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

Sviluppo di tecniche di estrazione ed inferenza di relazioni terminologiche nel sistema MOMIS

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Elisa Marri

Correlatore
Dott. Ing. Domenico Beneventano

Controrelatore
Chiar.mo Prof. Flavio Bonfatti

Anno Accademico 1999 - 2000

Parole chiave:

Intelligent Information Integrator
Integrazione semantica
Common Thesaurus
Mediatore
Chiave

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi e l'Ing. Domenico Beneventano per l'aiuto fornito durante la realizzazione della presente tesi.
Rivolgo un sincero ringraziamento all'Ing. Alberto Corni per la costante disponibilità dimostrata e per i preziosi consigli di ordine pratico.

Vorrei poi fare due ringraziamenti speciali: il primo alla mia famiglia, senza il sostegno della quale non avrei mai raggiunto questo importante traguardo, il secondo al Dr. Alessandro Scaglioli per tutto quello che la sua umanità mi ha insegnato.

Indice

Introduzione **1**

1 Un Sistema Intelligente di Integrazione **5**

1.1 Integrazione Intelligente di Informazioni . . . 6

1.1.1 Il programma I^3 . . . 6

1.1.2 Architettura di riferimento per sistemi I^3 . . . 7

1.2 L'Approccio Semantico all'Integrazione . . . 11

1.2.1 Trattamento dei Metadati e delle Ontologie . . . 12

1.3 Il sistema MOMIS . . . 15

1.3.1 Scelte implementative . . . 17

1.3.2 Il Modello dei dati . . . 21

1.3.3 L'architettura di MOMIS . . . 23

1.3.4 Il Global Schema Builder . . . 25

1.3.5 Esempio di riferimento . . . 26

2 Gli strumenti utilizzati **29**

2.1 ODB-Tools Engine . . . 29

2.1.1 La Logica Descrittiva OLCD: aspetti generali . . . 29

2.1.2 OLCD: un Formalismo per Oggetti Complessi e Vincoli di Integrità . . . 31

2.1.3 Le regole OLCD e l'espansione semantica di un tipo . . . 34

2.1.4 Validazione e Sussunzione . . . 35

2.1.5 Architettura di ODB-Tools . . . 36

2.2 Il Database lessicale WordNet . . . 37

2.2.1 La Matrice Lessicale . . . 38

2.2.2 Tipi di relazioni . . . 39

3 Il processo di integrazione di sorgenti in MOMIS **43**

3.1 Generazione del Thesaurus Comune . . . 43

3.1.1 Acquisizione degli schemi ODL_{I^3} . . . 45

3.1.2 Estrazione delle relazioni intra-schema . . . 47

3.1.3 Estrazione delle relazioni inter-schema 47
3.1.4 Integrazione dell'insieme di relazioni 50
3.1.5 Validazione delle relazioni tra attributi 50
3.1.6 Validazione delle relazioni tra classi ed inferenza di nuove relazioni intensionali 51
3.2 Analisi di Affinità delle classi ODL$_{I^3}$ 51
3.2.1 Coefficienti di Affinità 53
3.3 Generazione dei Cluster 54
3.4 Generazione degli attributi globali e delle mapping-table 57

4 Il modulo SIM 63
4.1 Obiettivi del Modulo 63
4.2 Il modulo **SIMA** 65
4.3 Il modulo **SIMB** 68
4.3.1 Validazione delle relazioni tra attributi 69
4.3.2 Generazione del nuovo schema ed inferenza 72
4.4 Confronto fra il primo prototipo e l'attuale 76

5 Progetto e realizzazione del software 79
5.1 Il parser ODL$_{I^3}$ 79
5.2 Organizzazione software del modulo **SIM** 85
5.3 Implementazione del modulo **SIMA** 87
5.3.1 Analisi degli schemi relazionali 87
5.3.2 Analisi degli schemi ad oggetti 89
5.3.3 Popolamento del Thesaurus 93
5.4 Implementazione del modulo **SIMB** 94
5.4.1 Software per la validazione delle relazioni tra attributi . . 94
5.4.2 La classe ThesaurusByObject 96
5.4.3 Identificazione dei cicli 98
5.4.4 Software per la creazione dello schema virtuale 99
5.5 Funzionalità dell'interfaccia 104

Conclusioni 108

A Glossario I^3 111
A.1 Architettura 111
A.2 Servizi 113
A.3 Risorse 116
A.4 Ontologia 118

B Il linguaggio descrittivo ODL$_{I^3}$ 121

C Esempio di riferimento in ODL$_{I^3}$ **123**

D L'architettura CORBA **125**

Elenco delle figure

1.1 Diagramma dei servizi I^3 8
1.2 Servizi I^3 presenti nel mediatore 16
1.3 Architettura del sistema MOMIS 24
1.4 Architettura del Global Schema Builder 25
1.5 Esempio di riferimento 27

2.1 Gli schemi ODL$_{I^3}$ dell'esempio dei ristoranti 33
2.2 Architettura di ODB-Tool 36
2.3 La Matrice Lessicale 39

3.1 Il processo di generazione del Thesaurus Comune 44
3.2 L'interfaccia grafica di MOMIS 46
3.3 Significati di `address` 49
3.4 Relazioni intensionali ottenute dopo SLIM 49
3.5 Procedura di clustering 55
3.6 Albero di affinità 56
3.7 Pannello di visualizzazione dei cluster 57
3.8 Mapping-table di `Person` 61
3.9 Pannello di visualizzazione di una mapping-table 61

4.1 Il modulo **SIM** 64
4.2 Architettura del modulo **SIM** 65
4.3 Automa a stati finiti del modulo **SIMB** 69
4.4 DFD della procedura di Validazione delle relazioni fra attributi . . 70
4.5 DFD della procedura di inferenza 72
4.6 Traduzione OLCD degli schemi modificati in riferimento all'esempio dei ristoranti 74

5.1 Il modello ad oggetti della struttura dati ODL$_{I^3}$ 84
5.2 **SIM** integrato in SI_Designer 86
5.3 Algoritmo di analisi degli schemi relazionali 88
5.4 Traduzione OLCD degli schemi nell'esempio di riferimento . . . 91

5.5 Documento XML contenente l'output di ODB-Tools 92
5.6 Traduzione OLCD degli schemi per la validazione delle relazioni tra attributi 95
5.7 Algoritmo di ricerca dei cicli di relazioni NT 99
5.8 Alberi gerarchici 101
5.9 Traduzione OLCD degli schemi per la fase di inferenza 103
5.10 Visualizzazione del Thesaurus prima dell'intervento del progettista 105
5.11 Visualizzazione del Thesaurus dopo l'intervento del progettista . . 106

D.1 La invocazione di un metodo di un oggetto CORBA remoto 128
D.2 Esempio di albero creato dal naming server 128
D.3 Traduzione in Java di una interfaccia IDL di un oggetto CORBA . 129
D.4 Collegamento all'oggetto ODB-Tools realizzato da SIM 130

Introduzione

In questi ultimi anni abbiamo assistito ad un'esplosione del numero e della varietà di sorgenti di informazioni reperibili da parte di ognuno di noi. L'esempio più eclatante è quello della rete *Internet*, un fenomeno di massa che oggi va sempre più affermandosi grazie al diminuire dei costi relativi ed all'aumentare della facilità d'uso degli strumenti necessari. La grande mole di fonti a disposizione rende però critico il problema del ritrovamento ed dell'integrazione di informazioni: alla crescita delle sorgenti di informazione si è infatti accompagnato un degrado della qualità dell'informazione stessa. Il dato si trova spesso collocato in documenti non strutturati e quindi difficilmente comprensibili dai tradizionali strumenti di analisi; a questo si associano fenomeni quali l'*information overload* (che rende difficile discernere tra le varie sorgenti quelle che contengono dati significativi), l'inevitabile rindondanza e duplicazione di contenuti, la presenza di sorgenti di tipo ”semistrutturato” (quindi *disomogenee* anche nella propria costruzione).
L'eterogeneità dei dati si manifesta a diversi livelli: differenti piattaforme hardware e software sulle quali sono implementate le sorgenti (DBMS, linguaggi di interrogazione, ...), differenti modelli di dati (relazionale, ad oggetti, ...), differenti schemi di rappresentazione della struttura logica.
Lo scenario appena descritto risulta particolarmente problematico se si considera che molti dei compiti che devono essere portati a termine dagli utenti di sistemi complessi basati sui dati impongono l'interazione con una molteplicità di fonti di informazioni. Esempi possono essere trovati nelle aree che coinvolgono l'analisi aggregata dei dati (e.g. previsioni logistiche), come pure nella pianificazione delle risorse e nei sistemi di supporto alle decisioni.
Per gestire correttamente tutte le informazioni disponibili, l'utente dovrebbe essere in grado di eseguire (nel miglior modo possibile) diversi procedimenti:

- effettuare un'analisi accurata della propria richiesta;
- scomporre la propria richiesta in una miriade di ”sotto-richieste” rivolte alle singole sorgenti di informazione;
- inviare le ”sotto-richieste” ottenute alle rispettive sorgenti;
- eseguire un'operazione di *unione* dei risultati parziali, in modo da ottenere la risposta cercata.

Tutto ciò dovrebbe essere fatto tenendo conto delle possibili trasformazioni che possono subire i dati, delle relazioni che li legano, delle proprietà che possono avere in comune, nonchè delle discrepanze sussistenti tra le diverse rappresentazioni. Possedere uno strumento in grado di eseguire queste operazioni in maniera *automatica* o *semi-automatica* rappresenterebbe quindi una risorsa importantissima: l'utente potrebbe così formulare la sua richiesta esulando dall'effettiva natura ed organizzazione delle sorgenti.
Lo studio di applicazioni in grado di combinare e gestire dati provenienti da più sorgenti è di grande attualità ed interesse commerciale, come dimostra lo sviluppo di sistemi quali *Sistemi di Workflow, Datawarehouse, Dataminer, ecc*.
Questa tesi si inserisce all'interno di un progetto più ampio denominato **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources), sviluppato allo scopo di realizzare un processo semi-automatico di integrazione di sorgenti eterogenee e distribuite.
MOMIS adotta un'architettura a tre livelli:

1. livello *dati*
2. livello *mediatore*
3. livello *utente*

Il mediatore rappresenta il cuore del sistema in quanto realizza l'integrazione degli schemi e provvede alla gestione dei risultati delle operazioni di query. Al fine di generare una *vista globale* il più possibile espressiva e sulla quale potere effettuare delle interrogazioni significative, sono utilizzate tecniche di Intelligenza Artificiale e tecniche di Analisi degli schemi.

Più precisamente la seguente tesi si occupa dell'aspetto di integrazione *intensionale* degli schemi. Uno degli obiettivi è quello di sollevare il progettista da compiti che possono essere realizzati in automatico (vedi estrazione delle relazioni dagli schemi, validazione delle stesse, controlli di coerenza, ecc). La struttura della tesi è la seguente:

Nel **Capitolo 1** si introduce il concetto di Integrazione Intelligente di Informazioni, trattando l'architettura di riferimento I^3e la struttura di un mediatore. Si prenderà poi in esame il sistema MOMIS descrivendolo brevemente.

Nel **Capitolo 2** vengono descritti due componenti esterni utilizzati dal sistema MOMIS, ovvero ODB-Tools e WordNet. In particolare la presente tesi svilupperà moduli del sistema MOMIS che interagiscono con ODB-Tools.

Nel **Capitolo 3** viene presentato il processo di integrazione vera e propria eseguita dal sistema MOMIS, riportando anche le interfacce grafiche che ne visualizzano i risultati e che consentono l'interazione col progettista.

Nel **Capitolo 4** verrà presentato, dal punto di vista teorico, il modulo software che questa tesi realizza, ovvero il modulo **SIM**(Sources Integrator Module)

le cui funzionalità sono: estrazione di relazioni semantiche intra-schema, validazione delle relazioni semantiche inter-schema ed inferenza di nuove relazioni semantiche.

Nel **Capitolo 5** verrà discusso il software realizzato per implementare le funzionalità precedentemente descritte, motivando l'uso delle strutture dati e dei procedimenti scelti.

Sono inoltre presenti quattro appendici. In particolare in Appendice A viene riportato un glossario dei termini usati in ambito I^3, in Appendice B viene mostrato il linguaggio descrittivo ODL$_{I^3}$, in Appendice C si riporta l'esempio di riferimento, ed infine in Appendice D si descrive l'architettura CORBA.

Capitolo 1

Un Sistema Intelligente di Integrazione

Grazie alla crescita delle sorgenti di dati (tanto all'interno dell'azienda, quanto sulla rete) è oggi possibile accedere ad una quantità di informazioni decisamente molto ampia. La probabilità quindi di reperire un dato che interessa è aument

ata vertiginosamente, ma allo stesso tempo vediamo diminuire quella di giungerne in possesso nei tempi, ma soprattutto nei modi, desiderati. Questo è dovuto soprattutto alla grande eterogeneità dei dati disponibili, sia per quanto riguarda la natura (testi, immagini, etc.), sia il modo in cui vengono descritti.

Gli standard esistenti (TPC/IP, ODBC, OLE, CORBA, SQL, etc.) risolvono parzialmente i problemi relativi alle diversità hardware e software, dei protocolli di rete e di comunicazione tra i moduli; rimangono però irrisolti quelli relativi alla modellazione delle informazioni. Difatti i modelli e gli schemi dei dati sono differenti e questo crea una eterogeneità semantica (o logica) non risolvibile da questi standard.

Un'importante area, sia di ricerca che di applicazione, riguarda l'integrazione di DataBase eterogenei ed anche i *datawarehouse* (magazzino di dati). Questi lavori studiano la possibilità di materializzare presso l'utente finale delle viste, ovvero porzioni di sorgenti, replicando fisicamente i dati e quindi si devono poi affidare a complicati algoritmi di mantenimento ai fini di garantirne la consistenza.

Con il temine Integrazione di Informazioni (I^2) [1] si indicano invece in letteratura quei sistemi che, basandosi sulle descrizioni dei dati, combinano tra loro informazioni provenienti da diverse sorgenti (o parti selezionate di esse) senza quindi dover ricorrere alla duplicazione fisica.

1.1 Integrazione Intelligente di Informazioni

L'integrazione delle Informazioni va dunque distinta da quella dei dati (e dei DataBase); per ottenere risultati essa richiede *conoscenza* ed *intelligenza* volte all'individuazione delle sorgenti e dei dati, nonché alla loro fusione e sintesi.
Quando l'Integrazione di Informazioni fa uso di tecniche di Intelligenza Artificiale si parla allora di Integrazione Intelligente di Informazioni (*Intelligent Integration of Information*, I^3). Al contrario delle tecniche tradizionali (che si limitano ad una semplice aggregazione), questa forma di integrazione si prefigge lo scopo di accrescere il valore delle informazioni gestite (ottenendone anche di nuove dai dati utilizzati).

1.1.1 Il programma I^3

Il programma I^3 è un progetto di ricerca, operativo dal 1992, fondato e sponsorizzato dall'ARPA (Agenzia facente capo al Dipartimento della Difesa degli Stati Uniti); l'obiettivo che esso si propone è quello di individuare un'architettura di riferimento che realizzi in maniera automatica l'integrazione delle sorgenti di dati eterogenee [2].
In questo ambito le tecniche di Intelligenza Artificiale, che possono dedurre informazioni dagli schemi delle sorgenti, risultano uno strumento utile e prezioso per la costruzione automatica di soluzioni integrate flessibili e riusabili.

I^3propone l'introduzione di architetture modulari sviluppabili secondo i principi proposti da uno standard che definisca i servizi da includere in qualsiasi integratore, ed abbassi i costi di sviluppo e mantenimento. Questo renderebbe possibile ovviare ai problemi di realizzazione, manutenzione, adattabilità che sorgerebbero nella costruzione premeditata di supersistemi comprendenti una notevole quantità di sorgenti non correlate semanticamente; tali supersistemi risulterebbero inoltre strettamente legati alla risoluzione del problema specifico per cui sono stati implementati.
Se si riesce a riutilizzare la tecnologia già sviluppata, la costruzione di nuovi sistemi risulta più veloce e meno difficoltosa, con conseguente abbassamento dei costi. Per poter sfruttare un'elevata riusabilità bisogna disporre di interfacce ed architetture standard. Il paradigma suggerito per la suddivisione dei servizi e delle risorse nei diversi moduli si articola su due dimensioni:

- l'orizzontale, divisa in tre livelli: livello utente, moduli intermedi che fanno uso di tecniche di IA, risorse di dati;

- la verticale: molti domini, con un numero limitato (e minore di 10) di sorgenti.

I domini nei vari livelli non sono strettamente connessi, ma si scambiano dati ed informazioni la cui combinazione avviene a livello dell'utilizzatore, riducendo la complessità totale del sistema e permettendo lo sviluppo di applicazioni con finalià diverse.

I^3 si concentra sul livello intermedio della partizione, quello che media tra gli utenti e le sorgenti. In questo livello sono presenti vari moduli, tra i quali è giusto evidenziare:

- **Facilitator** e **Mediator** (le differenze tra i due sono sottili ed ancora ambigue in letteratura): ricercano le fonti *interessanti* e combinano i dati da esse ricevuti;
- **Query Processor**: riformula le query aumentando le loro probabilità di successo;
- **Data Miner**: analizza i dati per estrarre informazioni intensionali implicite.

Nell'Appendice A è presente un glossario di termini comunemente usato in ambito I^3, allo scopo di spiegare quei termini che dovessero risultare ambigui o poco chiari, visto il campo recente ed in evoluzione in cui si muove il progetto.

1.1.2 Architettura di riferimento per sistemi I^3

L'obiettivo del programma I^3 è di ridurre il tempo necessario per la realizzazione di un integratore di informazioni, raccogliendo e strutturando le soluzioni prevalenti finora nel campo della ricerca. Esso individua cinque famiglie di attività omogenee, illustrate in Figura 1.1 unitamente ai loro legami. La reciproca iterazione tra queste attività consente di eseguire le operazioni di comunicazione, traduzione ed integrazione dei dati nelle sorgenti. Sono inoltre evidenti i due assi, orizzontale e verticale, che permettono di intuire i diversi compiti dei vari servizi.

Più in dettaglio, percorrendo l'asse orizzontale, si nota il rapporto tra i servizi di Coordinamento ed Amministrazione, che hanno il compito di mantenere informazioni sulle capacità delle sorgenti, vale a dire che tipo di dati sono in grado di fornire e come vanno interrogate; mentre i servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti, forniscono anche funzionalità di supporto.
Lungo l'asse verticale invece viene messo in evidenza come avviene lo scambio di informazioni nel sistema: i servizi di Wrapping provvedono ad estrarre le informazioni dalle sorgenti, queste saranno poi impacchettate ed integrate dai servizi di Integrazione e Trasformazione semantica, per poi essere passate ai servizi di Coordinamento che ne avevano fatto richiesta.

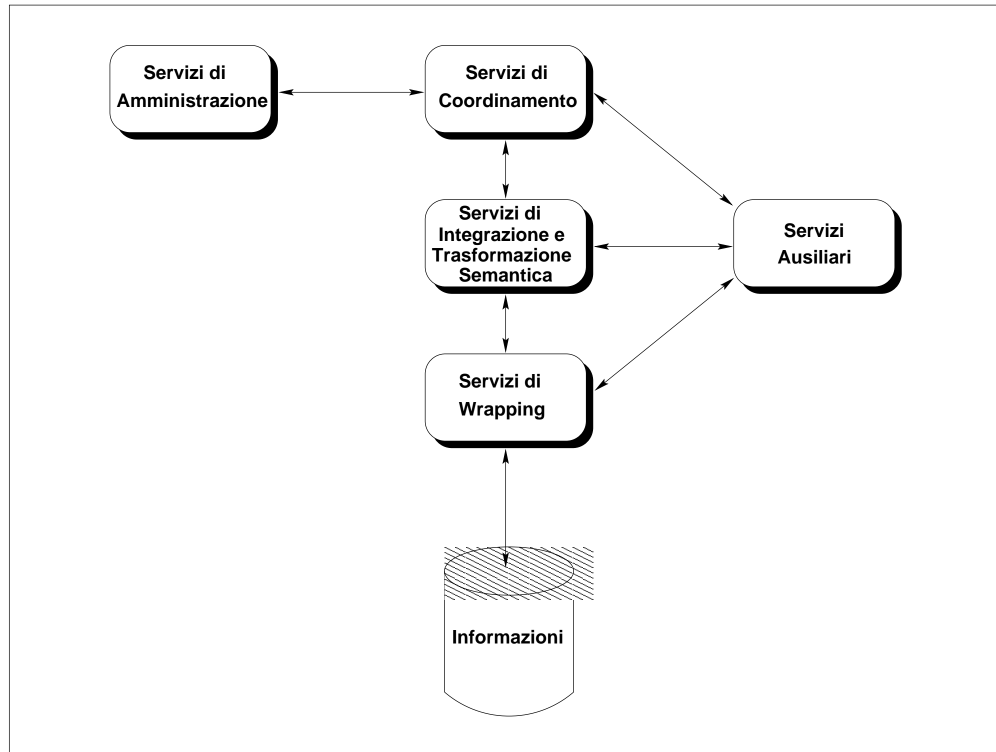

Figura 1.1: Diagramma dei servizi I^3

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 stanno 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.

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 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. Altri moduli interessanti sono:

- **Resource Discovery**: sono in grado di riconoscere e ritrovare gli strumenti che gestiscono una determinata richiesta a run time. Questi servizi sono in grado di acquisire ed aggiornare dinamicamente informazioni sui tool (nomi e servizi forniti) e sul loro stato. Inoltre acquisiscono e mantengono le informazioni sui domini informativi.

- **Iterative Query Formulation**: sono di supporto all'utente in fase di formulazione di query particolarmente complesse sullo schema integrato; specialmente qualora la query formulata non abbia prodotto informazioni interes-

santi, suggeriscono quali condizioni rilasciare, come rendere più specifica o più generale la query.

- **Primitive di costruzione delle configurazioni**: servono a scegliere i servizi di tool e le sorgenti appropriati per svolgere un opportuno task e come collegarli fra loro per creare una configurazione.

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).

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 numero maggiore 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.

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 L'Approccio Semantico all'Integrazione

Il cuore dell'architettura I^3 è rappresentato dai Servizi di Trasformazione ed Integrazione Semantica. Essi si occupano dell'integrazione vera e propria delle informazioni cercando di risolvere le differenze fra gli schemi, i modelli ed i tipi di dati, mentre le differenze dalla prospettiva delle piattaforme (Hw, DBMS, API) sono risolte dai protocolli di rete e dagli standard: SQL, OLE (Object Linking

Embedded), per legare oggetti in applicazioni contenitori, ODBC (Object Database Connectivity): per la connessione fra DB eterogenee, ODMG, che fornisce un modello dei dati ed i linguaggi di descrizione, manipolazione ed interrogazione per basi di dati orientate agli oggetti, CORBA, per lo scambio degli oggetti fra sorgenti eterogenee. Un secondo problema cui si dedica la Semantica riguarda il potere espressivo delle interrogazioni: molti sistemi, specialmente quelli accessibili via WWW, supportano solo un ristretto numero di query predefinite, aggiungendo conoscenza semantica si riesce a ricondurre le query più complesse in quelle predefinite o in un loro sovrainsieme.

L'integrazione degli schemi porta alla definizione di *viste* omogenee: queste rappresentano un superschema, virtuale nella maggior parte dei casi, degli schemi delle sorgenti integrate. Le viste possono essere *convenzionali* se rappresentano in maniera omogenea gli schemi delle sorgenti o parti di esse, altrimenti sono dette *integrate*: in questo caso gli schemi sono fusi e combinati e non è più facilmente riconoscibile il contributo portato dalle diverse sorgenti. La fusione degli schemi può essere completata dalla materializzazione dei dati, come ad esempio si verifica nei data warehouse: in cui i dati sono duplicati e fusi per essere conservati presso l'integratore. Questa scelta presenta vantaggi in termini di velocità di risposta, specialmente quando la fusione dei dati richiede l'esecuzione di join su molti attributi, però diventa pesante il mantenimento della consistenza tra le sorgenti e la vista. In ragione di queste considerazioni si sta diffondendo sempre più l'approccio virtuale: esso consiste nel non replicare i dati, ma nell'eseguire a run-time la decomposizione della query globale, l'interrogazione delle sorgenti e la fusione delle informazioni.

1.2.1 Trattamento dei Metadati e delle Ontologie

Il concetto su cui si basa l'integrazione semantica di sorgenti, autonome nel fine ed eterogenee nella struttura, riguarda la sovrapposizione dei rispettivi domini, che si traduce in corrispondenze fra gli schemi delle sorgenti. Per introdurre i problemi legati al processo di individuazione delle similitudini fra questi schemi introduciamo i concetti di *Metadato* ed *Ontologia*.

Metadati Essi rappresentano informazioni relative agli schemi: significato delle classi (o relazioni) e delle loro proprietà (attributi), relazioni sintattiche e/o semantiche esistenti tra i modelli nelle sorgenti, caratteristiche delle istanze nelle sorgenti: tipi degli attributi, valori null. Utilizzare i metadati può essere di supporto nell'affrontare problematiche relative ad aspetti semantici. Vediamo di approfondire l'argomento. Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare, e quindi un insieme di concetti

comuni, niente assicura che i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti, ne tantomeno le stesse strutture dati. Anzi, visto che le diverse sorgenti sono state progettate e modellate da persone differenti, è molto improbabile che queste persone condividano la stessa concettualizzazione del mondo esterno: in altre parole, non esiste nella realtà una semantica univoca a cui chiunque possa riferirsi.
Come riportato in [3] la causa principale delle differenze semantiche è identificabile nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma non è l'unica. Una medesima realtà può essere rappresentata su DBMS differenti che utilizzano modelli differenti: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate, ad esempio, 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 difficile compito di identificare i concetti comuni all'interno delle sorgenti e risolvere le differenze semantiche che ci possono essere. Possiamo classificare queste incoerenze semantiche in tre gruppi principali:

1. *eterogeneità tra le classi di oggetti*: benchè due classi in due differenti sorgenti rappresentino lo stesso concetto nello stesso contesto, possono usare nomi diversi per gli stessi attributi o per i metodi, oppure avere gli stessi attributi con domini di valori differenti o ancora avere regole diverse sui 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 cioè valori di attributi sono invece parte dei metadati in un altro schema (ad esempio, l'attributo SESSO presente in uno schema può essere assente in un altro schema che rappresenta le persone attraverso le classi 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.

