  attribute integer number; };
```

```
interface Course
( source object Computer_Science
  extent Courses
  key course_name )
{ attribute string course_name;
  attribute Professor taught_by; };
```

Tax_Position source:

```
interface University_Student
( source file Tax_Position
  extent University_Students
  key student_code )
```

```
{ attribute string name;
  attribute integer student_code;
  attribute string faculty_name;
  attribute integer tax_fee; };
```

Appendice D

ESCA: il codice sorgente

Bibliografia

[1] Gio Wiederhold et al. *Integrating Artificial Intelligence and Database Technology*, volume 2/3. Journal of Intelligent Information Systems, June 1996.

[2] Simone Montanari. Un approccio intelligente all' integrazione di sorgenti eterogenee di informazione, 1996-1997.

[3] S.Bergamaschi, S.Castano, S.De Capitani di Vimercati, S.Montanari, and M.Vincini. exploiting schema knowledge for the integration of heterogeneous sources. *Accepted for: Sistemi Evoluti per Basi di Dati, SEBD98*.

[4] S.Bergamaschi, S.Castano, S.De Capitani di Vimercati, S.Montanari, and M.Vincini. An intelligent approach to information integration. *Accepted for: Formal Ontology in Information Systems FOIS98*.

[5] Arpa i^3 reference architecture. Available at http://www.isse.gmu.edu/I3_Arch/index.html.

[6] E.Rodriguez F.Saltor. On intelligent access to heterogeneous information. In *Proceedings of the 4th KRDB Workshop*, Athens, Greece, August 1997.

[7] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.

[8] N.Guarino. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. Technical report, Summer School on Information Extraction, Frascati, Italy, July 1997.

[9] N.Guarino. Understanding, building, and using ontologies. A commentary to 'Using Explicit Ontologies in KBS Development', by van Heijst, Schreiber, and Wielinga.

[10] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. Odb-qoptimizer: a tool for semantic query optimization in oodb. In *Proc. of Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, April 1997.

[11] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. Odb-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Proc. of Int. Conference of the Italian Association for Artificial Intelligence (AI*IA97)*, Rome, 1997.

[12] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem Solving Technologies*, 4:185–203, 1994.

[13] D. Beneventano and S. Bergamaschi. Incoherence and subsumption for recursive views and queries in object-oriented data models. *Data & Knowledge Engineering*, 1996. To appear.

[14] D. Beneventano, S. Bergamaschi, and C. Sartori. Taxonomic reasoning with cycles in LOGIDATA+. In P. Atzeni, editor, *LOGIDATA+: Deductive Databases with Complex Objects*. Springer-Verlag, Heidelberg - Germany, 1993. Lecture Notes in CS - N. 701.

[15] C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, Philadelphia, PA, 1989.

[16] S. Castano and V. De Antonellis. Semantic dictionary design for database interoperability. In *Proc. of Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, April 1997.

[17] S. Castano, S. De Capitani Di Vimercati, V. De Antonellis, M. Melchiori. Information Integration on Multiple Heterogeneous Data Sources.

[18] S. Bergamaschi. Extraction of informations from highly heterogeneous sources of textual data. In *Cooperative Information Agents, First International Workshop, CIA' 97 Proceedings.*, Kiel, Germany, February 1997.

[19] Object Request Broker Task Force. The common object request broker: Architecture and specification, December 1993.

[20] R. Cattel, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1996.

[21] P. Buneman, L. Raschid, and J. Ullman. Mediator languages - a proposal for a standard, April 1996. Available at ftp://ftp.umiacs.umd.edu/pub/ONRrept/medmodel96.ps.

[22] N.V. Findler, editor. *Associative Networks*. Academic Press, 1979.

[23] BYACC/JAVA. Available at http://www.lincom-asg.com/ rjamison/byacc

[24] Alberto Zanoli. *SI-Designer, un tool di ausilio all'integrazione di sorgenti di dati eteregenee distribuite: Progetto e Realizzazione*, 1997-1998.