Ontologie Si intende "l'insieme dei termini e delle relazioni esistenti in un dominio per denotare concetti ed oggetti". Un'ontologia relativa ad un insieme di sorgenti è costituita da tutti quei termini che identificano in maniera non ambigua lo stesso concetto o la stessa porzione di conoscenza. Un'ontologia può essere più o meno generale a seconda del livello cui ci si riferisce: se ci si riferisce ad una precisa applicazione l'ontologia sarà limitata dipendendo dal dominio e dall'obiettivo della stessa. Le ontologie di livello superiore sono più vaste riferendosi solo

al dominio ma essendo indipendenti dall'obiettivo, quelle di livello ancora superiore sono indipendenti anche dal dominio e definiscono concetti molto generali. Più precisamente esistono i seguenti livelli di ontologia:

1. *top-level ontology*: descrive concetti molto generali (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 ad 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*: descrive concetti che dipendono sia da un particolare dominio che da un particolare obiettivo.

È lecito supporre che, integrando sorgenti seppur autonome, il livello dell'ontologia sia vincolato all'interno di un certo dominio; questa restrizione è congruente con l'idea di integrare i domini comuni di sorgenti che descrivono campi almeno parzialmente sovrapposti.

Non disponendo delle informazioni sui Metadati e sulle Ontologie risulta impossibile realizzare una buona integrazione. Questa affermazione risulta facilmente comprensibile osservando che, già trattando una sola sorgente, la rappresentazione del dominio per la costruzione ad esempio di un DB sarà fatta in un'infinità di modi diversi a seconda del progettista incaricato, delle informazioni disponibili e della specifica applicazione richiesta; questo fatto risulta ancora più evidente dovendo integrare domini che solo parzialmente sono sovrapposti, che sono stati realizzati da persone diverse, in momenti diversi e con fini del tutto autonomi.

Se dunque si desidera integrare è indispensabile potere disporre di strumenti in grado di dedurre, direttamente dagli schemi, informazioni sui metadati e sulle ontologie. Le metodologie di supporto a questo processo sono basate su tecniche di manipolazione della conoscenza, studiate nell'area dell'intelligenza artificiale (AI): queste tecniche sono in grado di simulare *intelligenza* trasformando problemi di apprendimento in uno equivalente di ricerca. Una volta ricavata questa conoscenza si possono stabilire dei mapping intersorgenti e costruire lo schema integrato.

1.3 Il sistema MOMIS

Recependo l'esigenza di avere ambienti software "$intelligenti$" e funzionali, in grado non solo di fornire accesso a grosse moli di dati ma soprattutto capaci di aumentare la qualità ed il valore delle informazioni ottenibili, i gruppi operativi delle Università di Modena e Reggio e di Milano hanno avviato la realizzazione del sistema **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources). **MOMIS** è iniziato all'interno del progetto **MURST 40%** ***INTERDATA*** (97/98) e continuerà ad essere sviluppato nell'ambito del progetto **MURST D2I**, nel biennio 2000/2001. L'obiettivo del progetto è la realizzazione di uno strumento che, seguendo le linee guida tracciate nell'ambito I^3, permetta la reale integrazione di sorgenti distribuite, eterogenee sia strutturate sia semistrutturate.

Questa tesi fa parte di un progetto di ricerca più ampio che ha come obiettivo la progettazione e realizzazione di un *mediatore* [4].
Secondo la definizione proposta da Wiederhold in [5] "un mediatore è quindi 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.

L'approccio architetturale adottato è stato quello *classico* caratterizzato dalla presenza di tre livelli distinti che interagiscono mediante interfacce standard. La validità di questo approccio è ormai riconosciuta a livello internazionale in quanto permette il conseguimento del necessario grado di astrazione e modularità. I tre livelli che compongono l'architettura sono:

1. utente: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un'unica risposta, come se stesse interrogando una sola sorgente di informazioni;
2. mediatore: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dalle sorgenti. Perché ciò sia possibile verrà impiegato un modello (e quindi un linguaggio di interrogazione) comprensibile da tutte le fonti;

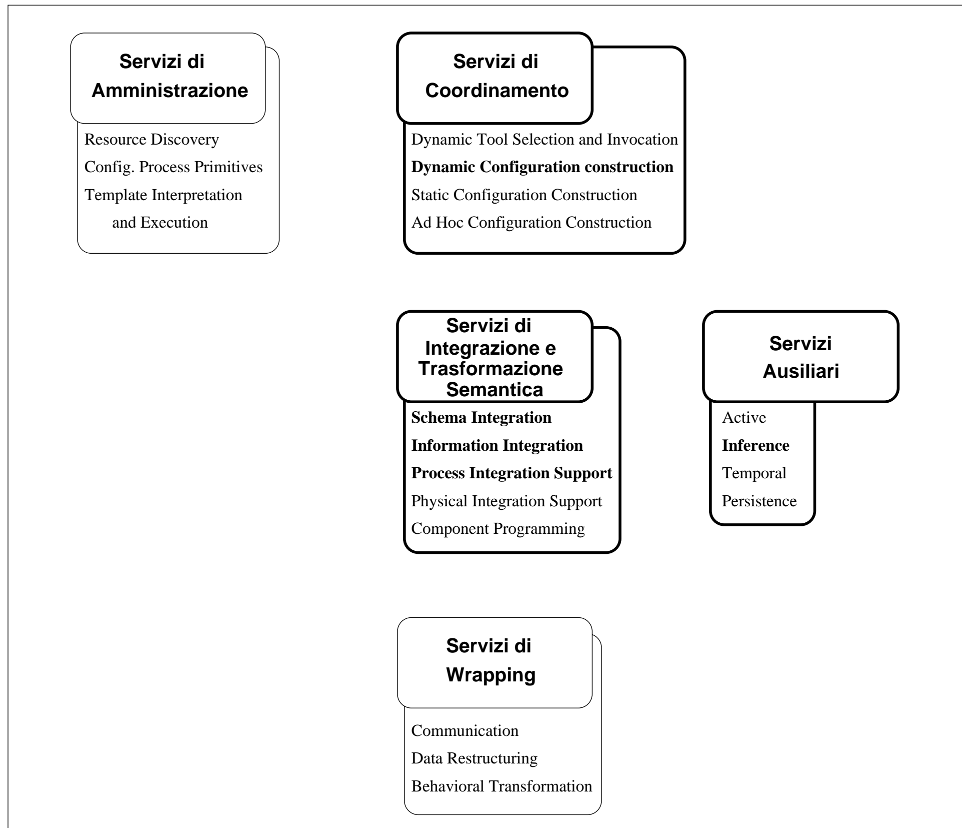

Figura 1.2: Servizi I^3 presenti nel mediatore

3. sorgenti: sono le fonti di informazioni che devono essere integrate dal sistema. Esse costituiscono il livello più basso della struttura e si prevede che possono essere dei database tradizionali (sia ad oggetti, sia basati sul modello relazionale), oppure dei semplici file system. Le sorgenti non saranno accedute in modo diretto, ma verranno gestite da moduli software (Wrapper) in grado di convertire 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 le funzionalità esaminate, in questo e nei precedenti lavori, sono le seguenti:

- servizi di Coordinamento: sul modello di facilitatori e mediatori, il sistema sarà in grado, in presenza di una interrogazione, di individuare automaticamente tutte le sorgenti che ne saranno interessate (Query Decompo-

sition), ed eventualmente di scomporre la richiesta in un insieme di sotto-interrogazioni diverse da inviare alle differenti fonti di informazione (Query Transformation);

- servizi di Integrazione e Trasformazione Semantica: saranno forniti dal mediatore servizi che facilitino l'integrazione sia degli schemi che delle informazioni, nonchè funzionalità di supporto al processo di interrogazione (come può essere la Query Decomposition);

- servizi Ausiliari: sono utilizzate tecniche di Inferenza per realizzare, all'interno del mediatore, una fase di ottimizzazione delle interrogazioni;

Sebbene in un primo momento l'attenzione fosse esclusivamente rivolta alla gestione di sorgenti tradizionali o strutturate, successivamente si è cercato di estendere il contesto applicativo di **MOMIS** in modo da poter trattare anche dati "*multimediali*".
La necessità di gestire dati semistrutturati [6] deriva dall'incredibile aumento dei formati in cui i dati possono essere rappresentati, tuttavia la loro trattazione apre problematiche di complessa soluzione. Questi dati sono infatti caratterizzati dall'avere una struttura estremamente irregolare, non riconducibile ad uno schema preciso, quindi si rende necessario individuare un modello idoneo alla loro trattazione.

1.3.1 Scelte implementative

In letteratura sono stati presentati diversi "*approcci*" al problema dell'integrazione di database convenzionali, come pure applicabili all'integrazione di dati semistrutturati. Per comprendere come MOMIS possa essere collocato rispetto ad altri sistemi esistenti, o in fase di sviluppo, è opportuno fare un'analisi di tali approcci.

Le proposte di integrazione Di tali approcci, seguendo quanto esposto in [7], si può realizzare una prima categorizzazione sulla base del diverso approccio utilizzato per la risoluzione dei conflitti semantici.
Ad un'estremo dello spettro di soluzioni troviamo una proposta di standardizzazione dei database e della rappresentazioni dei dati, come ad esempio vien fatto in SAP [8]. Tale proposta si basa quindi sulla definizione di un *modello globale dei dati* mediante il quale deve essere fatta un reingegnerizzazione dei sistemi locali. Questa strada comporta ingenti investimenti ovviamente ed un forte grado di collaborazione e interazione, presupposti questi che possono essere trovati solo in contesti caratterizzati da un forte accentramento amministrativo.

All'estremo opposto troviamo sistemi che risolvono le eterogeneità presenti nelle sorgenti fornendo strumenti in grado di fornire all'utente esterno una visione omogenea degli schemi e delle informazioni. Questa soluzione preserva quindi l'autonomia delle sorgenti ed è l'unica strada percorribile quando è necessario un elevato grado di indipendenza dei singoli database.

Concentrando l'attenzione sul secondo tipo di approccio è possibile raffinare la classificazione sulla base del modo in cui vengono descritte le sorgenti ed i dati in esse contenuti. Come descritto in [9] è possibile distinguere tra approcci *semantici* e *strutturali*.

Per quanto riguarda l'approccio *strutturale* (e tra questi l'esempio piú importante è senza dubbio costituito dal progetto TSIMMIS [4]) possiamo sottolineare l'impiego di 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. I punti caratterizzanti di tale approccio sono quindi:

- utilizzo di un modello autodescrittivo per trattare tutti i singoli oggetti presenti nel sistema, sopperendo all'eventuale mancanza degli schemi concettuali delle diverse sorgenti;
- inserimento delle informazioni semantiche in modo esplicito attraverso l'impiego di regole dichiarative (ed in particolare, in TSIMMIS, attraverso le MSL rule);
- utilizzo di un linguaggio self-describing che facilita l'integrazione anche e soprattutto di dati semi-strutturati;

Come è facile intuire, in questo modo si ha la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo. Oggetti simili provenienti da una stessa sorgente possono avere strutture differenti, semplificando quindi la trattazione di dati semistrutturati.
D'altro canto, però, l'assenza di schemi concettuali vincola le possibili interrogazioni ad un insieme predefinito dall'operatore (per queste viene preventivamente memorizzato un piano di accesso), limitando in questo modo la libertà di richieste all'utente del sistema ed inoltre non permette, in caso di database di grandi dimensioni, la realizzazione di una ottimizzazio-

ne semantica. L'approccio *semantico* è invece caratterizzato dai seguenti aspetti:

- il mediatore dispone, per ogni sorgente, di uno schema concettuale;
- nello schema concettuale sono presenti oltre ai metadati anche informazioni semantiche che possono essere sfruttate sia nella fase di integrazione delle sorgenti, sia in quella di ottimizzazione delle interrogazioni;
- deve essere disponibile un modello comune per descrivere le informazioni da condividere (e dunque per descrivere anche i metadati);
- viene realizzata un'unificazione (parziale o totale) degli schemi concettuali per arrivare alla definizione di uno schema globale.

Lo *schema globale* sopra citato rappresenta una *vista integrata* delle sorgenti e tale vista può essere realizzata seguendo due approcci distinti, quello *materializzato* e quello *virtuale* [7].
Nella prima soluzione, adottata nei *datawarehouse* [10], le informazioni vengono raccolte in un database centralizzato: le interrogazioni possono quindi essere eseguite senza dover accedere alle sorgenti. Sebbene i tempi di risposta siano decisamente contenuti, la necessità di mantenere l'allineamento tra vista globale e sorgenti impone la definizione e l'impiego di complesse procedure per l'aggiornamento dei dati.
La seconda strategia si basa invece su un modello di *decomposizione* delle query che, analizzando le richieste, porti all'individuazione delle sorgenti "*interessanti*" e alla generazioni di sotto-interrogazioni che possano essere eseguite localmente. Lo schema globale deve poi disporre di tutte le informazioni necessarie alla ricombinazione, o *fusione*, dei dati ricevuti, in modo da ottenere informazioni significative, cioè al contempo complete e corrette.

L'approccio adottato In base alla classificazione fatta possiamo dire che MOMIS segue un approccio *semantico* e *virtuale*. Partendo dagli schemi concettuali locali, con una metodologia *bottom up*, si arriva infatti a definire uno schema globale in grado di fornire un accesso integrato alle sorgenti. Tale schema è quindi arricchito di tutte quelle informazioni che permettono l'individuazione dei dati ed il loro reperimento direttamente dalle fonti di informazione.

Diverse sono le motivazioni che hanno spinto all'adozione di un approccio di questo tipo:

1. la presenza di una schema globale permette all'utente di formulare qualsiasi interrogazione che sia consistente con lo schema;
2. le informazioni semantiche che esso comprende possono contribuire ad una eventuale ottimizzazione delle interrogazioni;
3. l'adozione di una semantica *type as a set* per gli schemi permette di controllarne la consistenza, facendo riferimento alle loro descrizioni;
4. la vista virtuale rende il sistema estremamente flessibile, in grado cioè di sopportare frequenti cambiamenti sia nel numero e tipo di sorgenti, sia nei loro contenuti (non occorre prevedere onerose politiche di allineamento);

Parallelamente a questa impostazione si è deciso di adottare, sia per la rappresentazione degli schemi che per la formulazione delle interrogazioni, un unico modello dei dati basato sul paradigma ad oggetti. Questa scelta è stata fatta per diverse ragioni:

1. la necessità di disporre di un linguaggio di interrogazione espressivo, in grado cioè di rappresentare quei concetti di alto livello fondamentali per l'estrazione di conoscenza da un insieme di dati;
2. la natura stessa degli schemi che utilizzano i modelli ad oggetti, attraverso l'uso delle primitive di generalizzazione e di aggregazione, permette la riorganizzazione delle conoscenze estensionali;
3. ampi sforzi sono sono già stati realizzati per lo sviluppo di standard rivolti agli oggetti: CORBA [11] per lo scambio di oggetti attraverso sistemi diversi; ODMG-93 [12] (e con esso i modelli ODM e ODL per la descrizione degli schemi, e OQL come linguaggio di interrogazione) per gli object-oriented database;
4. l'adozione di una semantica di *mondo aperto* permette il superamento delle problematiche legate all'uso di un convenzionale modello ad oggetti per la descrizione di dati semistrutturati: 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.
5. la possibilità di tradurre, in modo automatico, i modelli ad oggetti in logiche descrittive (ad esempio OLCD) permette l'introduzione di comportamenti intelligenti di supporto all'operatore sia nella fase di integrazione sia in quella di interrogazione.

Queste scelte sono poi state tradotte in un modello dei dati ed in un'architettura.

1.3.2 Il Modello dei dati

Come si è detto all'interno del sistema è stato adottato un modello comune dei dati (ODM$_{I^3}$) di alto livello in modo da facilitare la comunicazione tra i Wrapper ed il Mediatore.
La base di partenza per la definizione di questo modello è rappresentata dalle raccomandazioni relative alla proposta di standardizzazione per linguaggi di mediazione, risultato del lavoro svolto in ambito I^3. Tali raccomandazioni sottolineano la necessità per un mediatore di poter gestire sorgenti con modelli complessi, come quello ad oggetti, e sorgenti molto più semplici come file di strutture: l'impiego di un formalismo il più possibile completo, e quindi in grado di rappresentare in modo adeguato tutte le possibili situazioni, risulta una possibile soluzione.

Per quanto riguarda il linguaggio di definizione degli schemi si è cercato di cogliere le indicazioni emerse in ambito I^3discostandosi, nel contempo, il meno possibile dal linguaggio ODL proposto dal gruppo di standardizzazione ODMG-93. Si è così definito il linguaggio ODL$_{I^3}$ come estensione del linguaggio standard ODL in modo da supportare le necessità del nostro mediatore.

Le principali caratteristiche del linguaggio ODL$_{I^3}$sono:

- possibilità di rappresentare sorgenti strutturate (database relazionali, ad oggetti, e file system) e semistrutturate (XML). Ciò significa che tutte le fonti di informazione, indipendentemente dal modello originario, e lo schema globale verranno descritti mediante il modello comune, facendo quindi riferimento al concetto di classe ed aggregazione (sarà poi compito dei Wrapper provvedere alla traduzione in termini del modello originale);

- dichiarazione di regole di integrità (*if then rule*), definite sia sugli schemi locali (e magari da questi ricevute), che riferite allo schema globale, e quindi inserite dal progettista del mediatore;

- per ogni classe, il wrapper può indicare nome e tipo del sorgente di appartenenza;

- per le classi appartenenti ai sorgenti relazionali è possibile definire le chiavi candidate ed eventuali foreign key;

- dichiarazione di regole di mediazione, o *mapping rule*, utilizzate per specificare l'accoppiamento tra i concetti globali e i concetti locali originali;

- utilizzo della *semantica di mondo aperto*, che permette alle classi descritte di cambiare formato (magari aggiungendo attributi agli oggetti) senza necessariamente cambiarne la descrizione (prerogativa, questa, indispensabile per la gestione di sorgenti semistrutturate);

- il linguaggio supporta la definizione di grandezze locali e di grandezze globali;

- traduzione automatica e trasparente all'utente delle descrizioni nella logica descrittiva **OLCD**, con conseguente possibilitá di utilizzare comportamenti intelligenti nei controlli di consistenza e nell'ottimizzazione semantica delle interrogazioni;

- introduzione dell'operatore di *unione* (`union`), che permette l'espressione di strutture dati in alternativa nella definizione di una classe;

- introduzione del costruttore *optional* (*), specificato dopo il nome di un attributo per indicare la sua natura opzionale;

- dichiarazione di relazioni terminologiche, che permettono di specificare relazioni di sinonimia (SYN), ipernimia (BT), iponomia (NT) e relazione associativa (RT) tra due tipi.

- dichiarazione di relazioni estensionali: Il sistema MOMIS integra gli schemi locali secondo criteri sia intensionali che estensionali. I conflitti intensionali rappresentano quelle incompatibilità derivanti dall'avere porzioni di schemi sovrapposte, ossia gli stessi aspetti del dominio applicativo rappresentati usando strutture differenti (vedi Capitolo 1, Sezione 1.2.1). Ciò che occorre fare è quindi fornire una rappresentazione unificata ed omogenea dei medesimi concetti descritti in sorgenti differenti.
  L'integrazione degli schemi non è però l'unico aspetto che occorre gestire per ottenere un'effettiva integrazione di sorgenti eterogenee: è necessario risolvere anche i conflitti derivanti dalla sovrapposizione delle estensioni, cioè dalla presenza, in sorgenti diverse, di informazione relativa alla stessa entità del "mondo reale". Per arrivare ad un'integrazione corretta può non essere sufficiente fare una semplice unione delle estensioni delle classi, in quanto dati relativi alla stessa entità potrebbero essere presenti in più classi. Questi inconvenienti possono essere risolti soltanto gestendo in modo adeguato le relazioni fra le estensioni. Si introducono, a tale scopo, gli *assiomi estensionali* [13], i quali descrivono le relazioni insiemistiche esistenti fra le estensioni delle sorgenti. Ogni assioma estensionale definito *vincola* le classi coinvolte ad avere anche un legame intensionale. In particolare, facendo riferimento alla traduzione ODL$_{I^3}$ degli assiomi estensionali, abbiamo:

  - **C1** SYN_{Ext} **C2**: le istanze della classe C1 sono le stesse della classe C2; è semanticamente equivalente alle relazioni C1 ISA C2, C2 ISA C1 più una relazione intensionale C1 SYN C2;

- **C1** NT_{Ext} **C2**: le istanze della classe C1 sono un sottoinsieme di quelle della classe C2; è semanticamente equivalente alla relazione C1 ISA C2 più la relazione intensionale C1 NT C2;
- **C1** BT_{Ext} **C2**: le istanze della classe C1 sono un sovrainsieme di quelle della classe C2; è equivalente alla relazione C2 ISA C1 più la relazione intensionale C1 BT C2;
- **C1** $DISJ_{Ext}$ **C2**: le istanze della classe C1 sono diverse da quelle della classe C2; questo assioma non implica nessun tipo di relazione intensionale.

Utilizzando questo linguaggio (vedi Appendice B), il wrapper compie la traduzione delle classi da integrare e le fornisce al mediatore: 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.

1.3.3 L'architettura di MOMIS

Nel sistema MOMIS, che ricordo adotta l'architettura di riferimento I^3 [2], i componenti sono disposti su tre livelli (come si puó vedere da Fig. 1.3):

1. **Livello Dati.** Qui si trovano i **Wrapper**. Posti al di sopra di ciascuna sorgente, sono i moduli che fungono da interfaccia tra mediatore vero e proprio e le sorgenti locali di dati. La loro funzione è duplice:

   - in fase di integrazione, forniscono la descrizione delle informazioni contenute nelle sorgenti. Questa descrizione viene fornita attraverso il linguaggio ODL_{I^3} (descritto in Sezione 1.3.2);
   - in fase di query processing, traducono la query ricevuta dal mediatore (espressa quindi nel linguaggio comune di interrogazione OQL_{I^3}, che è definito in analogia al linguaggio OQL) in una interrogazione comprensibile (e realizzabile) dalla sorgente stessa. Devono inoltre esportare i dati ricevuti in risposta all'interrogazione, presentandoli al mediatore attraverso il modello comune di dati utilizzato dal sistema;

2. **Livello Mediatore.** Il mediatore è il cuore del sistema, in grado di fornire una rappresentazione omogenea delle informazioni ed un accesso integrato alle sorgenti. Esso è composto da due moduli distinti:

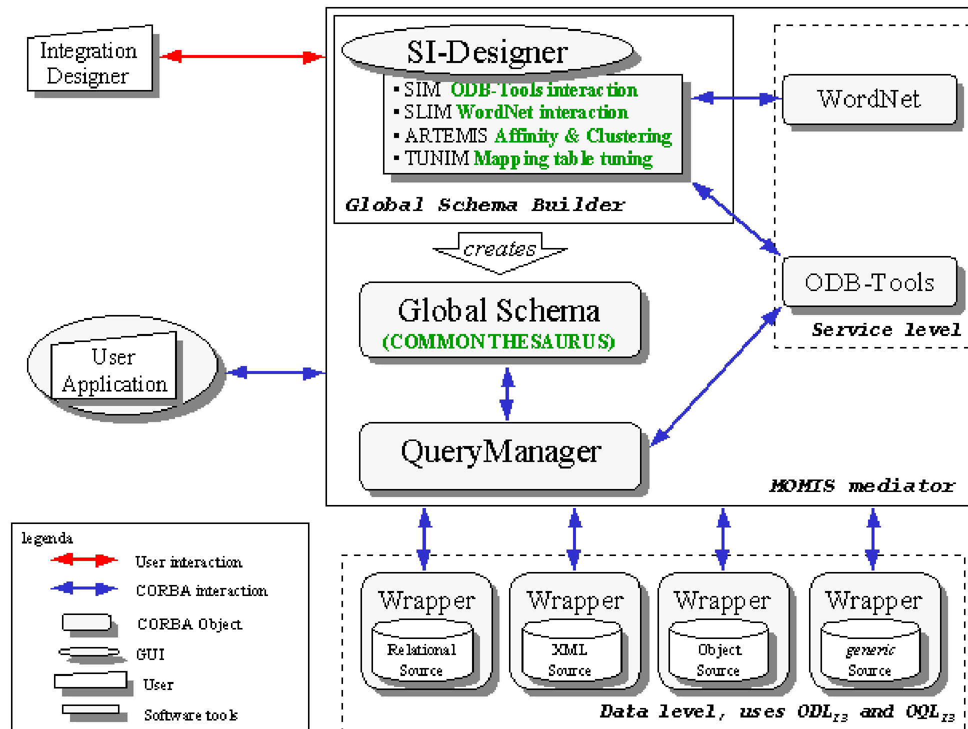

Figura 1.3: Architettura del sistema MOMIS

- Global Schema Builder (GSB): è il modulo di integrazione degli schemi locali che, partendo dalle descrizioni delle sorgenti espresse attraverso il linguaggio 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, fa uso di ODB-Tools (componente esterno sfruttato dal modulo software **SIM**) e del DataBase lessicale WordNet (componente esterno utilizzato di SLIM).
  L'interfaccia grafica del Global Schema Builder è SI_Designer.

- Query Manager (QM): è il modulo di gestione delle interrogazioni. Provvede a gestire la query dell'utente, deducendone da essa un insieme di *sottoquery* da spedire alle fonti locali, ed a ricomporre le informazioni da esse ricevute. Tra i suoi compiti vi è anche l'ottimizzazione semantica delle interrogazioni, realizzata utilizzando ODB-Tools.

3. **Livello Utente.** Il progettista interagisce col Global Schema Builder e crea la vista integrata delle sorgenti: l'utente formula le interrogazioni sullo

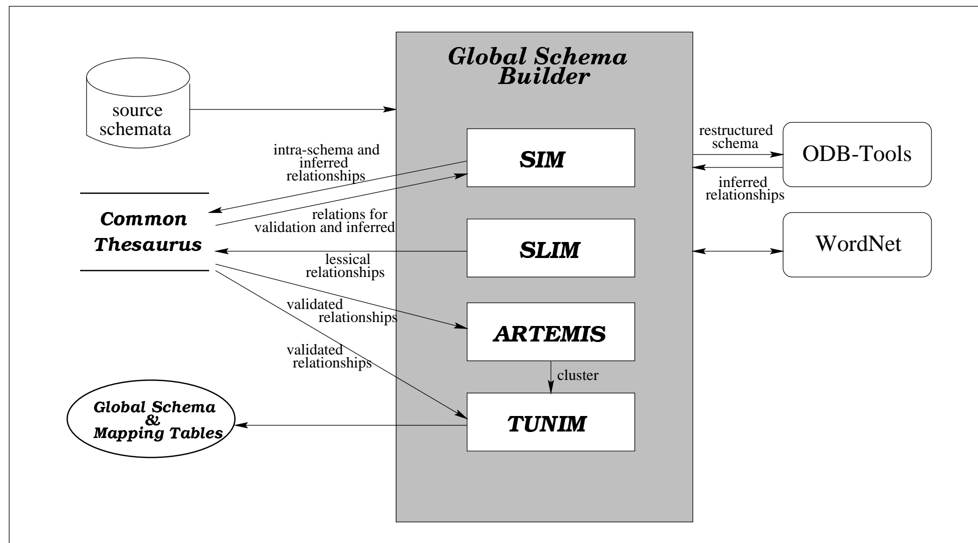

Figura 1.4: Architettura del Global Schema Builder

schema globale passandole come input al Query Manager, che interrogherà le sorgenti e fornirà all'utente la risposta cercata.

Mediante questa struttura e queste funzionalità MOMIS è quindi in grado di fornire un accesso integrato ad informazioni eterogenee memorizzate sia in database di tipo tradizionale (e.g. relazionali, object-oriented) o file system, sia in sorgenti di tipo semistrutturato.

1.3.4 Il Global Schema Builder

L'integrazione delle sorgenti realizzata dal *Global Schema Builder*(GSB) si basa sull'individuazione di un'ontologia, comune alle diverse sorgenti, sotto forma di thesaurus: un insieme di relazioni terminologiche chiamato *Common Thesaurus*. Come mostrato in Figura 1.4, GSB è composto da quattro moduli:

- SIM (Sources Integrator Module): estrae automaticamente le relazioni intra-schema deducibili dalla struttura delle classi ODL_{I^3}; inoltre si occupa della "validazione semantica" e dell'inferenza di nuove relazioni attraverso il componente esterno ODB-Tools.

- SLIM (Schemata Lessical Integrator Module): estrae relazioni intensionali inter-schema tra nomi di attributi e classi ODL_{I^3} utilizzando la conoscenza espressa in WordNet;

- ARTEMIS (Analisys and Reconciliation Tool Environment for Multiple Infomation Sources): è un tool basato su tecniche di affinità e di clustering che esegue l'analisi semantica ed il clustering delle classi ODL$_{I^3}$. L'analisi semantica ha come obiettivo l'identificazione degli elementi legati da relazioni semantiche nei diversi schemi. I coefficienti di affinità sono calcolati tra coppie di elementi per esprimere la loro similarità nel rappresentare la stessa informazione in schemi differenti;

- TUNIM (Tuning of the Mapping-Table): gestisce l'ultima fase di integrazione degli schemi per giungere allo schema globale. Partendo dalle relazioni del Common Thesaurus, per ciascuno dei cluster creati da ARTEMIS viene generata una classe globale. Ciascuna classe è caratterizzata da un insieme di attributi globali e da una mapping-table: l'insieme di attributi ne definisce la struttura mentre la mapping-table indica quali informazioni rappresenta ogni attributo globale.

Per completezza, è bene ricordare che il sistema MOMIS affronta problematiche relative anche alla conoscenza estensionale [13]. A tale proposito è stato progettato il modulo EHB (Extensional Hierarchy Builder), il quale individua ed elabora le relazioni estensionali tra classi locali. Tale modulo è composto da due parti ben distinte: la prima permette al progettista di specificare degli *assiomi estensionali* tra tutte le classi delle sorgenti, la seconda gestisce il calcolo delle *base extension* e della gerarchia estensionale, considerando una classe globale per volta.
Malgrado l'integrazione del suddetto modulo all'interno di SI_Designer non sia ancora stata del tutto completata, è comunque evidente come la prima parte che lo compone dovrà essere inserita nel Global Schema Builder. Ogni assioma estensionale di equivalenza ed inclusione implica, infatti, un legame intensionale molto forte tra le classi coinvolte (vedi Sezione 1.3.2); per tale motivo, due classi tra le quali è stato definito un assioma di questo tipo verranno *forzate* ad appartenere allo stesso cluster. Risulta così evidente la necessità di posizionare questo sottomodulo prima delle fasi di clustering e generazione delle mapping-table (TUNIM).

1.3.5 Esempio di riferimento

In Appendice C è riportata la descrizione ODL$_{I^3}$ delle sorgenti che costituiscono l'esempio di riferimento utilizzato per illustrare le fasi del processo di integrazione. In Figura 1.5 lo stesso viene invece presentato in modo schematico.
Il contesto preso a modello è una realtà universitaria: le sorgenti da integrare sono tre.

La prima sorgente, `University` (UNI), è un DataBase di tipo relazionale, che contiene informazioni sullo staff e sugli studenti di una determinata università; le tabelle che lo compongono sono: `Research_Staff`, `School_Member`,

Sorgente UNIVERSITY (UNI)

```
Research_Staff(name,e_mail,dept_code,section_code)
School_Member(name,faculty,year)
Department(dept_name,dept_code,budget)
Section(section_name,section_code,length,room_code)
Room(room_code,seats_number,notes)
```

Sorgente COMPUTER_SCIENCE (CS)

```
CS_Person(first_name,last_name)
Professor:CS_Person(belongs_to:Office,rank)
Student:CS_Person(year,takes:set⟨Course⟩,rank)
Office(description,address:Location)
Location(city,street,number,county)
Course(course_name,taught_by:Professor)
```

Sorgente TAX_POSITION (TP)

```
Student(name,student_code,faculty_name,tax_fee)
ListOfStudent(Student:set⟨Student⟩)
```

Figura 1.5: Esempio di riferimento

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 (e_mail), e sul corso da lui tenuto (section_code). Per il corso inoltre viene memorizzata l'aula (Room) dove questo si svolge, mentre del dipartimento è descritto, oltre al nome (dept_name) ed al codice (dept_code), il budget (budget) che ha a disposizione. Per gli studenti presenti nella tabella School_Member sono invece mantenuti il nome (name), la facoltà di appartenenza (faculty) e l'anno di corso (year).

La sorgente Computer_Science (CS) contiene invece informazioni sulle persone afferenti a questa facoltà, è un DataBase ad oggetti. Sono presenti sei classi: CS_Person, Professor, Student, Office, Location e Course. I dati mantenuti sono comunque abbastanza simili a quelli della sorgente UNI: per quanto riguarda i professori, sono memorizzati il livello(rank), e l'ufficio 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), l'anno di corso (year) ed il livello raggiunto (rank). Il corso ha poi un attributo complesso che lo lega al professore che ne è titolare (taught_by), mentre per l'ufficio si tiene l'indirizzo (address) e la

descrizione (`description`).

È presente infine una terza sorgente, `Tax_Position` (TP), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (`tax_fee`). A tale sorgente appartengono `Student` e `ListOfStudent`. `Tax_Position` è una sorgente semistrutturata.

Capitolo 2

Gli strumenti utilizzati

2.1 ODB-Tools Engine

Uno degli aspetti più innovativi di MOMIS è rappresentato dall'impiego di logiche descrittive e tecniche di intelligenza artificiale sia in fase di costruzione della vista globale sia nell'ottimizzazione semantica delle interrogazioni.
Questi comportamenti intelligenti sono stati introdotti utilizzando ODB-Tools che è uno strumento software, sviluppato presso il Dipartimento di Ingegneria dell'Università di Modena [19, 21], per la validazione di schemi e l'ottimizzazione semantica di interrogazioni per le Basi di Dati Orientate agli Oggetti (OODB). Gli algoritmi operanti in ODB-Tools sono basati su tecniche di inferenza che sfruttano il calcolo della *sussunzione* e la nozione di *espansione semantica* di interrogazioni per la trasformazione delle query al fine di ottenere tempi di risposta inferiori. Il primo concetto è stato introdotto nell'area dell'Intelligenza Artificiale, più precisamente nell'ambito delle Logiche Descrittive, il secondo nell'ambito delle Basi di Dati. Questi concetti sono stati studiati e formalizzati in **OLCD** (**O**bject **L**anguages with **C**omplements allowing **D**escriptive cycles), una logica descrittiva per basi di dati. Come interfaccia verso l'utente esterno è stata scelta la proposta ODMG-93 [25], utilizzando il linguaggio ODL (Object Definition Language) per la definizione degli schemi ed il linguaggio OQL (Object Query Language) per la scrittura di query. Entrambi i linguaggi sono stati estesi per la piena compatibilità al formalismo **OLCD**.

2.1.1 La Logica Descrittiva OLCD: aspetti generali

Le Logiche Descrittive (DLs) [26, 27] costituiscono una restrizione delle $1^{st}OPL$[28]: esse consentono di esprimere i *concetti*[1] sotto forma di formule

[1] Un concetto è una struttura del tutto simile ad una classe

logiche, usando solamente predicati unari e binari, contenenti solo una variabile (corrispondente alle istanze del concept). Un grande vantaggio offerto dalle DLs, per le applicazioni di tipo DBMS, è costituito dalla capacità di rappresentare la semantica dei modelli di dati ad oggetti complessi (*CODMs*), recentemente proposti in ambito di basi di dati deduttive e basi di dati orientate agli oggetti. Questa capacità deriva dal fatto che, tanto le DLs quanto i CODM, si riferiscono esclusivamente agli aspetti *strutturali*: tipi, valori complessi, oggetti con identità, classi, ereditarietà. Unendo quindi le caratteristiche di similarità coi CODM e le tecniche di inferenza tipiche delle $1^{st}OPL$ si raggiunge l'ambizioso obiettivo di dotare i sistemi di basi di dati di componenti intelligenti per il supporto alle attività di design, di ottimizzazione e, come sarà illustrato nei prossimi capitoli, di integrazione di informazioni poste in sorgenti eterogenee.

Il formalismo **OLCD** deriva dal precedente **ODL**, proposto in [29], che estende l'espressività dei linguaggi sviluppati nell'area delle DLs. La caratteristica principale di **OLCD** consiste nell'assumere una ricca struttura per il sistema dei tipi di base: oltre ai classici tipi atomici *integer*, *boolean*, *string*, *real*, e tipi *mono-valore*, viene ora considerata anche la possibilità di utilizzare dei sottoinsiemi di questi (come potrebbero essere, ad esempio, intervalli di interi). A partire da questi tipi di base si possono definire i *tipi valore*, attraverso gli usuali costruttori di tipo definiti nei *CODMs*, quali *tuple*, *insiemi*, *liste* e *tipi classe*, che denotano insiemi di oggetti con una identità ed un valore associato. Ai tipi può essere assegnato un nome, mantenendo la distinzione tra nomi di *tipi valore* e nomi di *tipi classe*, che d'ora in poi denomineremo semplicemente *classi*: ciò equivale a dire che i nomi dei tipi vengono partizionati in nomi che indicano insiemi di oggetti (*tipi classe*) e nomi che rappresentano insiemi di valori (*tipi valore*). L'ereditarietà, sia semplice che multipla, è espressa direttamente nella descrizione di una classe tramite l'operatore di *intersezione*.
OLCD introduce inoltre la distinzione tra nomi di tipi *virtuali*, che descrivono condizioni necessarie e sufficienti per l'appartenenza di un oggetto del dominio ad un tipo (concetto che si può quindi collegare al concetto di *vista*), e nomi di tipi *primitivi*, che descrivono condizioni necessarie di appartenenza (e che quindi si ricollegano alle classi di oggetti). In [14] **OLCD** è stato esteso per permettere la formulazione dichiarativa di un insieme rilevante di vincoli di integrità definiti sulla base di dati. Attraverso questo logica descrittiva, è possibile descrivere, oltre alle classi, anche le *regole di integrità*: permettono la formulazione dichiarativa di un insieme rilevante di vincoli di integrità sotto forma di regole *if-then* i cui antecedenti e conseguenti sono espressioni di tipo **OLCD**. In tale modo, è possibile descrivere correlazioni tra proprietà strutturali della stessa classe, o condizioni sufficienti per il popolamento di sotto-classi di una classe data. In altre parole le rule costituiscono uno strumento dichiarativo per descrivere gli oggetti che popolano il sistema.

Per la rappresentazione dei vincoli di integrità è stata introdotta una sintassi intuitiva coerente con la proposta ODMG-93. In particolare si sono sfruttati il costrutto `for all`, il costrutto `exists`, gli operatori booleani e i predicati di confronto utilizzati nell'OQL.

Una regola di integrità è dichiarata attraverso la seguente sintassi:

rule `<nome-regola>` **for all** `<nome-iteratore>` **in** `<nome-classe>` **:**
`<condizione-antecedente>`
then `<condizione-conseguente>`

Le `condizioni`, antecedente e conseguente, hanno la medesima forma e sono costituite da una lista di espressioni booleane in `and` tra loro; all'interno di una condizione, attributi e oggetti sono identificati mediante la *dot notation*. La nozione di ottimizzazione semantica di una query è stata introdotta, per le basi di dati relazionali, da King [15, 16] e da Hammer e Zdonik [17]. L'idea di base di queste proposte è che i vincoli di integrità, espressi per forzare la consistenza di una base di dati, possano essere utilizzati anche per ottimizzare le interrogazioni fatte dall'utente, trasformando la query in una *equivalente*, ovvero con lo stesso insieme di oggetti di risposta, ma che può essere elaborata in maniera più efficiente.

2.1.2 OLCD: un Formalismo per Oggetti Complessi e Vincoli di Integrità

Di seguito verrà brevemente descritta la sintassi del linguaggio **OLCD**, mostrando anche come vengono tradotti in tale formalismo le descrizioni ODL_{I^3} degli schemi.

Sia **A** un insieme numerabile di *nomi di attributi* (denotati con a, a_1, a_2, ...), e sia **N** un insieme numerabile di *nomi di tipi* (denotati con N, N_1, N_2, ...). L'insieme **N** include l'insieme $\mathbf{B} = \{$**Integer, String, Real, Bool**$\}$ dei designatori di tipi base (atomici) ed i simboli $\top$, $\bot$ rappresentativi rispettivamente del *tipo universale* e del *tipo vuoto*.

Un *path p* è una sequenza di elementi $p = e_1 \,.\, e_2 \,.\, \ldots \,.\, e_n$, con $e_i \in \mathbf{A} \cup \{\triangle, \exists\}$; con ϵ si indica il path vuoto mentre l'insieme di tutti i *path* è denominato **W**.

S(**A,N**) indica l'insieme di tutte le descrizioni di tipo finite (denotate da S, S_1, S_2, ...), dette brevemente *tipi*, su di un dato **A,N**, ottenuto in accordo con la seguente regola sintattica, dove $a_i \neq a_j$ per $i \neq j$ (nella seguente sequenza p, p_1, p_2, ..., denota un *path*, *d* denota un valore base, θ un operatore relazionale):

$$S \to N|S_1 \sqcup S_2|S_1 \sqcap S_2|\neg S|\{S\}_\forall|\{S\}_\exists|[a_1 : S_1, ..., a_k : S_k]|\triangle S|p\theta d|p \uparrow$$

$\{\}_\forall$ corrisponde al comune costruttore di insieme, [] denota il costruttore di tupla, mentre il costruttore $\{S\}_\exists$ denota un insieme in cui *almeno* un elemento è di tipo S. Il costrutto $\sqcap$ indica la *congiunzione* (intersezione), $\sqcup$ l'*l'unione*, $\triangle$ è il costruttore di oggetto ed il costrutto $\neg$ indica il complemento. Infine $p\theta d$ e $p \uparrow$ rappresentano dei *predicati atomici*: $p\theta d$ definisce una restrizione ad un intervallo, $p \uparrow$ un *path* "indefinito".
Dato un sistema di tipi **S**(**A,N**), uno *schema* σ è una funzione totale da **N** a **S**(**A,N**) che associa ai nomi dei tipi la loro descrizione. σ è partizionata in due funzioni: σ_p, che introduce la descrizione di tipi primitivi la cui estensione dev'essere fornita dall'utente; σ_v che introduce la descrizione di tipi virtuali la cui estensione può essere ottenuta ricorsivamente dall'estensione dei tipi che compaiono nelle loro descrizioni.
In OLCD sono ammessi i *nomi di tipo* ciclici: possono esistere *riferimenti circolari*, ovvero *nomi di tipo* che fanno riferimento a se stessi (direttamente o indirettamente).
Vediamo ora le principali regole di traduzione da ODL$_{I^3}$ ad OLCD.

ODL$_{I^3}$ classes. La traduzione delle classi ODL$_{I^3}$ in OLCD risulta piuttosto semplice: ogni attributo della classe ODL$_{I^3}$ diventa un attributo della corrispondente classe OLCD. Facendo riferimento all'esempio di figura 2.1, la traduzione della classe **FD.Restaurant** sarà la seguente:
$\sigma_p(\mathbf{FD.Restaurant}) = \triangle[r_code : String, name : String, street : String, zip_code : String, pers_id : Integer, special_dish : String, category : Integer, tourist_menu_price : Integer]$

Alcuni aspetti della descrizione ODL$_{I^3}$ delle classi non sono tradotti in OLCD, ma verranno usati nel processo di integrazione semantica delle informazioni (ad esempio la chiave **r_code** della classe Restaurant).

Union constructor. Il costruttore **union** dell'ODL$_{I^3}$ è tradotto in OLCD usando il construtto $\sqcup$. Sempre facendo riferimento alla Figura 2.1, la traduzione della classe **ED.Address** sarà:

$\sigma_p(\mathbf{ED.Address}) = \triangle(dummy : String \sqcup [city : String, street : String, zipcode : String])$

Optional Constructor. Il construtto $\sqcup$ può inoltre essere usato per tradurre gli attributi *opzionali* in OLCD. Infatti, un attributo *opzionale* indica che un valore può o meno esistere per una data instanza. Tutto questo è espresso in OLCD come l'unione tra la descrizione dell'attributo e l'attributo '*non definito*', denotato con l'operatore $\uparrow$. Se consideriamo la descrizione ODL$_{I^3}$ della classe

```
Eating_Source (ED):

 interface Fast_Food                          interface Address
 ( source semistructured ED )                 ( source semistructured ED )
 { attribute string         name;             { attribute string city;
   attribute Address        address;            attribute string street;
   attribute integer        phone*;             attribute string zipcode; };
   attribute set<string>    specialty;         union Address2
   attribute string         category;         { attribute string dummy; };
   attribute Restaurant     nearby*;
   attribute integer        midprice*;
   attribute Owner          owner*; };

  interface Owner
  ( source semistructured ED )
  { attribute string   name;
    attribute Address  address;
    attribute string   job; };


Food_Guide_Source (FD):

 interface Restaurant                            interface Person
 ( source relational FD                          ( source relational FD
   key r_code                                      key pers_id)
   foreign_key(pers_id) references Person )      {  attribute integer      pers_id;
 { attribute string       r_code;                   attribute string       first_name;
   attribute string       name;                     attribute string       last_name;
   attribute integer      category;                 attribute integer      qualification;};
   attribute string       street;
   attribute string       zip_code;
   attribute integer      pers_id;
   attribute integer      tourist_menu_price;
   attribute string       special_dish; };

 interface Bistro
 ( source relational FD                          interface Brasserie
   key r_code                                    ( source relational FD
   foreign_key(r_code) references Restaurant,      key b_code )
   foreign_key(pers_id) references Person)       {  attribute string     b_code;
 { attribute string      r_code;                    attribute string     name;
   attribute set<string> type;                      attribute string     address; };
   attribute integer     pers_id;};
```

Figura 2.1: Gli schemi ODL_{I^3} dell'esempio dei ristoranti

Fast_Food, notiamo la presenza di attributi opzionali (contraddistinti da un *) che vengono così tradotti:

$\sigma_p(\mathbf{ED}.\mathbf{Fast_Food}) = \triangle([name : String, address : ED.Address, specialty : \{String\}, category : String] \sqcap ([phone : Integer] \sqcup phone \uparrow) \sqcap ([nearby : ED.Fast_Food] \sqcup nearby \uparrow) \sqcap ([midprice : Integer] \sqcup midprice \uparrow) \sqcap ([owner : ED.Owner] \sqcup owner \uparrow))$

Vincoli d'integrità. Le regole d'integrità di tipo *if-then* sono integrate in OLCD usando i costrutti $\sqcup$, $\sqcap$ e $\neg$. Per esempio la rule:

```
rule Rule1 forall X in Restaurant :
    (X.category > 5) then X.tourist_menu_price > 100;
```

aggiunta alla descrizione della classe **ES.Restaurant**, viene così tradotta:

$\sigma_p(\mathbf{FD}.\mathbf{Restaurant}) = \triangle([r_code : String, name : String, street : String, zip_code : String, pers_id : Integer, special_dish : String, category : Integer, tourist_menu_price : Integer] \sqcap (\neg(category > 5) \sqcup (tourist_menu_price > 100)))$

Relazioni terminologiche (SYN, NT, BT, RT**).** Non sono tradotte.

Relazioni estensionali (SYN_{ext}, NT_{ext}, BT_{ext}, $DISJ_{ext}$**).** Ogni relazione del tipo C1 ISA C2 è espressa in ODL_{I^3} da una rule del tipo:

```
rule Rule1 forall X in C1 then X in C2
```

ed è integrata nella descrizione della classe C1 usando il costrutto $\sqcap$:

$\sigma_p(C1) = C2 \sqcap ...$

Mapping Rules. Non sono tradotte.

2.1.3 Le regole OLCD e l'espansione semantica di un tipo

Sia il processo di consistenza e classificazione delle classi dello schema, sia quello di ottimizzazione semantica di una interrogazione, sono basati in ODB-Tools sulla nozione di *espansione* semantica di un tipo: l'espansione semantica permette di incorporare ogni possibile restrizione che non è presente nel tipo originale, ma che è logicamente implicata dallo schema (inteso come l'insieme delle classi,

dei tipi, e delle regole di integrità). L'espansione dei tipi si basa sull'iterazione di questa trasformazione: se un tipo *implica* l'antecedente di una regola di integrità, allora il conseguente di quella regola può essere aggiunto alla descrizione del tipo stesso. Le *implicazioni* logiche fra i tipi (in questo caso il tipo da espandere è l'antecedente di una regola) sono determinate a loro volta utilizzando l'algoritmo di *sussunzione*, che calcola relazioni di sussunzione, simili alle relazioni di raffinamento dei tipi definite in [18].
Il calcolo dell'espansione semantica di una classe permette di rilevare nuove relazioni *isa*, cioé relazioni di specializzazione che non sono esplicitamente definite dal progettista, ma che comunque sono logicamente implicate dalla descrizione della classe e dello schema a cui questa appartiene. In questo modo, una classe può essere automaticamente classificata all'interno di una gerarchia di ereditarietà. Oltre che a determinare nuove relazioni tra classi virtuali, il meccanismo, sfruttando la conoscenza fornita dalle regole di integrità, è in grado di riclassificare pure le classi base (generalmente gli schemi sono forniti in termini di classi base).
Analogamente, rappresentando a run-time l'interrogazione dell'utente come una classe virtuale (l'interrogazione non è altro che una classe di oggetti di cui si definiscono le condizioni necessarie e sufficienti per l'appartenenza), questa viene classificata all'interno dello schema, in modo da ottenere l'interrogazione più specializzata tra tutte quelle semanticamente equivalenti alla iniziale. In questo modo l'interrogazione viene spostata verso il basso nella gerarchia e le classi a cui si riferisce vengono eventualmente sostituite con classi più specializzate: diminuendo l'insieme degli oggetti da controllare per dare risposta all'interrogazione, ne viene effettuata una vera ottimizzazione indipendente da qualsiasi modello di costo fisico.

2.1.4 Validazione e Sussunzione

Uno dei problemi principali che il progettista di una base di dati deve affrontare è quello della consistenza delle classi introdotte nello schema. Infatti, molti modelli e linguaggi di definizione dei dati sono sufficientemente espressivi da permettere la rappresentazione di classi inconsistenti, cioè classi che non potranno contenere alcun oggetto della base di dati. Tale eventualità sussiste anche in **OLCD**: ad esempio, la possibilità di esprimere intervalli di interi permette la dichiarazione di classi con attributi omonimi vincolati a intervalli disgiunti. Il prototipo rivela, durante la fase di validazione dello schema, come inconsistente una eventuale congiunzione di tali classi. Il concetto di *sussunzione* esprime invece la relazione esistente tra due classi di oggetti quando l'appartenenza di un oggetto alla seconda comporta necessariamente l'appartenenza alla prima. La relazione di sussunzione può essere calcolata automaticamente tramite il confronto sintattico tra le descri-

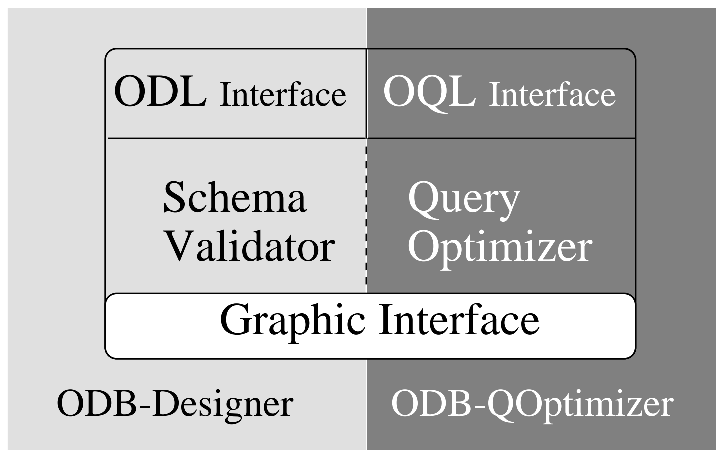

Figura 2.2: Architettura di ODB-Tool

zioni delle classi; l'algoritmo di calcolo è stato presentato in [35]. Poiché accanto alle relazioni di ereditarietà definite esplicitamente dal progettista possono esistere altre relazioni di specializzazione implicite, queste possono essere esplicitate dal calcolo della relazione di sussunzione presenti nell'intero schema: il prototipo, dopo aver verificato la consistenza di ciascuna classe, determina tali relazioni di specializzazione implicite fornendo un valido strumento inferenziale per l'utente progettista della base dei dati.

2.1.5 Architettura di ODB-Tools

ODB-Tool, sviluppato al Dipartimento di Scienze dell'Ingegneria dell'Università di Modena, è un prototipo software per la validazione di schemi e l'ottimizzazione di interrogazioni in ambiente OODB. L'architettura, mostrata in figura 2.2, presenta i vari moduli integrati che definiscono un ambiente *user-friendly* basato sul linguaggio standard ODMG-93. L'utente inserisce gli schemi in linguaggio ODL e le query in OQL ottenendo come risultato la validazione dello schema, l'ottimizzazione dell'interrogazione (in OQL) e la rappresentazione grafica della gerarchia di ereditarietà e di aggregazione dello schema(queste funzionalità sono visibili ed utilizzabili attraverso il sito web *http://www.sparc20.dsi.unimo.it*).

Vediamo in dettaglio la descrizione di ciascun modulo:

- **ODL Interface**
  È il modulo di input degli schemi. Accetta la sintassi ODL e trasforma le classi in descrizioni native del formalismo **OLCD**.

- **OQL Interface**
  È il modulo di input e output delle interrogazioni. Utilizza il linguaggio

OQL sia per l'input che per l'output della query ottimizzata. I predicati booleani in output sono differenziati a seconda del proprio significato:

- i fattori introdotti o modificati dall'ottimizzazione sono mostrati in colore rosso
- i fattori non modificati vengono mostrati in colore grigio
- i fattori ignorati vengono mostrati in colore nero

In output all'ottimizzazione non sono visualizzati i fattori ridondanti, cioè quei fattori identici a quelli descritti nelle classi referenziate dalla query.

- **Schema Validator**
  È il modulo di validazione degli schemi, ottenuta dal calcolo delle relazioni di sussunzione e dei tipi incoerenti e dall'espansione semantica dei tipi. Produce come output un insieme di file utilizzati dagli altri moduli per interpretare e rappresentare i risultati.

- **Query Optimizer**
  È il modulo che genera l'ottimizzazione delle interrogazioni. La query viene inserita come descrizione nativa **OLCD** dal modulo `OQL Interface` e, tramite l'interazione con lo `Schema Validator`, viene ottimizzata calcolandone l'espansione semantica. La query così ottimizzata viene nuovamente inviata all'`OQL Interface` che genera l'output corretto.

- **Graphic Interface**
  È il modulo per la visualizzazione dello schema. Tale rappresentazione è costituita da un grafo i cui nodi rappresentano le classi e gli archi orientati le relazioni di ereditarietà e di aggregazione (opportunamente distinte); per ciascuna classe è possibile visualizzare i nomi ed i domini degli attributi (sia semplici che complessi). Lo schema contiene anche i vincoli di integrità rappresentati ciascuno tramite due classi che specificano l'antecedente ed il conseguente della regola *if-then*. Durante il processo di ottimizzazione la query entra a far parte dello schema con la dignità di classe e di conseguenza viene automaticamente inserita nella gerarchia di ereditarietà.

2.2 Il Database lessicale WordNet

WordNet è un sistema di gestione di un dizionario lessicale basato sulle attuali teorie psicolinguistiche della memoria lessicale umana. Sviluppato sotto la direzione del professore George A.Miller presso l'università di Princeton, è attualmente giunto alla versione 1.6 e conta 129625 lemmi organizzati in 99759 insiemi

di sinonimi (*synset*).
WordNet è oggi considerato la più importante risorsa disponibile per i ricercatori nei campi della linguistica computazionale, dell'analisi testuale, e di altre aree associate.

I dizionari cartacei sono organizzati con un ordinamento alfabetico perché è l'unico modo che permette ad un lettore di trovare le parole cercate sfogliandolo una sola volta. Questo approccio però è lontano dall'essere perfetto, infatti vengono accostate parole con significati diversi e i termini correlati sono sparpagliati in modo casuale. Per ogni lemma vengono presentati tutti i significati assieme, anche se possono non aver nulla in comune, per esempio `number` ha i significati di "cifra" o "quantità", ma anche di "performance teatrale" o di fascicolo di una "rivista".

WordNet non si pone in competizione con i dizionari tradizionali per le informazioni reperibili, per esempio non mostra la sillabazione, la pronuncia, le forme derivate, l'etimologia, le definizioni ed esempi di usi alternativi, note sugli usi speciali, immagini o grafici descrittivi, né per completezza lessicale: il numero di termini lo pone al livello di un dizionario da *college*.
Quello che è innovativo è:

- la comprensione della differenza tra lemmi e significati;
- le relazioni, che permettono navigabilità alle informazioni;
- la strutturazione interna delle categorie sintattiche.

2.2.1 La Matrice Lessicale

Il principio della semantica lessicale è il riconoscimento che esiste una associazione convenzionale tra la *forma* delle parole (il modo in cui sono pronunciate e scritte) e i *concetti* che esse esprimono. La corrispondenza tra la forma delle parole e il loro significato può essere sintetizzata in una tabella, la **Matrice lessicale** (Figura 2.3): l'elemento M_i, j che assume il valore E_i, j indica che la parola F_j può assumere il significato M_i, mentre se M_i, j è vuoto significa che la parola F_j non assume mai il significato M_i.

Dalla figura si nota il tipo di corrispondenza molti-a-molti, da cui emergono due proprietà:

- *Polisemia*: una stessa parola può avere due o più significati. Nella matrice compaiono due o più elementi in colonna, come, ad esempio, accade per la forma F_2;

Word Meanings	Word Forms				
	F_1	F_2	F_3	·······	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
⋮				⋱	
M_m					$E_{m,n}$

Figura 2.3: La Matrice Lessicale

- *Sinonimia*: concetto che può avere due o più parole in grado di esprimerlo. Nella matrice compaiono due o più elementi in riga, ad esempio in corrispondenza di M_1.

La *Polisemia* e la *Sinonimia* costituiscono problemi lessicografici tra loro complementari. Durante una qualsiasi forma di comunicazione, il ”destinatario” riceve la parola, cercando di capirne il significato tra tutti quelli che la stessa può esprimere, il ”mittente” conosce il concetto che vuole esprimere e si trova nella situazione di dover scegliere una fra le varie forme che possono esprimerlo.

WordNet rappresenta i concetti seguendo la teoria cosiddetta *differenziale*, secondo cui i diversi significati non sono necessariamente denotati da definizioni scritte, ma vengono rappresentati e distinti tra loro attraverso l'uso di differenti simboli conosciuti dall'utente, ai quali l'utente stesso associa il concetto. L'utente deve quindi conoscere sufficientemente bene la lingua (nella fattispecie l'inglese), e pertanto essere in grado di riconoscere il significato di una determinata parola in base ai vari sinonimi ad essa associati (senza che ne venga data esplicitamente una definizione).

2.2.2 Tipi di relazioni

Nel Database lessicale WordNet ogni categoria sintattica (nomi, verbi, avverbi ed aggettivi)è organizzata in insiemi di sinonimi (chiamati *synset*) che rappresentano un concetto lessicale. Non si vedranno mai accostate nel medesimo *synset*, parole appartenenti a categorie differenti.

Il database lessicale collega i termini in base a relazioni semantiche. Poiché una relazione semantica è una relazione fra significati, e considerato che i signi-

ficati, a causa della sinonimia, sono associati a set di termini sinonimi, è naturale pensare alle relazioni semantiche come a relazioni tra insiemi di sinonimi. Da ciò emerge una distinzione tra la relazione di sinonimia e le altre relazioni semantiche, denotata anche dal fatto che ogni insieme di termini sinonimi è racchiuso fra parentesi graffe {}, mentre gli insiemi prodotti da tutte le altre relazioni sono racchiusi fra parentesi quadre [ ].

Le relazioni semantiche godono di due proprietà:

- Se esiste una relazione R fra gli insiemi $\{x,x', ...\}$ e $\{y,y', ...\}$, allora deve esistere una relazione inversa R' fra $\{y,y', ...\}$ e $\{x,x', ...\}$. R e R' possono, non necessariamente, coincidere.
- Se esiste una relazione R fra gli insiemi $\{x,x', ...\}$ e $\{y,y', ...\}$, allora vale $[x\ R\ y]$, $[x\ R\ y']$, ... , $[x'\ R\ y]$, etc...

Vediamo i principali tipi di relazioni che sono codificate in WordNet:

- **Sinonimia.** *Due termini si definiscono sinonimi se possono essere indifferentemente scambiati senza che, nel contesto in cui si trovano, il significato cambi.*
  É opportuno osservare che nella realtà sono pochi i sinonimi in senso stretto: più comunemente si parla di sinonimi riferiti ad un particolare contesto. La sinonimia tra termini è la relazione più importante, perché ogni insieme *synset* che ne scaturisce rappresenta la semantica di un concetto.
- **Antinomia.** *Due termini sono in relazione di antonimia se uno è il contrario dell'altro.*
  Da osservare che l'antinomo del termine x non sempre coincide con '*non* x': ad esempio 'gioioso' e 'triste' sono in relazione di antinomia, ma 'non triste' non coincide con 'gioioso', dal momento che esistono una serie di stati d'animo intermedi tra i due.
  Tra le relazioni citate, l'antinomia è l'unico tipo di relazione lessicale che si applica fra singoli termini e non fra concetti da *synset*.
  Ad esempio non si può affermare che {rise,ascend} e {fall,discend} siano antinomi, pur essendolo singolarmente [rise/fall] (e anche [ascend/discend]).
- **Iponimia.** *Un concetto è iponimo di un altro quando lo specializza: tra i due esiste un rapporto di tipo ISA.*
  Ad esempio 'studente' è iponimo di 'persona' e 'persona' è a sua volta iponimo di 'essere vivente'. L'iponimia gode della proprietà transitiva: nell'esempio si deduce che 'studente' è iponimo di 'essere vivente'. Questa proprietà consente la costruzione di sistemi ereditari, gerarchie nelle quali

ogni concetto iponimo eredita tutte le caratteristiche del suo superconcetto e ne aggiunge almeno una che lo distingue dallo stesso superconcetto e da qualsiasi suo altro iponimo. Inoltre, l'iponimia è una relazione asimmetrica: la sua relazione duale è **Ipernimia**.

- **Olonimia.** *La Olonimia è una relazione semantica che si esprime fra due concetti x e y ("x olonimo di y") quando y* **is a part of** x.
  Per esempio, 'riempire' è olonimo di 'versare' perché certamente il concetto di "riempire qualcosa" implica il concetto di "versare qualcosa", ma "versare qualcosa" da solo non basta a rappresentare il concetto di riempimento (per 'riempire' qualcosa bisogna 'versare' qualcos'altro in un recipiente), cioè il concetto di riempimento è un concetto composto e 'versare' è solo una parte (*is a part of*) di tale concetto.
  Come la relazione precedente, anche la Olonimia gode della proprietà transitiva, ed è asimmetrica: la relazione duale è **Meronimia**.
  Come nel caso precedente si possono realizzare gerarchie di concetti olonimi/meronimi. In questo caso però uno stesso meronimo può avere più olonimi: uno stesso componente può contemporaneamente far parte di differenti concetti composti.

- **Correlazione.** *Due termini sono correlati quando condividono uno stesso ipernimo.* Questa relazione dunque è indiretta poiché è derivata da altre relazioni.

Nel database implementato da WordNet, l'insieme di tutte le relazioni tra le parole, dei diversi tipi appena descritti, formano una rete complessa. In questo modo, secondo la teoria *differenziale* adottata, il significato di una parola data può essere determinato in base alla collocazione che la stessa ha all'interno della rete.

Capitolo 3

Il processo di integrazione di sorgenti in MOMIS

3.1 Generazione del Thesaurus Comune

Nell'ambito di un approccio semantico all' integrazione di dati (quale è quello adottato nel progetto **MOMIS**) è di fondamentale importanza la conoscenza delle informazioni semantiche relative al contesto e alla struttura dei vari schemi sorgenti. Tale conoscenza è contenuta nel cosiddetto *Common Thesaurus*, un dizionario all'interno del quale sono presenti un insieme di relazioni terminologiche che legano tra loro classi ed attributi. Tali relazioni sono di tipo intensionale ed esprimono la conoscenza inter-schema e intra-schema riguardo gli schemi delle sorgenti in esame; esse sono di tre tipi:

- SYN (SYNonym-of, *relazioni di sinonimia*): definita tra due termini che possono essere scambiati nelle sorgenti senza modificare il concetto del mondo reale rappresentato. Esempio: `faculty SYN faculty_name`.

- BT (Broader-Term, *relazioni di specializzazione)*: definita tra due termini t_i e t_j tali che t_i ha un significato più generale di t_j; NT (Narrower-Term) è la relazione opposta di BT. Esempio: `CS_Person BT Professor`, che è equivalente a `Professor NT CS_Person`.

- RT (Related-Term, *relazioni di aggregazione*): definita tra due termini t_i e t_j che sono generalmente usati nello stesso contesto e tra i quali esiste un legame generico (non meglio specificato). Esempio: `Research_Staff RT Department`.

La costruzione del *Common Thesaurus* è un processo incrementale durante il quale vengono aggiunte relazioni secondo il seguente ordine:

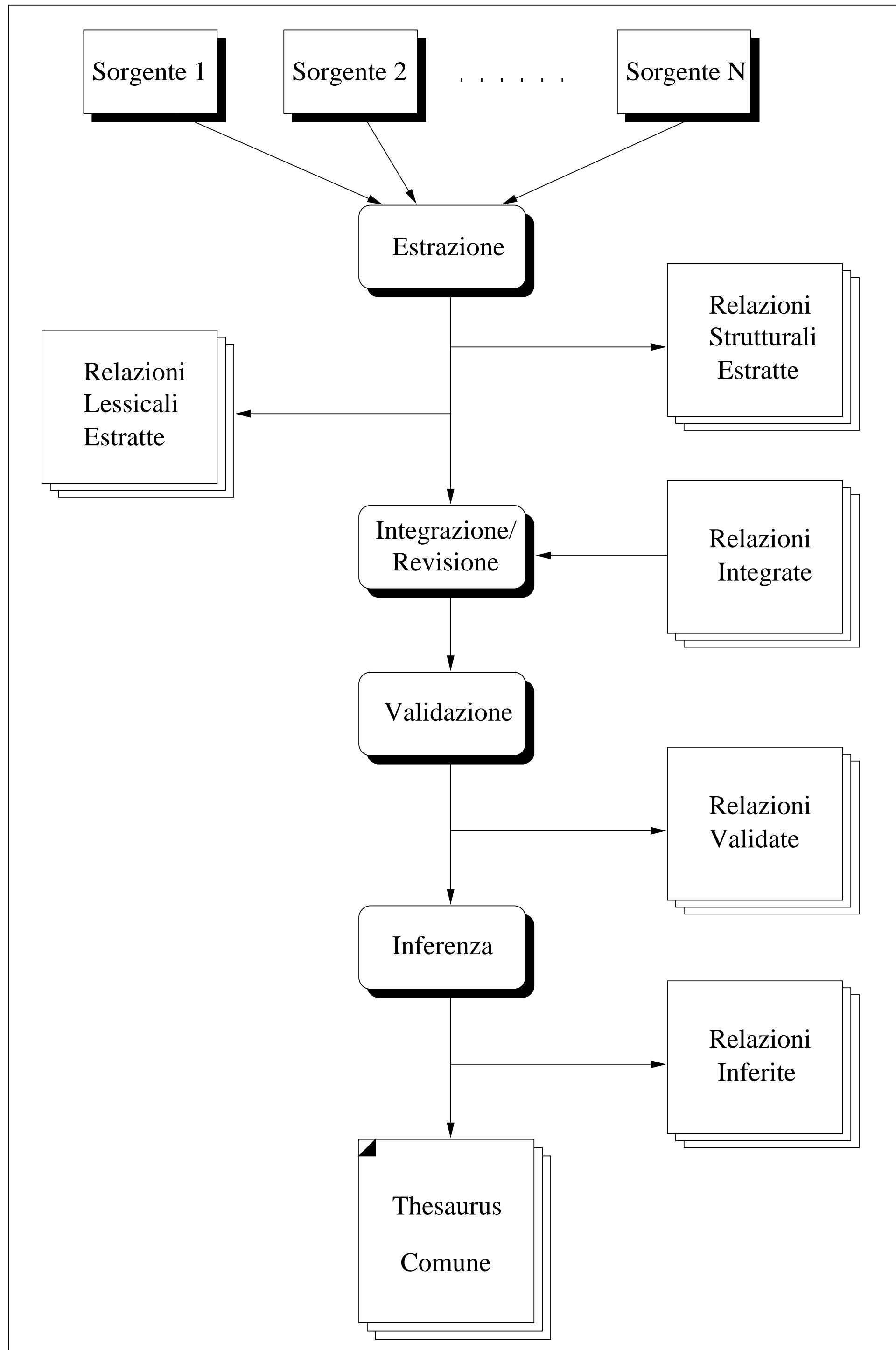

Figura 3.1: Il processo di generazione del Thesaurus Comune

1. *relazioni semantiche intra-schema*

2. *relazioni lessicali*

3. *relazioni aggiunte dal progettista*

4. *relazioni intensionali inferite*

In Figura 3.1 sono evidenziate le fasi del processo che porta alla generazione del Thesaurus Comune.

Lo sforzo compiuto a questo livello è quello di rendere il processo il più possibile automatico, minimizzando l'intervento del progettista, la cui partecipazione si rende comunque indispensabile. Nei paragrafi successivi verranno descritte tutte le fasi del processo di integrazione; l'esempio a cui si fará riferimento è riportato in appendice C.

3.1.1 Acquisizione degli schemi ODL$_{I^3}$

Prima di descrivere nel dettaglio le fasi del processo di integrazione degli schemi, è opportuno mostrare come gli stessi vengano acquisiti da SI_Designer. Il modulo preposto a tale compito è denominato SAM (Sources Acquisition Module) e la sua interfaccia grafica (che rappresenta anche la schermata iniziale di SI_Designer) è mostrata in Figura 3.2.
Il modulo SAM deve popolare le strutture dati atte a memorizzare le informazioni sugli schemi locali: nella fattispecie le proprietà della classe **GlobalSchemaProxy**. Tale operazione viene fatta attraverso il parser ODL$_{I^3}$. Facendo riferimento all'interfaccia grafica di Figura 3.2, di seguito verrà descritto come avviene l'acquisizione di uno schema ODL$_{I^3}$ da un wrapper.

Per prima cosa occorre indicare al tool le informazioni per accedere all'oggetto wrapper, vale a dire: il nome dell'oggetto wrapper, l'indirizzo Internet della macchina sulla quale *gira* il wrapper e la porta di accesso. Tutte queste informazioni possono essere inserite semplicemente digitandole negli appositi campi posti in basso a destra. Una volta fornite tutte le informazioni si avvia la procedura di acquisizione dello schema premendo il pulsante ADD posto in basso a destra.
Il modulo SAM esegue l'acquisizione secondo i seguenti passi:

1. *controllo dell'esistenza del wrapper indicato*: il modulo richiede all'ORB un object-reference per il wrapper CORBA e, nel caso l'object reference venga fornito, esegue un test per controllare se il wrapper è attivo. Il test consiste nell'invocare un metodo del wrapper: in caso di fallimento viene visualizzato un messaggio di errore;

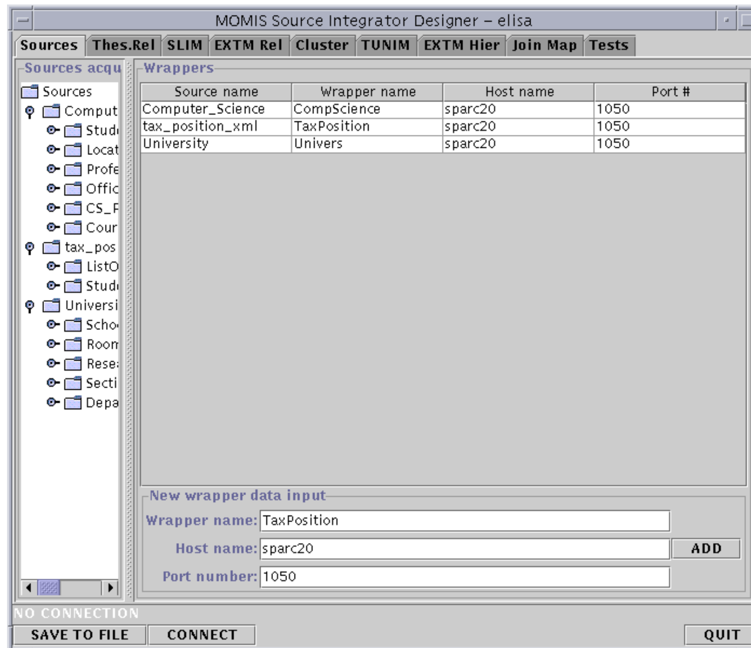

Figura 3.2: L'interfaccia grafica di MOMIS

2. *acquisizione dello schema ODL$_{I^3}$*: viene eseguito il *parsing* dello schema ODL$_{I^3}$ fornito dal wrapper sottoforma di stringa. Nel caso in cui si siano verificati errori sintattici o semantici viene visualizzato un messaggio di errore, e l'acquisizione è come se non fosse iniziata;

3. *aggiunta della nuova acquisizione alle precedenti*: le informazioni sul nuovo schema vengono aggiunte a quelle degli eventuali schemi acquisiti in precedenza. Se il modulo rileva che era stato acquisito lo schema di una sorgente avente lo stesso nome di quella descritta nello schema appena acquisito, viene richiesto al progettista di cambiare nome alla sorgente prima di mettere assieme le informazioni degli schemi.

Dopo l'acquisizione di ogni schema, il modulo provvede all'aggiornamento dell'interfaccia grafica che comprende:

- l'elenco delle sorgenti, delle classi e degli attributi organizzati in un albero in cui gli attributi sono le foglie (sulla sinistra della Figura 3.2);

- la tabella dell'elenco delle sorgenti acquisite con le informazioni sul relativo wrapper che le gestisce.

3.1.2 Estrazione delle relazioni intra-schema

La struttura di ogni schema locale contiene delle informazioni semantiche ricavabili dalle gerarchie di ereditarietà e aggregazione. Terminata la fase di acquisizione degli schemi locali, il sistema è in grado di estrarre in modo automatico alcune relazioni. Il modulo che realizza questa fase è **SIMA** (vedi capitolo successivo, Sezione 4.2).
Da un'analisi delle sorgenti relazionali esso estrae le relazioni terminologiche definite dalle foreign key: ogni volta in cui una classe definisce una chiave esterna, è possibile dedurre una relazione RT fra la classe stessa e quella riferita. Talvolta è possibile, estendendo l'analisi a tutte le chiavi delle classi coinvolte, estrarre delle relazioni di ereditarietà o sinonimia: la trattazione completa di tutti i casi sarà discussa nel capitolo successivo (Sezione 4.2).
L'analisi delle sorgenti ad oggetti invece produce:

- relazioni BT/NT ogni volta che c'è ereditarietà tra classi,
- relazioni RT quando si è in presenza di gerarchie di aggregazione.

Esempio 1 Dall'esempio di riferimento, si ricavano le seguenti relazioni terminologiche intra-schema:

⟨University.Department RT University.Research_Staff⟩
⟨University.Section RT University.Research_Staff⟩
⟨University.Room RT University.Section⟩
⟨Computer_Science.Student NT Computer_Science.CS_Person⟩
⟨Computer_Science.Professor NT Computer_Science.CS_Person⟩
⟨tax_position_xml.Student RT tax_position_xml.ListOfStude⟩
⟨Computer_Science.Course RT Computer_Science.Student⟩
⟨Computer_Science.Office RT Computer_Science.Professor⟩
⟨Computer_Science.Location RT Computer_Science.Office⟩
⟨Computer_Science.Professor RT Computer_Science.Course⟩

3.1.3 Estrazione delle relazioni inter-schema

Le relazioni terminologiche inter-schema sono estratte analizzando gli schemi ODL$_{I^3}$ nel loro insieme. La loro estrazione è basata sulle relazioni lessicali che sussistono tra nomi di classi ed attributi, derivanti dai significati delle parole usate: un tipo di conoscenza non esplicitata tramite costrutti di un linguaggio di definizione dei dati, ma impliciti nel nome assegnato dal progettista. È compito del progettista attribuire nomi descrittivi o che possano essere interpretati correttamente; per cui c'è un'incertezza di interpretazione insita nell'ambiguità del linguaggio. Qualunque conoscenza associata ai nomi dello schema è un'opportunità

che dev'essere presa in considerazione. Essendo impossibile compiere questo lavoro manualmente, diventa indispensabile l'uso di un database lessicale, quale WordNet (vedi capitolo precedente, Sezione 2.2).

Le relazioni trovate utilizzando WordNet sono sia intra-schema che inter-schema. Nel caso di sorgenti ad oggetti e, meno, nel caso di sorgenti relazionali, se gli schemi sono stati strutturati bene, non dovrebbero venir trovate relazioni intra-schema che **SIMA** non abbia già inserito nel Thesaurus alla fase precedente. Comunque le relazioni di maggior valore che vengono trovate da questa fase sono quelle inter-schema che non possono venir trovate da **SIMA**.

Le relazioni derivanti da WordNet vengono proposte come relazioni semantiche da inserire nel *Common Thesaurus* in base alla seguente corrispondenza:

Sinonimia: corrisponde ad una relazione SYN.

Iperonimia: corrisponde ad una relazione BT.

Olonimia: corrisponde ad una relazione RT.

Correlazione: corrisponde ad una relazione RT.

Partendo dagli schemi da integrare, il progettista deve fissare un'associazione tra ogni nome (di classe o di attributo) ed il significato utilizzato nel contesto. Cioè, dato un nome, si devono scegliere i significati ad esso associati.
Tale scelta è eseguita in due fasi:

1. **Scelta della forma base.** In tale scelta il progettista è assistito dal sistema che gli propone la forma base (word form) usando il processore morfologico presente in WordNet. Per forma base si intende la parola tolti suffissi dovuti alla declinazione o coniugazione.

2. **Scelta del significato.** Il progettista può decidere di far corrispondere ad un nome zero, uno o più significati. Ad esempio in Figura 3.3 per la forma base `address` ottengo tutti i 15 significati che WordNet le attribuisce tra cui scegliere quello calzante con il contesto.

   Alla fine della procedura di scelta dei significati, il progettista preme il pulsante "build" e vengono calcolate le relazioni (vedi Figura 3.4).

   Da queste il progettista può scartare quelle sbagliate o fuorvianti; le altre vengono accettate ed immesse nel *Common Thesaurus* come relazioni semantiche intensionali.

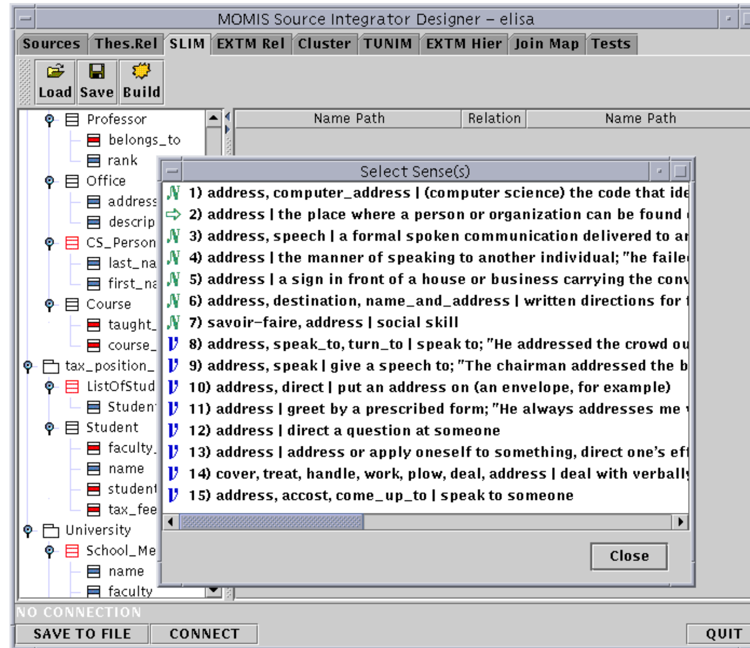

Figura 3.3: Significati di `address`

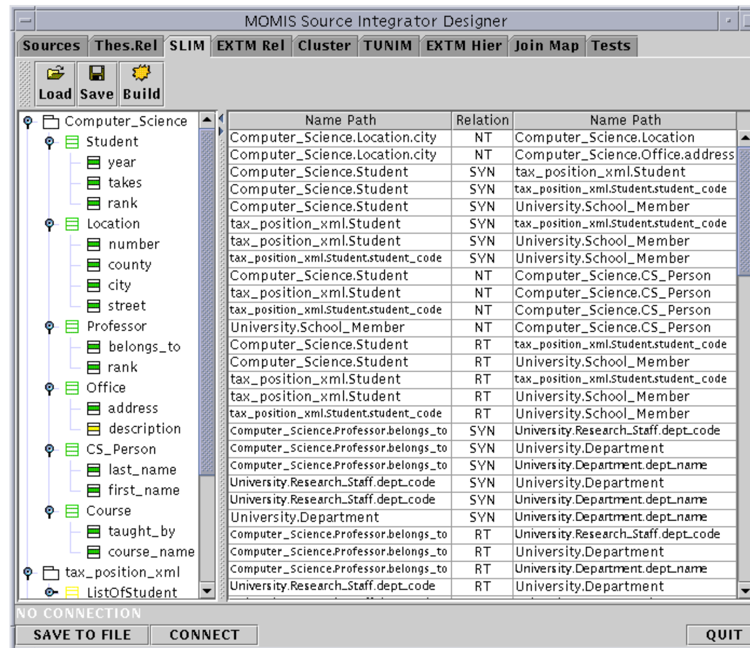

Figura 3.4: Relazioni intensionali ottenute dopo SLIM

3.1.4 Integrazione dell'insieme di relazioni

Nuove relazioni possono essere fornite direttamente dal progettista per aggiungere al Common Thesaurus una conoscenza specifica non ricavata automaticamente nelle fasi precedenti. In particolare il progettista può inserire sia relazioni intensionali, che assiomi estensionali. Ogni assioma estensionale vincola le *estensioni* delle classi coinvolte e genera una relazione intensionale da inserire nel Common Thesaurus (vedi Capitolo 1, Sezione 1.3.2).
Questa è un'operazione cruciale poichè le nuove relazioni vengono forzate ad entrare nel Common Thesaurus, contribuendo così alla creazione dello schema globale. Ciò significa che se viene inserita una relazione errata, si può giungere ad uno schema globale errato. La successiva fase di validazione delle relazioni fra attributi, unitamente a quella di validazione delle relazioni tra classi, rappresenta un supporto per evitare di incombere in una situazione di questo tipo.

Esempio 2 In relazione all'esempio di riferimento, le relazioni inserite dal progettista sono:

⟨University.Room SYN Computer_Science.Location⟩
⟨University.Room SYN Computer_Science.Office⟩

3.1.5 Validazione delle relazioni tra attributi

Le relazioni tra attributi inserite nei passi precedenti nel *Common Thesaurus* devono essere esaminate per verificare la compatibilità dei loro domini; tale fase è detta *validazione* e viene svolta dal modulo **SIM** (in particolare da **SIMB**) attraverso ODB-Tools. La descrizione dettagliata di tale fase sarà riportata nel capitolo successivo (Sezione 4.3.1); qui anticipo semplicemente le regole sulle quali si basa la procedura:

- ogni relazione di sinonimia è validata se i domini dei due attributi coinvolti sono equivalenti, o se uno dei due è più specializzato dell'altro;
- ogni relazione di specializzazione è validata se i domini dei due attributi sono equivalenti, oppure se il dominio dell'attributo più generale comprende il dominio dell'altro.

Esempio 3 In relazione all'esempio di riferimento, si riportano qui sotto alcune relazioni tra attributi che hanno subito la validazione ('[1]' significa relazione valida mentre '[0]' indica il contrario):

⟨Computer_Science.CS_Person.first_name NT tax_position_xml.Student.name⟩ [1]
⟨Computer_Science.CS_Person.last_name NT tax_position_xml.Student.name⟩ [1]
⟨Computer_Science.Course.course_name SYN University.Research_Staff.section_name⟩ [1]
⟨University.Research_Staff.dept_code BT Computer_Science.Professor.belongs_to⟩ [0]
⟨University.School_Member.faculty SYN tax_position_xml.Student.faculty_name⟩ [1]

La quarta relazione non è stata validata in quanto il dominio di `dept_code` è un `integer` mentre quello di `belongs_to` coincide con la classe `Computer_Science.Office`.

3.1.6 Validazione delle relazioni tra classi ed inferenza di nuove relazioni intensionali

Viene ora creato uno schema virtuale che descrive tutta la conoscenza semantica contenuta fino a questo momento nel Common Thesaurus. Il nuovo schema verrà inviato ad ODB-Tools, il quale eseguirà un controllo di consistenza degli schemi (fase di validazione delle relazioni tra classi) ed inferirà nuove relazioni semantiche da inserire nel Common Thesaurus (vedi Capitolo 4, Sezione 4.3.2).

Esempio 4 In relazione all'esempio di riferimento, le relazioni inferite sono:

⟨`tax_position_xml.ListOfStudent` RT `University.School_Member`⟩
⟨`Computer_Science.Course` RT `University.School_Member`⟩
⟨`Computer_Science.Professor` RT `University.Room`⟩
⟨`Computer_Science.Student` RT `University.Section`⟩
⟨`Computer_Science.Professor` RT `University.Section`⟩
⟨`Computer_Science.Course` RT `tax_position_xml.Student`⟩
⟨`tax_position_xml.ListOfStudent` RT `Computer_Science.Student`⟩
⟨`Computer_Science.Professor` RT `University.Location`⟩

Si può notare come tali relazioni siano inter-schema , ovvero definite tra classi appartenenti a sorgenti diverse.

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

Per realizzare una integrazione degli schemi ODL$_{I^3}$ delle differenti sorgenti in uno schema globale, abbiamo bisogno di tecniche per identificare le classi che descrivono le stesse informazioni (o comunque informazioni semanticamente equivalenti), e che sono localizzate all'interno di sorgenti diverse. A questo scopo, 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, delle classi vengono analizzate 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*.

La valutazione dei coefficienti di affinità si basa sulle relazioni terminologiche memorizzate nel Thesaurus. A questo scopo, il Thesaurus viene organizzato in una struttura simile alle Associative Networks [19], 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, e nelle sperimentazioni precedentemente realizzate presso l'Università di Milano, si è adottato $\sigma_{syn} = 1$, $\sigma_{bt} = \sigma_{nt} = 0.8$ e $\sigma_{rt} = 0.5$.

Definizione 1 (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.

Definizione 2 (Termini Affini) Due termini t_i, t_j si dicono *affini*, e si denotano con $t_i \sim t_j$, se la loro Funzione dei Affinità restituisce un valore maggiore o uguale ad un predefinito valore di soglia $\alpha > 0$, cioè:

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

3.2.1 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 3 (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:

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

Definizione 4 (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:

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

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

$$\text{C} = \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), n_t \sim n_q\}$$

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.
Il termine F_c realizza un controllo sui domini degli attributi coinvolti nella relazione da esaminare, 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 numero di relazioni validabili memorizzate nel Thesaurus tra attributi delle due classi, e numero di queste relazioni che sono state validate.
In questo modo, maggiore sarà il numero di attributi affini tra le due classi, e maggiore il numero di controlli positivi, più alto risulterà il valore dello *Structural Affinity Coefficient*.

Definizione 5 (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 $w_{NA} = w_{SA} = 0.5$.
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.

3.3 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.

Questa procedura di clustering (vedi Figura 3.5) utilizza una matrice M di rango k, con k uguale al numero totale di classi ODL$_{I^3}$ che devono essere analizzate. In ogni casella $M[h, k]$ della matrice è rappresentato il coefficiente globale $GA()$ riferito alle classi c_h e c_k. La procedura di clustering è iterativa e comincia allocando per ogni classe 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

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 3.5: Procedura di clustering

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 in Figura 3.6 (in tale albero la classe *tax_position_xml.Student* è denominata *Student0* per distinguerla dalla classe *Computer_Science.Student*). In questo albero, le foglie rappresentano tutte le classi locali rappresentate: foglie contigue sono caratterizzate da alta affinità, foglie tra loro molto lontane rappresenteranno invece concetti differenti.
Ogni nodo rappresenta un livello di clusterizzazione, ed ha associato il coefficiente di affinità tra i due sottoalberi (cluster) che unisce. 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 tra le 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 dell'esempio di riferimento sono state raggruppate iterativamente (attraverso la procedura esposta sopra) in cinque cluster finali. Ad ogni cluster così determinato viene associata una Global Class.
La figura 3.7 mostra il pannello di SI_Designer che visualizza i cluster determinati. Come si può osservare, sono messe a disposizione diverse funzionalità:

- *Rename Class*: permette di associare al cluster un nome significativo (che

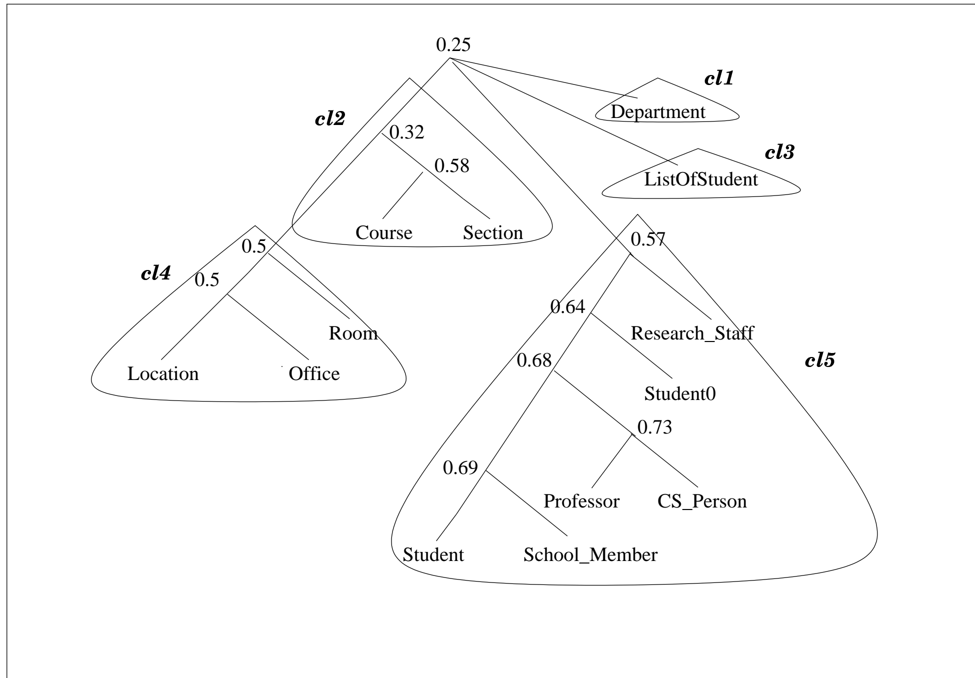

Figura 3.6: Albero di affinità

sarà poi il nome della classe globale ad esso associata);

- *Delete Mapping*: permette di cancellare tutti i cluster correnti;
- *Delete Class*: permette di cancellare un cluster;
- *Add new Class*: rende possibile l'inserimento di un nuovo cluster;
- *UnMap Interf*: effettua la rimozione di una classe selezionata da un determinato cluster;
- *Map Interface*: permette di inserire una classe selezionata all'interno di un cluster.

Osservazione: prima di avviare il procedimento di calcolo dei cluster è necessario inserire il peso associato ad ogni tipo di relazione e le soglie predefinite; a tale scopo si utilizza il pannello denominato "ARTEMIS Configuration".

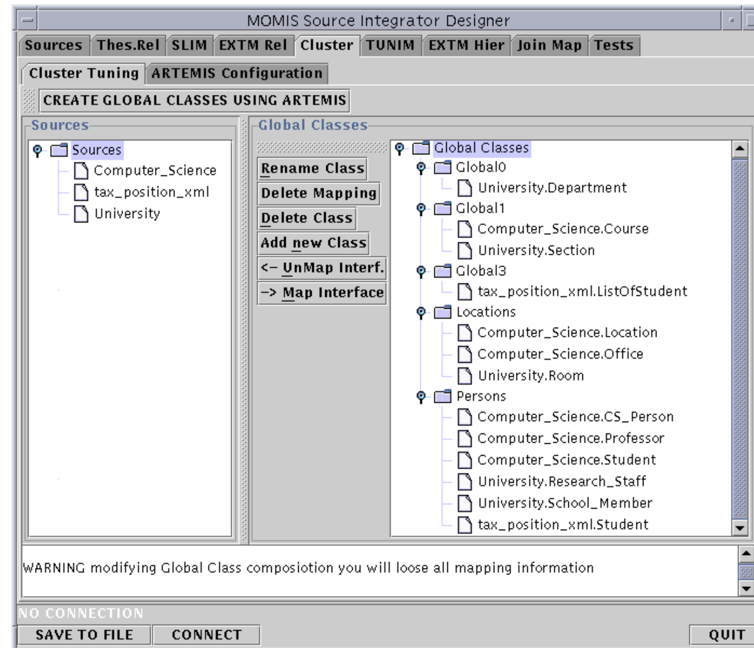

Figura 3.7: Pannello di visualizzazione dei cluster

3.4 Generazione degli attributi globali e delle mapping-table

In questa fase viene eseguita l'integrazione vera e propria degli schemi. Per ogni cluster individuato nella fase precedente viene costruita una classe globale.
Ogni classe è caratterizzata da:

- un **nome** che la identifica univocamente dalle altre;
- un insieme di **attributi globali**;
- una **mapping table** che indica la corrispondenza tra gli attributi globali e le informazioni contenute nelle sorgenti.

La costruzione delle classi avviene in due momenti: in un primo tempo il sistema costruisce, in modo semi-automatico, una prima versione di classi globali, con nomi, attributi globali e mapping table; successivamente il progettista interviene per apportare le modifiche che ritiene necessarie.

Dato un cluster, inizialmente viene creato un insieme d'attributi globali eseguendo l'unione di tutti gli attributi delle classi locali appartenenti al cluster: chiamiamo questo insieme iniziale *insieme-unione*. Indicando con Cl_i il cluster in

esame, con c_j una classe locale appartenente ad esso e con $A(Cl_i)$ gli attributi del cluster Cl_i, l'insieme-unione sarà dato da:

$$A(Cl_i) = \bigcup^{j} A(c_j), \forall c_j \in Cl_i$$

Esempio 5 Riferendoci al cluster Cl_5 di Figura 3.6, l'unione degli attributi delle classi locali che raggruppa fornisce il seguente insieme-unione:

$A(Cl_1)$ = { `first_name, last_name, e_mail,`
`dept_code, section_code, rank, name, takes,`
`student_code, faculty_name, tax_fee, year,`
`belongs_to, faculty` }

Si noti come vi sia una certa ridondanza tra gli attributi: `faculty` e `faculty_name` in realtà hanno lo stesso significato di "nome della facoltà", così come gli attributi `first_name`, `last_name` e `name` rappresentano nomi di persona.

Alla creazione dell'insieme-unione seguono due fasi:

- *Fusione degli attributi simili*,
- *Creazione della mapping table*.

Nella fase di fusione degli attributi "simili", SI_Designer tenta di eliminare queste ridondanze considerando le relazioni terminologiche del *Common Thesaurus*. Il procedimento di fusione è sempre automatico per gli attributi legati da relazioni validate, mentre lo è solo in certi casi se gli attributi sono legati da relazioni non validate. In particolare SI_Designer opera in questo modo:

- **Attributi in relazioni validate.** Per questi attributi la fusione è sempre automatica:
  - Agli attributi legati da relazioni `SYN` SI_Designer farà corrispondere un unico attributo globale: il dominio è lo stesso ed il nome può essere scelto dal progettista tra le proposte di SI_Designer, oppure inserito esplicitamente. Per esempio: agli attributi `section_name` e `course_name`, legati dalla relazione `section_name SYN course_name`, viene associato l'attributo globale `section_name`.
  - Gli attributi legati da relazioni `BT` vengono trattati da SI_Designer sostituendoli con un attributo globale che ha lo stesso nome e lo stesso dominio dell'attributo generalizzazione. Per esempio: gli attributi `name`, `first_name` e `last_name` sono legati dalle relazioni di specializzazione `name BT first_name` e `name BT last_name` per cui i tre attributi saranno rappresentati dall'attributo globale `name`.

- **Attributi in relazioni non validate.** A questa categoria appartengono gli attributi delle relazioni del *Common Thesaurus* che non hanno superato la validazione: SI_Designer è in grado di individuare un attributo globale in modo automatico solo per un numero limitato di casi, lasciando al progettista il compito di aggiungere altri attributi globali per completare l'integrazione. In particolare, l'individuazione automatica di un attributo globale, in presenza di relazioni non validate, è possibile se gli attributi nelle relazioni soddisfano i seguenti requisiti:

  1. sono legati da relazioni `SYN` o `BT`;
  2. le classi in relazione appartengono ad uno stesso cluster;
  3. rappresentano gerarchie di aggregazione (sono attributi complessi o foreign key);

  Come esempio, si consideri la relazione non validata `dept_code BT belongs_to`. Gli attributi `dept_code` e `belongs_to` soddisfano tutte le condizioni elencate prima, infatti:

  1. i due attributi sono legati da una relazione `BT`;
  2. le classi mappate dai due attributi, `Professor` e `Research_Staff`, sono nello stesso cluster;
  3. `dept_code` è foreign key per `Research_Staff` e `belongs_to` è un attributo complesso di tipo `Office`;

  Dunque è possibile aggiungere un attributo globale `works` che corrisponda ai due attributi locali `dept_code` e `belongs_to`.

L'insieme di attributi globali così individuato può essere ulteriormente ampliato dal progettista per rappresentare pienamente tutte le informazioni delle sorgenti locali: questo accade soprattutto quando alcune informazioni sono contenute nelle sorgenti sotto forma di metadato.

Contemporaneamente alla creazione degli attributi globali, SI_Designer costruisce una *mapping-table*. Essa é una tabella $MT[CL][AG]$ dove CL é l'insieme delle classi locali che appartengono al cluster cui la mapping-table si riferisce e AG é l'insieme degli attributi globali creato da SI_Designer. Indicando con C il nome di una classe locale, con A il nome di un attributo globale e con AL il nome di un attributo locale, ogni elemento $MT[C][A]$ della tabella può assumere i seguenti valori:

- AL, con $AL \in C$.
  Questo valore viene inserito quando:

  - l'attributo globale A deve rappresentare l'informazione contenuta nel solo attributo locale AL. Per esempio, l'attributo globale `name` corrisponde all'omonimo attributo `name` della classe locale `Research_Staff`;
  - ci sono relazioni di specializzazione che legano tra loro attributi appartenenti a classi diverse. Per esempio, l'attributo globale `faculty` della classe globale `Person` corrisponde all'omonimo attributo `faculty` della classe locale `University.School_Member` e all'attributo `faculty_name` della classe locale `tax_position_xml.Student`;

- AL_1 ***and*** AL_2 ***and ... and*** AL_n, con $AL_i \in C, i = 1, ..., n$.
  Usato quando il valore dell'attributo A é il concatenamento dei valori di piú attributi appartenenti alla medesima classe locale C. Per esempio, l'attributo globale `name` della classe globale `Person` corrisponde al concatenamento degli attributi `first_name` e `last_name` della classe locale `CS_Person`;

- ***case of*** AL $cost_1$**:** AL_1 $cost_2$**:** AL_2 **...** $cost_n$**:** AL_n
  dove $AL, AL_i \in C, i = 1, ..., n$ e $cost_i, i = 1, ..., n$ sono delle costanti.
  Questa situazione avviene quando l'attributo globale A può assumere il valore di uno tra un insieme di attributi locali $\{AL_i\}$ appartenenti alla medesima classe e la scelta avviene attraverso un terzo attributo locale AL, appartenente sempre alla stessa classe locale, che funge da selettore.

- $costante$.
  Si contempla il caso in cui l'attributo globale A non corrisponde ad alcun attributo della classe locale C. Il valore assunto da A viene attribuito dal progettista in base al significato dato all'attributo globale. Per esempio, l'attributo globale `rank` della classe globale `Person` assume il valore costante `'Professor'` se occorre accedere alla classe locale `Research_Staff` e il valore costante `'Student'` se occorre accedere alla classe locale `University_Student`.

- $null$.
  Questo é il caso in cui l'attributo globale A, durante un accesso alla classe locale C, non assume alcun valore. Per esempio, l'attributo globale `faculty` non assume alcun valore nella classe locale `Research_Staff`: ciò viene indicato nella mapping-table con la costante $null$.

SI_Designer crea una *mapping-table* per ogni classe globale e mette a disposizione del progettista un' interfaccia che permette sia di avere una visione completa

Person	name	rank	works	faculty	email	...
Research_Staff	first_name *and* last_name	'Professor'	dept_code	null	email	...
School_Member	first_name *and* last_name	'Student'	null	faculty	null	...
CS_Person	name	null	null	'Computer_Science'	null	...
Professor	name	rank	belongs_to	'Computer_Science'	null	...
Student	name	rank	null	'Computer_Science'	null	...
Student0	name	'Student'	null	faculty_name	null	...

Figura 3.8: Mapping-table di `Person`

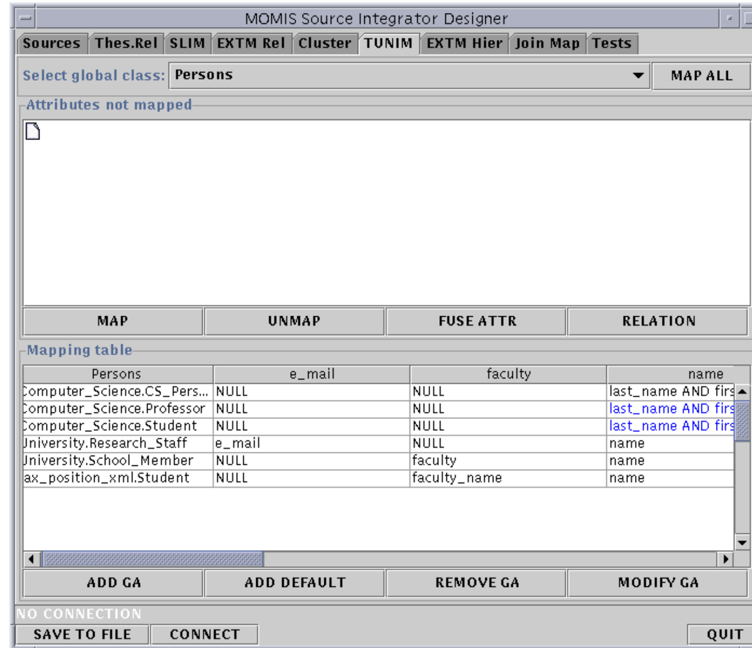

Figura 3.9: Pannello di visualizzazione di una mapping-table

di tutte le classi globali (nomi ed attributi), che l'inserimento dei nomi delle classi globali e la modifica delle *mapping table*. In Figura 3.8 sono riportate le colonne più significative della *mapping-table* **Person** mentre in Figura 3.9 le stesse sono mostrate all'interno del pannello "TUNIM" di SI_Designer. Le funzionalità messe a disposizione dal suddetto pannello sono diverse, in particolare:

- *MODIFY GA*: permette di modificare il nome di una classe globale;
- *REMOVE GA*: permette di rimuovere una classe globale;
- *ADD GA*: permette di inserire manualmente una nuova classe globale;

- *MAP ALL*: premendo questo pulsante si avvia la procedura di creazione automatica della *mapping-table* iniziale (quella ottenuta dall'unione di tutti gli attributi locali);
- *MAP*: avvia una fase di mapping di un attributo locale non ancora mappato;
- *UNMAP*: selezionando un attributo contenuto in una *mapping-table*, ne permette la rimozione dalla stessa;
- *FUSE ATTR*: permette di eseguire la fusione degli attributi fondibili. Quello che il modulo fa in questo caso è considerare tutti gli attributi globali che sono stati mappati su un solo attributo locale, tentando di individuare delle possibili fusioni in base all'analisi delle relazioni del Thesaurus. In questo modo, se alcune fusioni vengono eseguite, si riduce il numero degli attributi globali nella *mapping-table*;
- RELATION: selezionando un attributo non ancora mappato (quindi posto nel pannello in alto denominato "Attribute not mapped"), permette di visualizzare tutte le relazioni validate che lo coinvolgono;
- ADD DEFAULT: permette di inserire manualmente un valore di default per un attributo selezionato.

Capitolo 4

Il modulo SIM

Il modulo **SIM** (**S**ources **I**ntegrator **M**odule) rappresenta uno dei primi componenti realizzati nell'ambito del progetto MOMIS. La prima versione di **SIM**, sviluppata dall'Ing. Simone Montanari [3], risale al 1996.

Scopo di questa tesi è la ridefinizione del suddetto modulo alla luce dei diversi cambiamenti che hanno caratterizzato lo sviluppo del progetto da allora fino ad oggi. Nel seguente capitolo verranno mostrati gli obiettivi di **SIM** e le principali differenze fra l'attuale modulo ed il primo prototipo; si rimandano al capitolo successivo i dettagli relativi alla realizzazione del software.

4.1 Obiettivi del Modulo

Il modulo **SIM** è un componente del Global Schema Builder, ovvero di quella parte del mediatore di MOMIS che si occupa dell'integrazione degli schemi locali, giungendo alla generazione di uno schema globale da presentare all'utente. Le funzionalità che **SIM** mette a disposizione possono essere riassunte in tre punti:

1. *estrazione automatica delle relazioni dagli schemi* ODL_{I^3}: vengono estratte, in modo automatico, tutte le relazioni intraschema che coinvolgono sorgenti ad oggetti, relazionali e semistrutturate;

2. *validazione delle relazioni*: in seguito all'intervento del progettista viene eseguita una fase di validazione delle relazioni fra attributi presenti nel Common Thesaurus, al fine di verificare la compatibilità dei domini coinvolti;

3. *inferenza di nuove relazioni*: partendo da tutte le relazioni già presenti nel Common Thesaurus (e validate), si deducono automaticamente nuove relazioni semantiche utilizzando le tecniche di inferenza messe a disposizione da ODB-Tools.

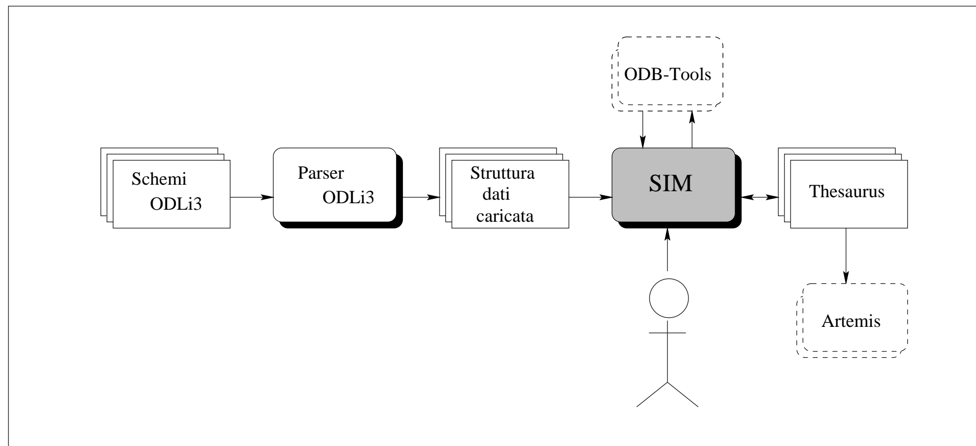

Figura 4.1: Il modulo **SIM**

Le funzioni di validazione ed inferenza sono necessarie per ottenere il Thesaurus definitivo che rappresenterà la base per la successiva fase di determinazione dei cluster realizzata dal modulo Artemis.
Al fine di realizzare tutti gli obiettivi sopra elencati, il modulo **SIM** interagisce più volte con ODB-Tools; tali interazioni, come sarà in seguito descritto più approfonditamente, sono indispensabili tanto nella fase iniziale di estrazione delle relazioni dalle sorgenti ad oggetti, quanto nelle fasi successive di validazione e deduzione automatica di nuove relazioni.

Prima di entrare nel dettaglio delle tre funzioni, ritengo sia opportuno anticipare un'importante differenza fra l'attuale modulo e quello precedentemente esistente. Il modulo **SIM** che questa tesi si è proposta di realizzare rappresenta un componente di SI_Designer ed, in quanto tale, non interagisce direttamente con i file contenenti le descrizioni ODL$_{I^3}$ degli schemi. SI_Designer utilizza, infatti, un parser Java che, presi in ingresso i suddetti schemi, carica in una struttura dati complessa (ovvero in un insieme di istanze di classi Java) tutte le informazioni che da essi si possono ricavare. **SIM** acquisisce la conoscenza necessaria alla propria elaborazione utilizzando quest'ultima struttura (vedi Figura 4.1). Al contrario, il precedente modulo acquisiva direttamente gli schemi delle sorgenti espressi nel linguaggio ODL$_{I^3}$: questo rendeva necessario, innanzitutto, implementare un parser interno e la definizione di una serie di controlli di coerenza (per rilevare, ad esempio, la presenza di foreign key prive degli attributi corrispondenti nella classe referenziata). Attualmente questi controlli sono demandati al parser Java utilizzato da SI_Designer il quale controlla che il testo di input soddisfi determinate regole sintattiche e grammaticali, eseguendo anche alcuni controlli semantici sulla consistenza delle informazioni acquisite. Per la descrizione del parser ODL$_{I^3}$

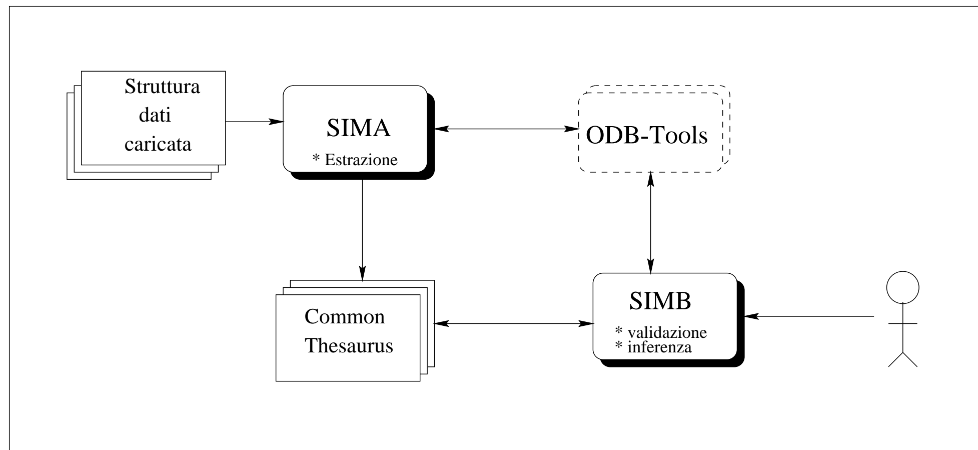

Figura 4.2: Architettura del modulo **SIM**

si rimanda al capitolo successivo (Sezione 5.1).

Dal punto di vista implementativo si è mantenuta la struttura già presente nel primo prototipo, ovvero una suddivisione del modulo **SIM** in due sottomoduli (vedi Figura 4.2):

- **SIMA**: si occupa della prima fase di estrazione automatica delle relazioni e del successivo popolamento del Thesaurus;
- **SIMB**: esegue la fase di validazione delle relazioni fra attributi presenti nel Thesaurus e la fase di inferenza di nuove relazioni.

4.2 Il modulo SIMA

Compito del modulo **SIMA** è di effettuare un'analisi preliminare degli schemi delle sorgenti al fine di esplicitare un'insieme di relazioni intra-schema in realtà già implicitamente presenti negli schemi stessi. Sulla base della natura delle sorgenti analizzate si distinguono diverse procedure:

- *analisi degli schemi ad oggetti*: sono utilizzate a tale proposito le funzionalità offerte da OLCD-Designer (componente interno ad ODB-Tools). Ad esso è demandata l'analisi delle gerarchie che legano le classi, siano esse esplicitamente indicate negli schemi (ereditarietà dirette, attributi con domini complessi), o dedotte attraverso l'algoritmo di sussunzione ed espansione semantica.

Esempio 6 Si consideri ora la classe `Professor`, proveniente dalla sorgente ad oggetti `Computer_Science`:

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

Il sistema, utilizzando al suo interno OLCD-Designer, dedurrà (ed inserirà nel Thesaurus) le seguenti relazioni (che quindi a questo livello non sono semplicemente relazioni basate sulla sola analisi dei termini, bensì dedotte dalle informazioni semantiche presenti negli schemi):

⟨`Computer_Science.Professor` NT_{ext} `Computer_Science.CS_Person`⟩
⟨`Computer_Science.Professor` RT `Computer_Science.Office`⟩

derivate rispettivamente dall' esplicita dichiarazione di ereditarietà della classe `Professor` rispetto alla classe `CS_Person` e dal dominio dell'attributo complesso `belongs_to`.

- *analisi degli schemi relazionali*: l'estrazione delle relazioni intraschema che riguardano le sorgenti relazionali viene effettuata mediante l'analisi delle foreign key. Ogni volta in cui è presente una foreign key è immediato dedurre una relazione RT fra la classe che la definisce e quella che viene da essa referenziata.

Esempio 7 Si consideri ad esempio la classe `Section`, così descritta in ODL$_{I^3}$:

```
interface Section
( source relational University
  extent Sections
  key section_code
  foreign_key room_code
  references Room (room_code))
{ attribute string section_name;
  attribute integer section_code;
  attribute integer lenght;
  attribute integer room_code; };
```

Dall'esame di questa classe (si ricorda che usiamo genericamente il termine classe anche per tabelle relazionali, al fine di uniformare la visione di tutte le entità integrate) il sistema riconosce che vi è una relazione che lega `Section` a `Room` attraverso la definizione della foreign key sull'attributo `room_code`, generando in questo modo la relazione:

⟨`University.Section` RT `University.Room`⟩

Affiancando all'analisi delle foreign key uno studio sulle chiavi (primarie e candidate) è possibile, talvolta, evidenziare ulteriori relazioni (più forti della semplice RT). In particolare:

- **relazione** NT_{ext}: si rileva quando la foreign key è anche chiave (primaria o candidata) della classe che la definisce. In questo caso è presente una relazione NT tra la classe che contiene la chiave esterna e quella riferita (in questo ordine);
- **relazione** SYN_{ext}: si rileva quando fra due classi è presente una relazione reciproca di NT (o equivalentemente di BT). In questo caso le due classi sono sinonime.
- **relazione** RT_{partof}: si rileva ogni volta in cui la foreign key è parte di una chiave (primaria o candidata) della classe che la definisce. In questo caso è presente una relazione RT_{partof} fra la classe definisce la foreign key e quella riferita (in questo ordine).
- **relazioni RT derivate per estensione di relazioni già presenti**: talvolta è possibile dedurre nuove relazioni RT estendendo relazioni RT o RT_{partof} già esistenti nel Thesaurus. Più esplicitamente si può affermare che, considerata una determinata classe e tutte le relazioni RT (o RT_{partof}) che la interessano al primo membro, si possono stabilire nuove relazioni RT fra ogni coppia di classi coinvolte al secondo membro nelle suddette relazioni.
- **relazione SYN fra attributi**: quando la foreign key è formata da un attributo il cui nome non corrisponde a quello dello stesso attributo nella classe riferita, si può segnalare tra i due una relazione di sinonimia.

• *analisi degli schemi semistrutturati*: su tali schemi viene eseguita un'analisi delle eventuali foreign key in modo analogo a quanto avviene per gli schemi relazionali. Si utilizzano poi le funzionalità di ODB-Tools per l'analisi delle gerarchie di aggregazione che legano le classi.

A questo punto è bene fare alcune precisazioni in merito a quanto appena descritto.
L'idea di aggiungere un nuovo tipo di relazione rispetto a quelle tradizionali (ricordo che a tutt'oggi sono contemplate nel Thesaurus le relazioni di NT, BT, RT, SYN e non la RT_{partof}) nasce dalla necessità di esprimere un legame fra classi che sia più forte di una semplice relazione di aggregazione (ovvero di una RT). Attualmente le relazioni RT_{partof} vengono segnalate come tali, ma inserite nel Common Thesaurus come semplici RT (in futuro si potrebbe pensare di inserire anche questa nuova tipologia nella famiglia di relazioni ammesse nel Thesaurus).
Un'altra nota in merito al popolamento del Thesaurus riguarda le relazioni RT derivanti da un'estensione di relazioni RT o RT_{partof} già presenti: tali relazioni vengono estratte, segnalate, ma non inserite nel Common Thesaurus. Questa scelta è dettata dalla volontà di mantenere nel Common Thesaurus solo le relazioni minimali (essenziali) per evitare di introdurre inutili ridondanze.

4.3 Il modulo SIMB

In figura 4.3 è mostrato l'automa a stati finiti del modulo **SIMB**. Il punto di partenza è l'insieme delle relazioni sino a quel punto memorizzate nel Thesaurus (comprendente dunque sia quelle estratte dal modulo **SIMA**, sia quelle aggiunte dal progettista). Il suo compito è quello di fornire nuovi schemi che esprimano tutte le informazioni già memorizzate nel Common Thesaurus; tali schemi saranno analizzati attraverso OLCD-Designer in modo da inferire nuove relazioni terminologiche.
Per arrivare alla definizione del Thesaurus definitivo sono realizzati dal modulo diversi passaggi che verranno approfonditi nei prossimi paragrafi.

Le relazioni da esaminare possono essere di tre tipi:

1. *relazioni tra attributi*: le relazioni che coinvolgono coppie di attributi sono soggette ad una fase di validazione preliminare alla modifica degli schemi.

2. *relazioni tra classi*: le relazioni che coinvolgono coppie di classi saranno analizzate in un secondo momento, successivamente all'analisi delle relazioni tra attributi da validare;

3. *relazioni miste*: nel caso siano state inserite dal progettista relazioni che coinvolgono un attributo e una classe, queste informazioni non sono utilizzate dal modulo **SIMB**.

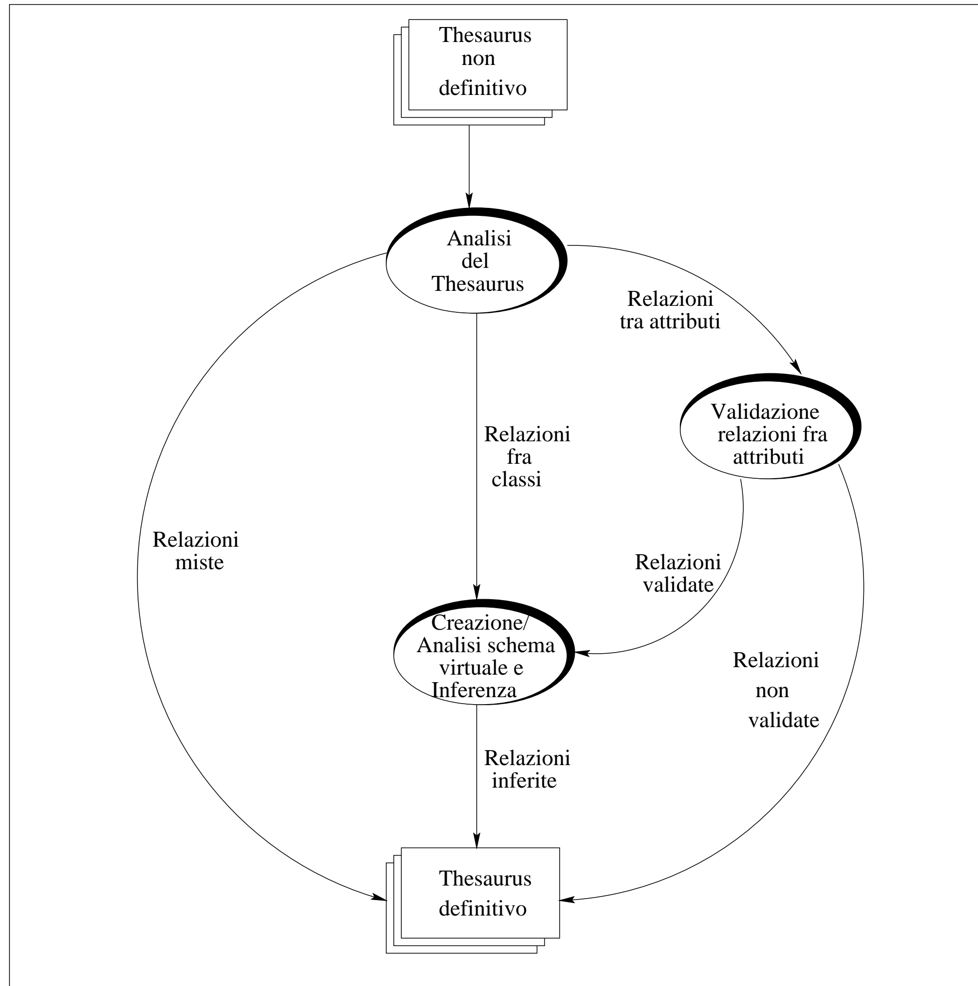

Figura 4.3: Automa a stati finiti del modulo **SIMB**

4.3.1 Validazione delle relazioni tra attributi

Un importante compito del modulo **SIMB** consiste nella validazione delle relazioni fra attributi inserite nel Common Thesaurus dal progettista. Si fa ora riferimento, quindi, a tutte le relazioni inserite dal modulo SLIM interagendo col progettista (individuate tramite WordNet), ed a tutte le relazioni aggiunte direttamente dal progettista (non individuabili attraverso WordNet poichè non affini in termini linguistici). La necessità di introdurre questo procedimento è legata alla natura delle suddette relazioni: esse hanno carattere esclusivamente *terminologico*, proprietà che prescinde da qualunque considerazione sulle estensioni (si ricorda che ogni relazione lessicale si basa solamente sull'analisi dei termini che

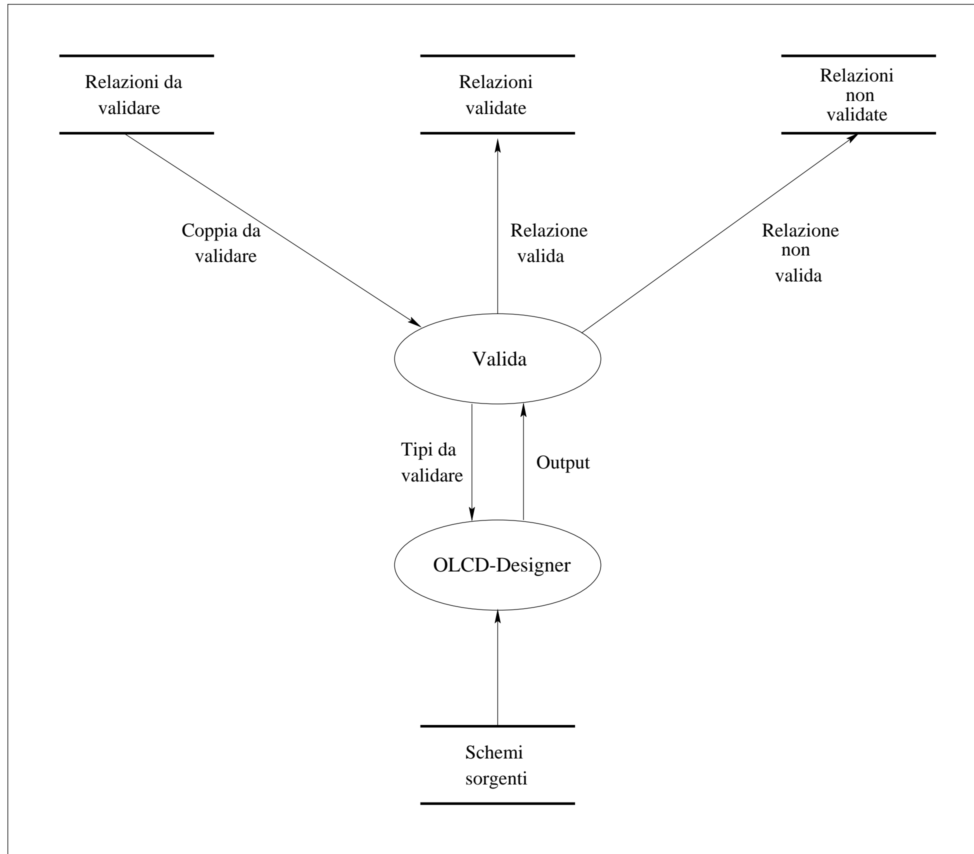

Figura 4.4: DFD della procedura di Validazione delle relazioni fra attributi

identificano un concetto).
Lo scopo della fase di validazione è verificare che i domini degli attributi coinvolti nelle relazioni in esame siano compatibili col tipo di relazione definita fra gli stessi. Per realizzare tale controllo il modulo **SIMB** interagisce con ODB-Tools, al cui interno sono presenti algoritmi in grado di valutare la compatibilità fra i diversi domini. Le modalità con cui si sviluppa tale iterazione, ovvero come **SIMB** effettua la richiesta e successivamente interpreta il risultato ottenuto dal componente esterno, saranno descritte nel capitolo successivo. Qui di seguito si riportano le regole su cui si basa il processo di validazione.

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. Siano $a_t = \langle n_t, d_t \rangle$ e $a_q = \langle n_q, d_q \rangle$ due attributi coinvolti in una relazione da validare.

- $\langle n_t$ 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 presente in OLCD-Designer: ad esempio, d_t contiene d_q se il tipo del secondo è sussunto dal tipo del primo.
Operare con sorgenti semistrutturate rende possibile definire per uno stesso attributo più domini, ciascuno appartenente ad una diversa struttura associata alla classe che lo contiene. Validare una relazione fra attributi può significare, quindi, validare una relazione fra una semplice coppia di domini, oppure tra due insiemi di domini. In quest'ultimo caso, si ritiene la relazione valida solo se esiste *almeno* una coppia di domini (ciascuno appartenente ad un insieme diverso) compatibile col tipo di relazione da validare.
In figura 4.4 è riportato il Data Flow Diagram della procedura di validazione come è realizzata del modulo **SIMB**.
Il risultato del processo di validazione viene reso noto al progettista, il quale può decidere di intervenire sulle relazioni non validate automaticamente, forzandone lui stesso la validazione.
È importante sottolineare il ruolo fondamentale che la validazione delle relazioni fra attributi riveste all'interno del processo di integrazione delle sorgenti. Numerose fasi successive sono, infatti, influenzate dal risultato di tale procedimento e fra esse ricordo:

- inferenza di nuove relazioni: questa fase prevede la definizione di uno schema virtuale ottenuto, a partire dagli schemi originali, applicando regole connesse alle relazioni presenti nel Thesaurus (e che verranno descritte nel prossimo paragrafo). Di tutte le relazioni fra attributi in esso contenute, si terrà conto solo ed esclusivamente di quelle validate.

- calcolo dei coefficienti di affinità: questa fase si prefigge l'obiettivo di quantificare il grado di affinità mutuo fra le classi locali al fine di raggrupparle in cluster. A tale scopo vengono calcolati dei coefficienti, tra cui lo *Structural Affinity Coefficient* dedotto fra due classi in base alle relazioni esistenti

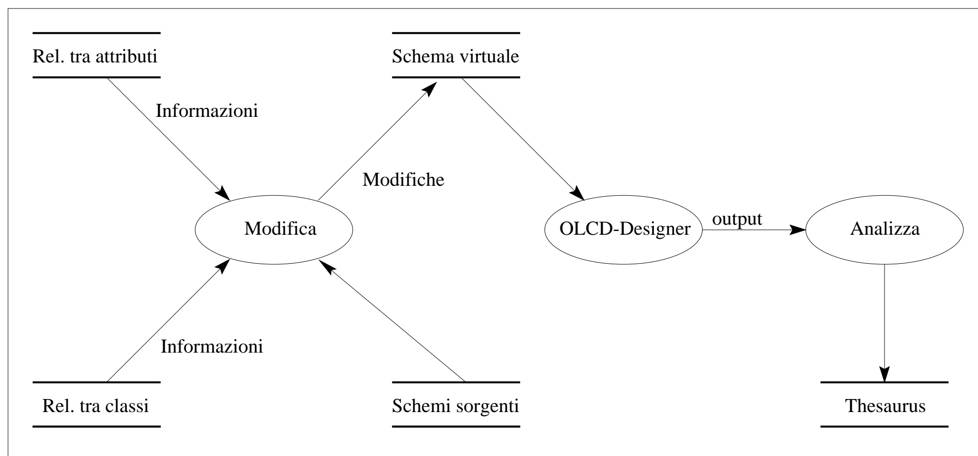

Figura 4.5: DFD della procedura di inferenza

fra i loro attributi. Come mostrato nel capitolo precedente (Sezione 3.2.1), nel calcolo di tale coefficiente si considera un fattore (F_c) che rappresenta il rapporto tra il numero di relazioni validabili ed il numero di relazioni validate.

- costruzione delle classi globali: questa fase comprende un procedimento di fusione degli attributi ”simili” che si articola diversamente a seconda che essi siano implicati in relazioni validate (la fusione sarà in questo caso sempre automatica) o in relazioni non validate (in tale circostanza la fusione è automatica solo in certi casi).

4.3.2 Generazione del nuovo schema ed inferenza

Le relazioni tra attributi che hanno superato il processo di validazione, come pure le relazioni tra classi (ivi comprese sia quelle dedotte dal modulo **SIMA**, sia quelle aggiunte dal progettista) costituiscono la base per la creazione di uno schema virtuale che racchiuda tutta la conoscenza intensionale espressa nel Common Thesaurus fino a questo momento (vedi Figura 4.5).
Lo schema così creato, opportunamente tradotto in OLCD, sarà inviato ad ODB-Tools, il quale, utilizzando tecniche di inferenza, è in grado di ottenere nuove relazioni semantiche; l’uso del componente esterno permette inoltre di eseguire un controllo sulla consistenza del nuovo schema, realizzando quindi una sorta di validazione delle relazioni tra classi che hanno contribuito a generare lo schema stesso.
La necessità di creare uno schema virtuale, lasciando inalterati quelli originali, na-

sce dal fatto che le relazioni tra classi inserite dal progettista nelle fasi precedenti potrebbero essere in conflitto con le descrizioni strutturali delle classi correlate.

Rispetto agli schemi originali, nel nuovo schema:

- classi: vengono modificate in modo da conformarsi pienamente a tutte le relazioni presenti nel Common Thesaurus; sono quindi aggiunte negli schemi informazioni di ereditarietà (per rappresentare BT o NT non presenti originariamente), di aggregazione (per rappresentare RT fra classi) e di sinonimia (per esprimere relazioni di SYN).

- attributi: tutti gli attributi coinvolti in relazioni validate sono riorganizzati in gerarchie. Per ogni attributo sono individuati gli attributi equivalenti (identificati da nomi definiti sinonimi), quelli più specializzati e quelli più generali. Ogni relazione di sinonimia pone gli attributi coinvolti allo stesso livello, mentre le relazioni BT ed NT pongono a livello superiore il termine più generale della gerarchia di appartenenza. Alla fine di tale riorganizzazione ogni attributo è sostituito col termine più generale della gerarchia di appartenenza: in questo modo si rendono il più possibile confrontabili le intensioni dello schema prodotto.

Da notare come, oltre a ricomporsi i singoli schemi locali, le relazioni lessicali aggiungano nuovi collegamenti *interschema* in modo tale che alla fine risulti uno schema unico. Sarà proprio questo schema ad essere tradotto nel formalismo OLCD ed inviato al modulo OLCD-Designer, il quale verificherà la consistenza del nuovo schema ed inferirà nuove relazioni. Le relazioni inferite possono derivare sia da informazioni precedentemente inserite attraverso la modifica degli schemi originali, sia da legami dedotti tra le strutture stesse delle classi applicando l'algoritmo di sussunzione: maggiore sarà l'insieme delle relazioni inferite, maggiore il legame che si creerà fra le classi in sede di calcolo dei coefficienti di affinità, e minore il compito di inserimento manuale di nuove relazioni da parte del progettista.

Inconsistenza dello schema creato Come già precedentemente accennato, modificare gli schemi sulla base delle relazioni validate presenti nel Common Thesaurus può originare delle inconsistenze: è il caso in cui esiste almeno una relazione in conflitto con la struttura degli schemi originali. Nell'esempio di riferimento universitario (che abbiamo fin d'ora preso a modello) questa situazione non si verifica; introduciamo quindi un altro contesto (quello dei ristoranti) che mostri un esempio di inconsistenza rilevata da ODB-Tools in relazione agli schemi introdotti nella fase di inferenza.

La descrizione ODL_{I^3} degli schemi a cui ora faremo riferimento è quella mostrata

```
prim  Owner = Person & ^[job:string,name:string,address:Address,
                        rtArg:Fast_Food,rtArg0:Address];

prim Fast_Food = Restaurant  & ^[Owner1:Owner,name:string, category:string,
                                address:Address,phone:integer,
                                speciality:<string>,midprice:integer,
                                nearby:Fast_Food,];

prim  Address = ^[dummy:string,zipcode:string,street:string,city:string,
                 rtArg3:Owner,rtArg4:Fast_Food];

prim  Restaurant = ^[r_code:string,street:string,pers_id:integer,
                    name:string,tourist_men:integer,zip_code:string,
                    special_dis:string,trARG10:Person, category:integer ];

prim  Person = ^[qualificati:integer,pers_id:integer,last_name:string,
                first_name:string,rtArg5:Bistro,rtArg6:Restaurant];
...
```

Figura 4.6: Traduzione OLCD degli schemi modificati in riferimento all'esempio dei ristoranti

nel Capitolo 2 in Figura 2.1.
Il nuovo schema creato dal modulo **SIMB** (preambolo alla fase di inferenza) è mostrato in Figura 4.6. L'elaborazione operata dal componente esterno fornisce in output una stringa XML (strutturalmente analoga a quella mostrata in Figura 5.5) nella quale sono significative le seguenti righe:

```
...
<CLASS name = "Owner">
            <ATTRIBUTE type = "bottom" name = ""/>
</CLASS>

<CLASS name = "Fast_Food">
            <ATTRIBUTE type = "bottom" name = ""/>
</CLASS>
<CLASS name = "Address">
            <ATTRIBUTE type = "bottom" name = ""/>
</CLASS>
...
```

Come si può vedere, le classi *Owner, Fast_Food* ed *Address* sono dichiarate da ODB-Tools inconsistenti: l'unico attributo di cui sono popolate è, infatti, di tipo "bottom". La causa di tale inconsistenza è chiara se si osservano con attenzione gli schemi di Figura 4.6: la classe *Fast_Food* eredita dalla classe *Restaurant* (tale legame deriva dalla relazione ⟨`ED.Fast_Food` `NT` `FD.Restaurant`⟩ inserita

manualmente dal progettista), ma entrambe hanno al loro interno l'attributo ' category', definito nella prima di tipo *string* e nella seconda di tipo *integer*.
Essendo inconsistente la classe *Fast_Food*, risultano esserlo anche le classi *Owner* ed *Address* in quanto hanno un attributo di tipo *Fast_Food*.
Per eliminare questa inconsistenza si può pensare di rendere omogenee (in modo automatico) le descrizioni delle classi *Fast_Food* e *Restaurant* all'interno dello schema virtuale (ad esempio ponendo in *union* i due domini coinvolti nell'incompatibilità): in questo modo il componente esterno non segnalerebbe alcuna inconsistenza.

Vediamo ora una situazione di inconsistenza più grave, in quanto derivata da una relazione estensionale. Supponiamo che il progettista abbia inserito la relazione:
$\langle$`ED.Fast_Food` SYN_{Ext} `FD.Restaurant`$\rangle$

che implica la definizione automatica della relazione intensionale:

$\langle$`ED.Fast_Food` SYN `FD.Restaurant`$\rangle$

da inserire nel Common Thesaurus.

La validazione di una relazione estensionale tra due classi C_1 e C_2 viene fatta valutando la consistenza della classe C_1. Per esempio, la relazione estensionale $\langle$`FD.Restaurant BT`$_{ext}$ `FD.Bistro`$\rangle$ viene espressa nella classe `FD.Bistro` in questo modo:

$$\sigma_P(\texttt{FD.Bistro}) = \texttt{FD.Restaurant} \sqcap \triangle\,[\,\texttt{r_code:String, type:String, pers_id:Integer}]$$

Dal momento che la descrizione della classe `FD.Bistro` è consistente, la relazione tra `FD.Bistro` e `FD.Restaurant` è valida. D'altra parte, la relazione estensionale $\langle$`ES.Fast-Food SYN`$_{ext}$ `FD.Restaurant`$\rangle$ viene rifiutata, perché la descrizione della classe:

$$\sigma_P(\texttt{ES.Fast-Food}) = \texttt{FD.Restaurant} \sqcap \ldots$$

è inconsistente (come prima l'attributo `category` è definito in entrambe le classi, ma con domini disgiunti).
Trattandosi di relazioni estensionali (che quindi vincolano le istanze delle classi coinvolte) questa inconsistenza va segnalata in modo *forte* al progettista; continuare il processo di integrazione a fronte di tale incongruenza potrebbe generare uno schema globale errato.

4.4 Confronto fra il primo prototipo e l'attuale

Le differenze principali fra l'attuale modulo e quello precedente possono essere così riassunte:

Linguaggio Il modulo **SIM** sviluppato nel 1996 è scritto in ANSI C mentre l'attuale è stato realizzato in Java. I motivi che hanno portato a scegliere questo linguaggio implementativo sono diversi:

- *Programmazione ad oggetti.* Essendo gli schemi locali da integrare descritti nel linguaggio ad oggetti ODL_{I^3}, è più semplice trattare le relative informazioni con un linguaggio ad oggetti come Java piuttosto che con un linguaggio strutturato. Mantenere un linguaggio quale l'ANSI C avrebbe comportato l'onere di operare con strutture dati complesse e pesanti (liste, puntatori, array, ecc).

- *Portabilità di Java.* Esiste l'interprete Java per moltissime piattaforme Hw/Sw. Un modulo realizzato in C necessita, invece, di ricompilazione per essere utilizzato su piattaforme diverse e non è così portabile come un modulo realizzato in Java.

- *Minor lavoro di programmazione.* L'interprete Java ha la caratteristica di deallocare automaticamente la memoria per le istanze non utilizzate (tale meccanismo è chiamato *garbage collection*): ciò rende più facile la scrittura del codice e permette al programmatore di concentrarsi maggiormente sulla codifica dell'algoritmo da implementare.

- *Larga diffusione di Java per applicazioni che utilizzano Internet.* L'obiettivo di MOMIS è sviluppare un sistema a mediatore che integri diverse sorgenti di informazioni eterogenee e distribuite, sia strutturate che semistrutturate: Internet è ricca di sorgenti di questo tipo e la necessità di accedere in modo integrato a tali informazioni è pressante, vista la continua espansione della rete.

- *Elevata modificabilità.* L'uso di un linguaggio ad oggetti come Java permette di creare facilmente applicazioni modulari, semplificando eventuali modifiche al codice già esistente: in un progetto complesso come il sistema MOMIS (del quale non è ancora terminata la fase di ricerca) ciò è sicuramente un pregio.

- *Conformità.* La scelta di riscrivere il modulo utilizzando il linguaggio Java è motivata anche dall'intenzione di conformarsi agli altri moduli del progetto, implementati tutti in tale linguaggio.

Sorgenti semistrutturate Il primo prototipo **SIM** non contemplava la possibilità di operare con sorgenti semistrutturate. Lo sviluppo che ha caratterizzato il progetto MOMIS in questi anni ha reso invece possibile interagire con tali sorgenti tramite wrapper XML. L'attuale modulo **SIM** opera, quindi, anche su schemi semistrutturati.

Analisi degli schemi relazionali La prima versione di **SIM** si proponeva di eseguire un'analisi degli schemi relazionali che portasse all'individuazione di relazioni RT derivate dalle foreign key. L'attuale modulo si propone, invece, di effettuare un'analisi più approfondita degli schemi al fine di ottenere ulteriori relazioni (quali la sinonimia o l'ereditarietà). Per raggiungere tale obiettivo viene eseguito, come già dettagliatamente descritto nel paragrafo 4.2, uno studio esteso a tutte le chiavi delle classi coinvolte.

Acquisizione degli schemi Come già anticipato all'inizio del capitolo, il primo prototipo **SIM** acquisiva in ingresso gli schemi delle sorgenti in linguaggio ODL_{I^3} ed, utilizzando un parser interno, caricava le informazioni in essi contenute all'interno di strutture dati C piuttosto complesse. Oggi **SIM** fa parte di SI_Designer e non si interfaccia più direttamente agli schemi, ma ad una sorta di contenitore (il GlobalSchemaProxy) che memorizza tutte le informazioni dello schema globale (preservando anche quelle relative agli schemi locali originali). La scelta di utilizzare questo "contenitore" di informazioni, non riproponendo un parser interno al modulo, è dettata principalmente da due motivazioni:

1. Il GlobalSchemaProxy rappresenta una risorsa comune a tutti i moduli del progetto: utilizzarlo aumenta il grado di compattezza ed interazione fra gli stessi;

2. L'esistenza di due diversi parser raddoppierebbe le moli di lavoro nel caso, ad esempio, di modifiche al linguaggio ODL_{I^3}.

Cicli di relazioni Prima di modificare gli schemi per eseguire la fase di inferenza, l'attuale modulo **SIM** effettua un controllo sulle relazioni NT e BT, al fine di individuare la presenza di eventuali cicli. In virtù del fatto che ODB-Tools contiene già al suo interno un algoritmo in grado di individuare cicli di relazioni di ereditarietà, il suddetto procedimento potrebbe sembrare ripetitivo ed inutile. In realtà il controllo realizzato dal modulo **SIM** (ed in particolare contenuto in **SIMB**) si propone di fornire un risultato che, non solo segnali l'esistenza di eventuali cicli (funzionalità già presente in ODB-Tools), ma indichi esplicitamente le relazioni che compongono tale ciclo e le segnali al progettista. Ogni ciclo segnalato esprime una relazione di sinonimia fra gli elementi coinvolti, siano essi classi o

attributi; il progettista potrà, eventualmente, confermare tale sinonimia sostituendo le relazioni ”imputate” con le equivalenti SYN. Fintanto esisteranno dei cicli di relazioni, il modulo segnalerà l’anomalia e non permetterà di eseguire la fase di inferenza.

Controllo sulle relazioni che coinvolgono gli stessi elementi Un altro controllo presente nell’attuale modulo **SIM** (e non previsto nel precedente) riguarda le relazioni che coinvolgono la stessa coppia di classi o attributi. Prima di modificare gli schemi sulla base delle informazioni contenute nel Common Thesaurus (preambolo alla fase di inferenza), è opportuno verificare la coerenza delle stesse; in particolare non devono essere presenti più relazioni che coinvolgono la stessa coppia di elementi. Un controllo di questo tipo viene già eseguito a monte da un apposito filtro (contenuto nella funzione di inserimento delle relazioni nel Thesaurus) che sarà descritto nel capitolo successivo. La necessità di riproporlo a questo punto del processo di integrazione nasce dal fatto che, tra le relazioni che popolano il Thesaurus, possono essere presenti relazioni non inserite da **SIM** o dal progettista, ma direttamente esplicitate nei file contenenti la descrizione ODL$_{I^3}$ degli schemi originali: mentre le prime sono immesse nel Thesaurus usando la funzione sopra citata, le seconde sono inserite dal parser ODL$_{I^3}$ non risultando quindi sottoposte ad alcun filtro che ne verifichi, a priori, la correttezza.
Anche nel caso siano individuate più relazioni che coinvolgono la stessa coppia di elementi viene segnalata l’anomalia e non si procede con la fase di inferenza finchè essa non è stata eliminata.

Capitolo 5

Progetto e realizzazione del software

In questo capitolo verrà descritto dettagliatamente il software realizzato per implementare le funzionalità di **SIM**. Verranno riproposti i passi che nel precedente capitolo sono stati presentati dal punto di vista teorico, illustrandone ora la caratteristiche realizzative.

Il software che sarà presentato nei prossimi paragrafi è stato realizzato in linguaggio Java, versione 1.2.2. Le giustificazioni per l'utilizzo di Java e l'abbandono del linguaggio C (utilizzato nel precedente prototipo e negli ODB-Tools) sono state ampliamente discusse nel capitolo precedente (Sezione 4.4). Ricordo soltanto che, essendo Java un linguaggio completamente orientato agli oggetti, favorisce la modularità e la componibilità del software, caratteristiche indispensabili nell'ambito di un progetto di ricerca a lungo termine come MOMIS.

5.1 Il parser ODL$_{I^3}$

Prima di addentrarci nella descrizione dei passi con cui il modulo **SIM** esplica le sue funzioni, ritengo opportuno approfondire gli aspetti fondamentali relativi parser ODL$_{I^3}$ ed alle classi da esso istanziate.

La fase di *parsing* consiste nell'acquisizione delle informazioni contenute in uno o più file di testo: ad esempio il parsing rappresenta la prima fase del processo di compilazione di un file sorgente; tale file deve essere scritto nel rispetto di un insieme di regole sintattiche e grammaticali note sia a chi lo produce (tipicamente una persona) sia al parser stesso, che si preoccupa di controllare che tali regole siano rispettate, generando, ove necessario, opportuni messaggi d'errore. Il software che realizza il parser deve, per quanto possibile, svolgere anche alcuni controlli semantici relativi alla consistenza delle informazioni contenute nel testo da analizzare.
L'obiettivo principale del *parser* resta, ovviamente, quello di definire una struttura

dati in memoria centrale e di riempirla in base alla conoscenza contenuta nei file che analizza. Tale struttura, che viene utilizzata dagli componenti del mediatore, deve avere le seguenti caratteristiche:

- deve essere rappresentativa di tutta la conoscenza che il linguaggio ODL$_{I^3}$ può esprimere;

- deve essere facile e intuitiva la sua consultazione da parte dei componenti che attingono alle informazioni memorizzate.

Qui di seguito verranno descritte le principali caratteristiche del linguaggio ODL$_{I^3}$ e come le informazioni da esso descritte sono implementate nella complessa struttura dati.

I tipi . L'ODL$_{I^3}$ prevede una distinzione fondamentale nei tipi; un tipo può essere:

- **Tipo valore**

- **Tipo classe**, chiamato comunemente **Classe**

I *tipi valore* sono propri delle variabili e degli attributi semplici, infatti, a differenza delle classi, ogni istanza di questo tipo è priva di identificatore, cioè ha come unica proprietà il suo valore.
I *ValueType* si suddividono in:

- **SimpleType**: sono tutti i tipi atomici base, cioè i tipi predefiniti (integer, float, char, boolean, etc...), i vari tipi collezione **TemplateType** (set, list, bag, array), nonché i tipi nuovi **DefinedType**: secondo la sintassi ODL$_{I^3}$ la definizione di nuovi tipi segue le stesse regole del linguaggio **ANSI C**.

- **ConstrType**: Fanno parte di questa classe i tipi **StructType**, **UnionType** e **EnumType**. Il tipo Enum si discosta da Struct e Union, perché rappresenta un semplice elenco finito di valori: tutte le variabili di questo tipo possono assumere solamente uno dei valori elencati nella definizione dello stesso.

L'altra famiglia in cui sono suddivisi i tipi dell'ODL$_{I^3}$ è quella dei *tipo-classe*: questi sono propri degli attributi complessi, delle viste e delle istanze di classi (che in ODL$_{I^3}$ sono chiamate *interface*), ed hanno un proprio OID, un comportamento ed un'interfaccia. L'unica classe che fa parte della famiglia *tipo-classe* è la classe **Interface**.

Le implementazioni di Interface. Le proprietà di ogni oggetto Interface sono:

- ereditarietà: ogni classe può avere una o più superclassi, cioè è ammessa l'ereditarietà multipla. Nello schema questa proprietà si manifesta attraverso la collezione complessa **inheritance** che mappa sulla stessa classe **Interface**.

- Source: ogni classe appartiene ad un sorgente, le cui caratteristiche sono nome e tipo (relazionale, a oggetti, file, semistrutturato). Per consentire una migliore navigazione, nella struttura dati viene mantenuta, all'interno dell'oggetto **Source**, la lista di tutte le classi che ne fanno parte. Infatti, dal momento che classi omonime possono esistere in sorgenti differenti, l'identificazione univoca di una classe è data dalla coppia $nomeSource.nomeInterface$. Il costrutto per la dichiarazione del sorgente è stato aggiunto in ODL$_{I^3}$.

- Extent: l'**Extent** rappresenta l'insieme di tutte le istanze della classe all'interno di un particolare database. La dichiarazione di un'estensione non è obbligatoria (una classe può non averne), tuttavia la sua presenza facilita il reperimento indicizzato (quindi rapido) delle informazioni da parte del DBMS.

- Chiavi e Foreign key: si possono dichiarare delle chiavi candidate, semplici o composte. Ad ogni istanza dell'oggetto **KeyList** corrisponde una chiave candidata, che mappa un attributo (chiave semplice) o più attributi (chiave composta). La possibilità di avere anche delle Foreign Key è una prerogativa del linguaggio ODL$_{I^3}$. Infatti le informazioni provenienti da uno schema relazionale sarebbero incomplete se si trascurasse questo aspetto, dal momento che nel modello relazionale le gerarchie di aggregazione si realizzano solo attraverso le foreign key.
  L'oggetto **ForeignKey**, che mantiene tale informazione, contiene la lista degli attributi componenti la chiave esterna, la lista dei medesimi attributi nella classe riferita, e la classe riferita stessa.

- Interface body: il corpo dell'interfaccia di una classe (o vista) è costituito dall'insieme di attributi e metodi che ne fanno parte. A differenza dell'ODL standard, nel linguaggio ODL$_{I^3}$ è prevista la definizione di più implementazioni per una stessa classe. Ogni implementazione è un oggetto della classe **IntBody**. Questa caratteristica consente l'acquisizione di dati semistrutturati che, per definizione, non sono tenuti a seguire una rigida formattazione, come invece avviene per i dati strutturati.

- Operazioni: le operazioni sono i metodi della classe, o meglio la loro signature: infatti tra le informazioni contenute nella classe **Operation** figurano il nome e il tipo restituito dal metodo, una lista di parametri passati (oggetti di tipo **OpVar**) ognuno avente un nome, un tipo e un elenco di eventuali dimensioni di array.

- Attributi: Una prima classificazione distingue gli attributi semplici da quelli complessi. La differenza è tutta nel tipo: tipo-valore nel primo caso e tipo-classe nel secondo. In ODL_{I^3} esiste anche la possibilità di dichiarare delle relationship, ovvero attributi complessi dei quali esiste la relazione inversa; infine il linguaggio ODL_{I^3} fornisce un costrutto per definire regole di mapping che legano grandezze appartenenti allo schema integrato e grandezze degli schemi locali. Quando alla dichiarazione di un attributo seguono un insieme di regole di mapping, lo stesso è automaticamente considerato globale.
  Si distinguono tre tipi di attributi: i **GlobalAttribute**, i **SimpleAttribute** e le **Relationship**.

  I **SimpleAttribute** hanno un tipo generico (**Type**), un flag booleano che indica se sono di sola lettura e un set di eventuali indici d'array.

  Le **Relationship** mappano un' interfaccia e contengono le informazioni sull'inversa attraverso un puntatore alla stessa classe **Relationship**, inoltre *type* indica se l'attributo è o meno una collezione, mentre *orderBy* indica un criterio di ordinamento.

  I **GlobalAttribute** hanno le stesse caratteristiche degli attributi locali, più un collegamento all'oggetto **MappingRule**

- Mapping rule: esistono vari tipi di corrispondenza fra grandezze globali e locali.
  Ad un attributo globale, con riferimento ad una classe locale, può corrispondere:

  – Un solo attributo locale; è il caso più semplice.

  – Un insieme di attributi locali in and tra loro. L'operatore and assume il significato di concatenazione.

  – Un insieme di attributi locali in $union$ tra loro; in questo caso all'attributo globale corrisponde uno solo alla volta tra quelli in $union$; il criterio che determina la scelta di quale attributo locale scegliere è dato dal valore di un terzo attributo locale, rappresentato nella struttura da **onParameter**.

- Un valore di default. È il tipico caso in cui si rende necessario esprimere un metaconcetto.

- Costanti: la sintassi delle costanti in ODL$_{I^3}$ è praticamente identica a quella dell'ANSI C.
  La classe **Constant** che le rappresenta ha tre attributi:
  - Nome della costante
  - Tipo della costante, deve essere un oggetto **BaseType**
  - Valore della costante. Per semplicità il valore viene archiviato in formato stringa: sarà cura di chi consulta la struttura dati eseguire il cast opportuno.

Osservazione: le classi **Source**, **Interface** ed **IntBody** ereditano tutte dalla classe **TypeContainer**, la quale introduce una fase di post-processing per la risoluzione dei tipi EnumType, UnionType, StructType. L'ODL$_{I^3}$ referenzia questi tipi con identificatori e perciò si è definito un insieme di nuovi tipi (**DefinedTypeToSolve**) che ereditano da **SympleType**. di fatto sono tipi temporaneamente non risolti che prevedono tutte le funzioni dei tipi risolti, contengono un puntatore al tipo "risolto" e tutti i metodi richiamati su questi oggetti vengono ridiretti al tipo "risolto". Appartiene alle sottoclassi di **Type Container** anche la classe **Schema** che descrive uno schema ODL$_{I^3}$. Ogni entità presente nello schema (interface, rule, ecc) è accessibile attraverso un'istanza di tale classe.

Relazioni terminologiche. Nel linguaggio ODL$_{I^3}$ è stata aggiunta la possibilità di inserire relazioni fra attributi ed attributi, classi e classi, e relazioni miste fra classi ed attributi. Il parser si preoccupa di interpretarle e di capire quali classi e/o quali attributi sono posti in relazione tra loro.
I tipi di relazione sono: sinonimia (SYN), ipernimia (BT), iponimia(NT), e associazione (RT). Si distinguono i seguenti casi:

- La relazione coinvolge due classi: viene creato un oggetto **InterfaceRel**
- La relazione coinvolge due attributi: viene creato un oggetto **AttributeRel**
- La relazione coinvolge una classe e un attributo: viene creato un oggetto **AttrIntRel**

Regole di integrità. È data, dal linguaggio ODL$_{I^3}$, la possibilità di definire regole di integrità di tipo *if-then*, che devono essere verificate per ogni istanza nel database.
La corretta sintassi di una regola è la seguente:

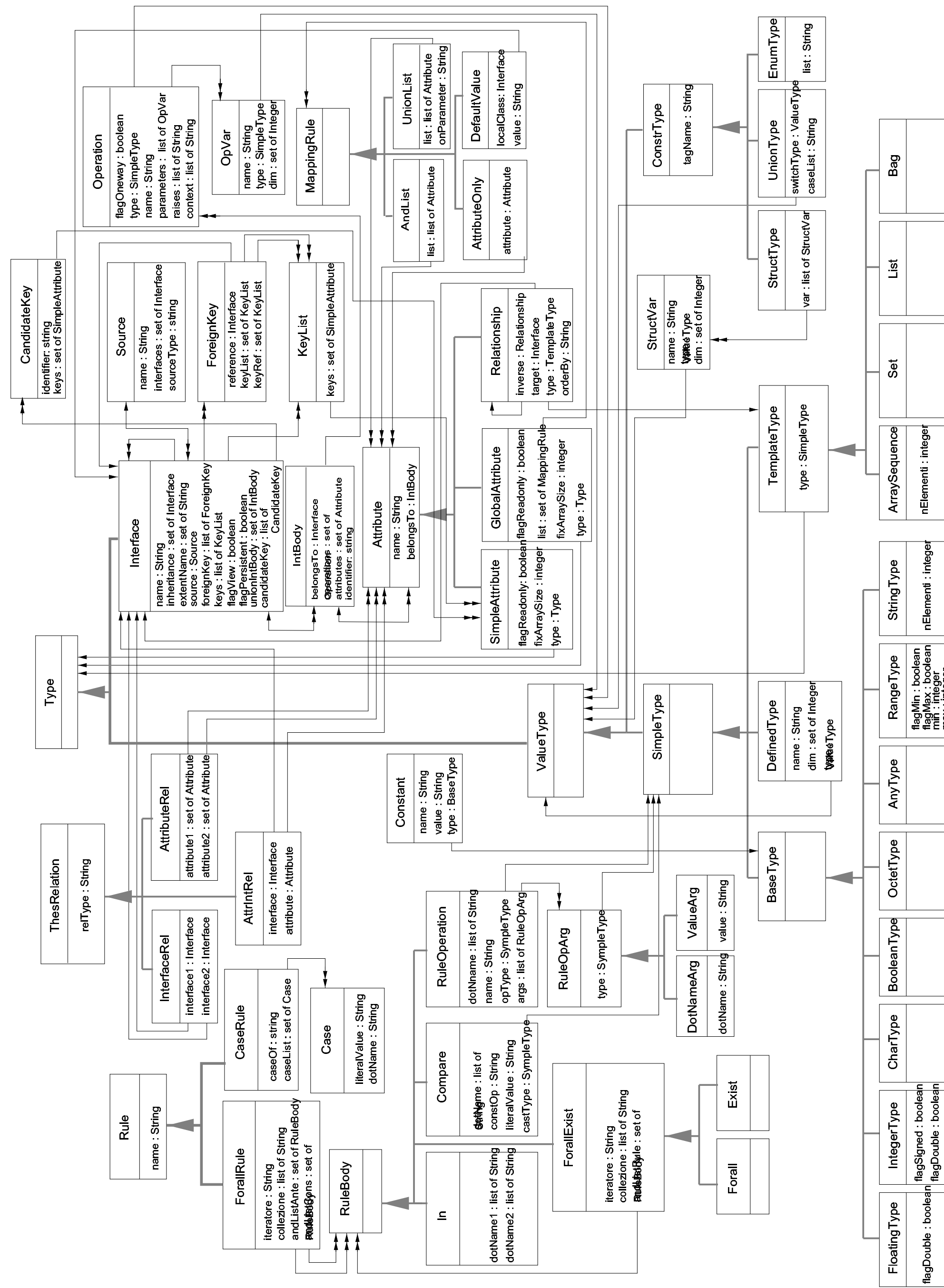

Figura 5.1: Il modello ad oggetti della struttura dati ODL_{I^3}

rule *nomerule*
  forall *iteratore* **in** *collezione* **:** *antecedente* **then** *conseguente*
oppure:
rule *nomerule*
  [{ **case of** *identifier* **:** *caselist* }]

La dichiarazione alternativa di *case*, fornisce il criterio di scelta fra più attributi locali che sono mappati in *union* da un unico attributo globale. Le condizioni *antecedente* e *conseguente* sono formate da un unico predicato booleano o da più condizioni in and tra loro. Ogni predicato è un oggetto della classe **RuleBody**. Nella struttura dati le regole sono rappresentate dalle classi **ForAllRule** e **CaseRule** che appartengono ad una gerarchia in cui la classe padre è **Rule**.

Compito del parser non è soltanto quello di acquisire informazioni e caricarle nella struttura dati sopra descritta; uno dei compiti principali che gli spettano riguarda i controlli sulla correttezza non solo sintattica, ma anche semantica. Infatti rispettare la sintassi di un particolare linguaggio non è condizione sufficiente per affermare che l'informazione è corretta: esistono regole di coerenza che possono essere garantite solamente da ulteriori controlli che interessano la sfera semantica. La Figura 5.1 mostra il modello ad oggetti della struttura dati appena descritta.

5.2 Organizzazione software del modulo SIM

Il modulo **SIM** rappresenta un valido strumento per operare con le relazioni del Common Thesaurus. Esso è in grado di estrarre relazioni intensionali dalla conoscenza della struttura degli schemi, validare relazioni inserite dal progettista e di calcolare (grazie al motore inferenziale ODB-Tools) tutte le possibili relazioni implicate da quelle già esistenti inserite dal progettista e validate.
Il software che realizza tali funzionalità è racchiuso in un package denominato **SIM**, il quale comprende l'omonima classe Java e rappresenta un componente del modulo SI_Designer. Per istanziare la classe **SIM** è necessario invocare il corrispondente costruttore a cui dev'essere passato, come unico parametro, un oggetto della classe Java **GlobalSchemaProxy**. Per comprendere il significato di quest'ultima classe è opportuno fare un passo indietro. Al fine di rendere il sistema MOMIS più aperto possibile, si è pensato di offrire al progettista la possibilità di creare oggetti CORBA contenenti tutte le informazioni relative ad uno schema globale: tali oggetti sono istanze della classe Java GlobalSchema. Per accedere agli oggetti GlobalSchema, SI_Designer utilizza un altro oggetto Java che funge da proxy: un oggetto istanza della classe Java GlobalSchemaProxy. In questo modo è possibile mettere in rete gli schemi globali di MOMIS e renderli consultabili

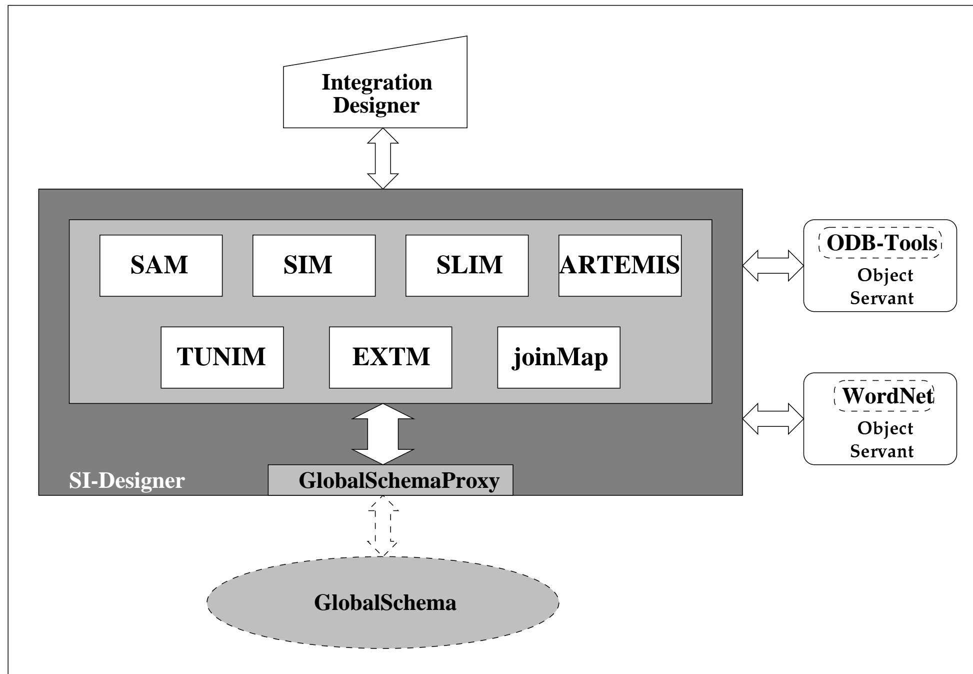

Figura 5.2: **SIM** integrato in SI_Designer

da uno o più Query Manager oppure, in futuro, da altre applicazioni. Dal punto di vista implementativo, le informazioni sullo schema globale sono memorizzate in:

- un oggetto **Schema** composto da tutte le istanze di classi Java appartenenti alla struttura dati riempita dal parser ODL_{I^3}, e da tutte le istanze delle classi che rappresentano le informazioni sullo schema globale;

- un insieme di altre proprietà che serviranno a SI_Designer per mantenere traccia delle operazioni di integrazione.

Il modulo **SIM**, preso in ingresso un oggetto della classe GlobalSchemaProxy, acquisisce tutte le informazioni relative agli schemi locali applicando il metodo *getLocalSchemata* sull'oggetto acquisito. La figura 5.2 mostra l'integrazione di **SIM** in SI_Designer.

I compiti realizzati dal modulo **SIM** possono essere distinti in:

1. operazioni da svolgere prima dell'intervento del progettista (estrazione automatica delle relazioni intraschema e primo popolamento del Thesaurus);

2. operazioni da svolgere dopo l'intervento del progettista (validazione, inferenza e secondo popolamento del Thesaurus).

In considerazione del fatto che queste due tipologie di operazioni si riferiscono a periodi temporali nettamente distinti, si è ritenuto opportuno (come già anticipato nel capitolo precedente) suddividere il modulo in due parti: **SIMA** (che svolge le operazioni al punto 1) e **SIMB** (che realizza le rimanenti). Le funzionalità di questi due sottomoduli sono implementate dai metodi *runSIMA* e *runSIMB*, entrambi facenti parte della classe Java **SIM**.

5.3 Implementazione del modulo SIMA

Il software che realizza la fase di analisi degli schemi ed il successivo popolamento del Thesaurus è costituito da due metodi principali (entrambi privati): *analisiRelazionale* ed *analisiOggetti*. Il primo si occupa di estrarre le relazioni dagli schemi relazionali, mentre il secondo prevede la connessione ad ODB-Tools per lo studio degli schemi ad oggetti. In ambedue i casi è richiesto come parametro un oggetto **Schema** ed il valore di ritorno è una stringa contente tutte le relazioni trovate dal rispettivo metodo.

5.3.1 Analisi degli schemi relazionali

Il punto di partenza è rappresentato dallo studio delle foreign key; ad esso verrà poi affiancata un'indagine sulle chiavi primarie e candidate. Le informazioni sulle chiavi esterne sono mantenute in un Vector di oggetti **ForeignKey** associato ad ogni istanza della classe **Interface**, ed il cui valore è "null" per ogni classe a cui non è associata alcuna foreign key.

L'algoritmo realizzato nel metodo *analisiRelazionale* implementa i casi già descritti nel capitolo precedente (paragrafo 4.2). Posto:

- FK(cl): una foreign key definita nella generica classe *cl*;

- REF[FK(cl)]: la classe referenziata dalla foreign key FK(cl);

- KEY(cl): l'insieme delle chiavi (primaria e candidate) di *cl*;

- SC: l'oggetto della classe **Schema** su cui si effettua l'analisi.

```
begin
  forall cl in SC do
    forall FK(cl) do
     if( FK(cl)is included in KEY(cl))
       then
         { clref = REF[FK(cl)];
           if((exists FK(clref) suchthat
              FK(clref) is included in KEY(clref))
              and(REF[FK(clref)] is cl))
               then (find relation cl SYN clref);
               else (find relation cl NT clref);
         }
       else
           if(exists K in KEY(cl)suchthat
             FK(cl) is a part of K)
               then (find relation cl RTpartof clref);
               else (find relation cl RT clref);
end
```

Figura 5.3: Algoritmo di analisi degli schemi relazionali

il procedimento implementato è mostrato in Figura 5.3;

Le informazioni sulle chiavi primarie e candidate sono ottenute utilizzando il metodo *getKeys* definito nella classe **Interface**, il quale restituisce un array contenente tutte le chiavi della classe a cui è applicato. Ogni elemento dell'array è un oggetto della classe **KeyList** e descrive una chiave. La chiave primaria si distingue dalle altre in quanto solo ad essa è associato il nome "*PrimaryKey*". Per ottenere poi gli attributi che compongono la chiave si utilizza il metodo *getAttributes* della classe KeyList. A questo proposito è opportuno fare una precisazione a carattere più generale: introdurre la possibilità di operare con sorgenti semistrutturate ha reso necessario introdurre nuovi operatori nella descrizione ODL_{I^3} degli schemi. Fra questi è importante l'operatore UNION che da la possibilità di definire diversi oggetti **IntBody** associati ad una medesima classe. Ogni istanza della classe **IntBody** descrive una possibile implementazione dell'Interface a cui fa riferimento. Non è rara la situazione in cui vi siano più attributi con lo stesso nome appartenenti a "corpi" diversi di una stessa classe (il loro tipo può essere lo stesso o cambiare).
Tutto questo rappresenta la giustificazione al fatto che, applicando il metodo *getAttributes* ad una generica chiave di cui si voglia conoscere la composizione, si ottenga una collezione di oggetti **Vector** anzichè di semplici oggetti **Attribute**. Ogni attributo, infatti, viene rappresentato da un Vector contenente tutti gli oggetti Attribute omonimi appartenenti alle diverse IntBody dell'Interface considerata.

L'ultima funzione messa a disposizione dal metodo *analisiRelazionale* riguarda l'estrazione di eventuali relazioni di sinonimia fra attributi coinvolti nelle foreign key. Si considerano, a questo proposito, solo i casi di chiavi esterne formate da un solo attributo: il nome di tale attributo viene confrontato col nome del corrispondente nella classe riferita, e nel caso i due non coincidano viene segnalata la loro sinonimia. Tutto questo viene implementato utilizzando i metodi (della classe ForeignKey):

- *getAttributeNames:* restituisce i nomi che compongono la foreign key nella classe in cui essa è definita;
- *getKeyTargetAttributeNames:* restituisce i nomi degli stessi attributi nella classe referenziata.

Si escludono da tale analisi le foreign key formate da più attributi in quanto non avrebbe senso stabilire relazioni di SYN fra gruppi di attributi.

5.3.2 Analisi degli schemi ad oggetti

L'estrazione delle relazioni intensionali dagli schemi ad oggetti è effettuata da **SIMA** utilizzando l'oggetto remoto ODB-Tools. L'accesso a tale oggetto è realizzato attraverso una connessione CORBA che prevede, innanzitutto, la creazione di un oggetto ORB (Object Request Broker). Volendo esprimere nel modo più semplice ed immediato possibile il significato di un ORB, si può dire che esso rappresenta un "*bus aperto*" sul quale oggetti diversi possono interagire via rete, indipendentemente dal sistema operativo su cui sono stati implementati. Affinchè un client possa ottenere un collegamento (ovvero un Object Reference) ad un oggetto remoto, l'architettura CORBA mette a disposizione un servizio standard implementato negli ORB e denominato *Naming Service*. Su ogni macchina dovrà esistere un registro degli oggetti CORBA accessibili in remoto; ad ogni oggetto è associato un nome ed il Naming Server gestisce proprio il registro di questi nomi. Per permettere all'ORB di connettersi al gestore dei nomi residente su una macchina remota è necessaria una sua inizializzazione in cui si specifichi:

1. l'Host dell'oggetto remoto a cui ci si deve connettere;
2. la porta su cui effettuare il collegamento.

Tutte le richieste indirizzate ad ODB-Tools sono gestite da un oggetto server (chiamato *OdbToolsFactory*), il quale crea un oggetto Servant per ogni funzionalità richiesta e gestisce i Servant già esistenti (uccidendoli, ad esempio, al termine di un certo periodo di tempo). Il software con cui si realizza la connessione appena descritta è riportato in Appendice D (Figura D.4).

Stabilita la connessione può iniziare l'interazione con ODB-Tools. Affinchè esso possa comprendere le informazioni contenute negli schemi ODL_{I^3} delle sorgenti, è necessario tradurre gli stessi nel formalismo OLCD (vedi Capitolo 2, Sezione 2.1.1). Tale compito è assolto dal metodo *toOlcdOnMap* (inserito nella classe Schema) che, preso in ingresso un oggetto della classe **TranslationMap** (package *tools*), restituisce una stringa OLCD rappresentante tutta la conoscenza espressa dagli schemi stessi. Un'istanza della classe TranslationMap identifica una mappa nella quale ad ogni oggetto si associa un nome univoco: sarà questo il nome con cui l'oggetto verrà presentato ad ODB-Tools e col quale esso sarà identificato durante l'intera elaborazione eseguita dal componente remoto. Gli oggetti che popolano la mappa appartengono ad una delle seguenti classi:

- DefinedType
- EnumType
- Rule
- ExtRule
- Relationship
- SimpleAttribute
- Interface
- StructType

Il caricamento della mappa viene effettuato utilizzando il metodo *put* della classe TranslationMap, il quale prevede due parametri: una stringa rappresentante il nome suggerito per l'oggetto da inserire e l'oggetto stesso. Nel caso il nome suggerito sia già esistente ne verrà generato uno nuovo; nel caso sia l'oggetto ad essere già presente, verrà lasciata immutata la composizione della mappa. Dal momento che ODB-Tools supporta nomi di lunghezza massima di quindici caratteri, è previsto un controllo a tale proposito: esiste una routine che permette di troncare i nomi di lunghezza superiore alla soglia preservandone l'unicità. La necessità di utilizzare una mappa come quella appena descritta è legata al fatto che, mentre l'ODL_{I^3} permette una suddivisione gerarchica dello schema mediante l'uso di moduli, l'OLCD è piatto: è questo il motivo che obbliga ad utilizzare nomi univoci per tutto lo schema e che ha indotto alla creazione della classe TranslationMap.

In relazione all'esempio di riferimento, la figura 5.4 mostra gli schemi tradotti in OLCD che **SIMA** passa ad ODB-Tools.

Passiamo ora ad esaminare i risultati dell'elaborazione eseguita dal componente

```
prim Student = CS_Person &^[year:integer,takes:<Course>,rank:string];
prim Professor = CS_Person &^ [belongs_to:Office,rank:string];
prim Location = ^[number:integer,city:string,county:string,
                street:string];
prim Office = ^ [address:Location,description:string];
prim CS_Person = ^[last_name:string,first_name:string];
prim Course = ^[course_name:string,taught_by:Professor];
prim ListOfStude = ^ [Student:<Student0>];
prim Student0 = ^[faculty_nam:string,name:string,student_cod:string,
                tax_fee:string];
prim School_Memb = ^[name:string,facolty:string,year:integer];
prim Research_Staff = ^[e_mail:string,dept_code:integer,name:string,
                section_cod:integer];
prim Room = ^[room_code:integer,notes:string,seats_numbe:integer];
prim Section = ^[section_nam:string,section_cod:integer ,
                room_code:integer,length:integer];
prim Department = ^[dept_code:integer,budget:integer,
                dept_name:string].
```

Figura 5.4: Traduzione OLCD degli schemi nell'esempio di riferimento

remoto. L'output di ODB-Tools è rappresentato da un documento XML (in realtà è una stringa XML) nel quale sono riportate informazioni relative:

- alle classi dello schema analizzato (in particolare i loro attributi specificandone il tipo);
- alle relazioni terminologiche di SYN, NT e RT fra le classi stesse.

Per poter accedere a queste informazioni è stato necessario effettuare una fase di *parsing* del suddetto documento XML. A tale scopo si è utilizzato il package JAXP (Java Api For XML Parsing) il quale mette a disposizione una serie di funzionalità per leggere, generare e manipolare documenti XML attraverso delle Java APIs. In particolare la libreria JAXP include un parser che, preso in ingresso un file XML, produce un documento DOM (Document Object Model), ovvero un'interfaccia per documenti XML raccomandata dalla World Wide Web Consortium. Tutte le informazioni costituenti il documento XML sono riportate dal parser all'interno di una struttura ad albero (Object Model Tree), in cui la radice rappresenta l'intero documento e tra i diversi nodi sono distribuite le informazioni di cui sopra. Ci saranno perciò nodi rappresentanti ciascuno una relazione (EQU, ISA, RELATION), i quali avranno come attributi (o meglio come nodi attributi) le classi coinvolte nella relazione. L'esplorazione di tale albero permette di disporre di tutte le relazioni terminologiche estratte da ODB-Tools in riferimento allo

```
<SCHEMA>

<LEVEL value="0" type="LAST">
<CLASSES_TO_DISPLAY>
<CLASS name="Student">
        <INHERIT name="CS_Person"/>
        <ATTRIBUTE type="string" name="first_name"/>
        <ATTRIBUTE type="string" name="last_name"/>
        <ATTRIBUTE type="string" name="rank"/>
        <ATTRIBUTE type="list&lt;Course&gt;" name="takes.list"/>
        <ATTRIBUTE type="long" name="year"/>
</CLASS>
<CLASS name="Professor">
        <INHERIT name="CS_Person"/>
        <ATTRIBUTE type="Office" name="belongs_to"/>
        <ATTRIBUTE type="string" name="last_name"/>
        <ATTRIBUTE type="string" name="first_name"/>
        <ATTRIBUTE type="string" name="rank"/>
</CLASS>
<CLASS name="Location">
        <ATTRIBUTE type="string" name="city"/>
        <ATTRIBUTE type="string" name="county"/>
        <ATTRIBUTE type="long" name="number"/>
        <ATTRIBUTE type="string" name="street"/>
</CLASS>
<CLASS name="Office">
        <ATTRIBUTE type="Location" name="address"/>
        <ATTRIBUTE type="string" name="description"/>
</CLASS>
<CLASS name="CS_Person">
        <ATTRIBUTE type="string" name="first_name"/>
        <ATTRIBUTE type="string" name="last_name"/>
</CLASS>
<CLASS name="Course">
        <ATTRIBUTE type="string" name="course_name"/>
        <ATTRIBUTE type="Professor" name="taught_by"/>
</CLASS>
...
</CLASSES TO DISPLAY>
<EQU>
        <RELEQU c1= "Student" c2="Student"/>
        <RELEQU c1="Professor" c2="Professor"/>

 ...
</EQU>
<CLASSES_COLORS>
        <color class="Student" value="#CCCCCC"/>
        <color class="Professor" value =#CCCCCC"/>
...
</CLASSES_COLORS>
<ISA>
        <RELISA c1="Student" c2="CS_Person"/>
        <RELISA c1= "Professor" c2="CS_Person"/>
...
</ISA>
<RELATIONS_TO_DISPLAY>
        <RELATION target="Course" source="Student" path  ="takes.list"/>
        <RELATION target="Office" source="Professor" path="belongs_to"/>
        <RELATION target="Location" source="Office" path="address"/>
        <RELATION target="Professor" source="Course" path="taught_by/>
        <RELATION target="Student0" source="ListOfStude" path="stude.list/>
</RELATIONS_TO_DISPLAY>
</LEVEL>

</SCHEMA>
```

Figura 5.5: Documento XML contenente l'output di ODB-Tools

schema analizzato. A questo punto si può popolare il Common Thesaurus con le stesse.
La Figura 5.5 mostra la stringa XML restituita da ODB-Tools a seguito della sua elaborazione.

5.3.3 Popolamento del Thesaurus

Tutte la relazioni intra-schema ottenute dal modulo **SIMA** attraverso l'analisi degli schemi sono, in un secondo momento, inserite nel Common Thesaurus. Ogni relazione è rappresentata da un'istanza della classe **ThesRelation**. Ad ogni oggetto di questa classe sono associate le informazioni relative al tipo di relazione (SYN, BT, NT o RT), agli elementi coinvolti, al fatto che la relazione sia stata o meno validata ed alla natura della stessa (intensionale o estensionale).
Il popolamento del Thesaurus può essere realizzato utilizzando uno fra i tre metodi definiti nella classe **TypeContainer**:

- *addThesRelation*: accetta come parametro l'oggetto ThesRelation rappresentante la relazione da inserire, consentendo l'immissione di una sola relazione per volta;
- *addThesRelations*: accetta come parametro un array di oggetti ThesRelation permettendo, così, l'inserimento di più relazioni per volta;
- *addThesRelationWithControl*: ha le stesse funzionalità del primo metodo descritto, ma in più fornisce dei controlli che permettono di evitare rindondanze ed anomalie nel contenuto del Common Thesaurus.
  Innanzitutto è previsto un "filtro" che permette l'inserimento di una relazione solo se essa non è già presente nel Thesaurus (per ogni relazione RT e SYN l'inserimento è impedito anche se è già presente la relazione simmetrica). Il tentativo di duplicare una relazione genera un'eccezione (OdlException).
  L'altro controllo è finalizzato ad evitare che esistano nel Common Thesaurus relazioni diverse fra la stessa coppia di elementi. Fra i quattro tipi di relazioni contemplate può essere definita una gerarchia che vede a capo la relazione di sinonimia (è la più "pesante"), seguita dalle relazioni BT ed NT (allo stesso livello) ed infine la relazione RT. Prima di inserire una nuova relazione nel Thesaurus si verifica, quindi, se ne esiste già una che coinvolge la stessa coppia di elementi: in caso affermativo si immette la nuova relazione, eliminando la "vecchia", solo se la prima ha maggior peso della seconda.

Il modulo **SIM** utilizza l'ultimo metodo descritto per ogni inserimento da effettuare (non solo in relazione alla fase iniziale, ma anche per memorizzare le relazioni

inferite).
Infine è bene ricordare che, nel momento in cui si crea una nuova relazione, viene associata alla stessa l'informazione relativa al produttore attraverso l'attributo *producerId*. Questo risulta di particolare utilità nel caso in cui si vogliano identificare tutte le relazioni inserite da un determinato modulo, per poi manipolarle o eliminarle. Un metodo utilizzato dal modulo **SIM** a tale riguardo è denominato *delThesRelationForProducer*, della classe TypeContainer: esso accetta l'identificatore di un produttore ed elimina dal Thesaurus tutte le relazioni che da lui provengono. In particolare, ad invocare tale metodo è **SIMA** il quale, ogni volta in cui viene chiamato, pulisce il Thesaurus da tutte le relazioni già presenti prodotte da se stesso e da **SIMB**.

5.4 Implementazione del modulo SIMB

Il software che realizza la fase di validazione ed inferenza è racchiuso all'interno del metodo *runSIMB* della classe **SIM**. In entrambe le fasi si utilizzano le funzionalità di ODB-Tools: questo rende necessaria una connessione CORBA (analoga a quella descritta nella Sezione 5.3.2) che verrà chiusa al termine dell'elaborazione. Nei prossimi paragrafi verranno illustrati i metodi più significativi che caratterizzano il modulo **SIMB**; sarà inoltre descritta la classe ThesaurusByObject (package tools), in quanto essa rappresenta un valido supporto per la procedura di inferenza.

5.4.1 Software per la validazione delle relazioni tra attributi

Anche la fase di validazione delle relazioni fra attributi è realizzata utilizzando le funzionalità di ODB-Tools, il quale comprende, al suo interno, algoritmi in grado di valutare la compatibilità e le relazioni fra domini. Il software che implementa tale procedimento è composto da due metodi: *fillMapRel* e *relAttrAnalysis*, entrambi appartenenti alla classe **SIM** e richiamati dal metodo *runSIMB*. Il primo elabora l'output di ODB-Tools e carica in una mappa (ovvero in un oggetto della classe HashMap) le informazioni utili per la validazione; il secondo effettua la vera e propria fase di validazione confrontando le informazioni contenute nella suddetta mappa con quelle espresse dalle relazioni sottoposte a controllo.
Vediamo ora più al dettaglio le operazioni che caratterizzano la fase di validazione.
Il primo problema che si è dovuto affrontare è stato capire come proporre le relazioni da esaminare ad ODB-Tools in modo che esso potesse fornire informazioni interessanti ai fini della validazione. In realtà, ciò che si è dovuto passare ad ODB-Tools non sono state le singole relazioni ma, ancora una volta, degli schemi

```
prim Student = CS_Person &^[year:integer,takes:<Course>,rank:string];
prim Professor = CS_Person &^ [belongs_to:Office,rank:string];
prim Location = ^[number:integer,city:string,county:string,
                  street:string];
...
prim Section = ^[section_nam:string,section_cod:integer ,
                 room_code:integer,length:integer];
prim Department = ^[dept_code:integer,budget:integer,
                    dept_name:string].

virt year = [attr : string];
virt takes = [attr : <Course>];
virt rank = [attr : string];
virt belongs_to = [attr : <Office>];
virt rank0 = [attr : string];
virt number = [attr : integer];
...
virt dept_code = [attr : integer];
virt budget = [attr : integer];
virt dept_name = [attr : string];
```

Figura 5.6: Traduzione OLCD degli schemi per la validazione delle relazioni tra attributi

tradotti in OLCD ed arricchiti della conoscenza relativa ai domini coinvolti nelle relazioni stesse: quello che ODB-Tools è chiamato ad esplicitare sono i rapporti di equivalenza o inclusione fra tali domini. Il fine ultimo dell'intero procedimento consiste nel verificare se fra i domini coinvolti esiste un legame compatibile col tipo di relazione esaminata, sulla base delle regole esposte nel precedente capitolo (Sezione 4.3.1).

La fase di arricchimento degli schemi da presentare al componente esterno prevede, innanzitutto, che ad ogni attributo coinvolto in una relazione da validare si associ una nuova *view*, che sarà inserita nello schema da esaminare: essa contiene un solo attributo il cui dominio coincide con quello dell'attributo al quale è associata. Ogni *view* generata ha un nome unico (nome col quale è presentata ed identificata da ODB-Tools) ed il legame con l'attributo a cui corrisponde è preservato all'interno della TranslationMap (vedi Sezione 5.3.2).
Lo schema così ottenuto è elaborato da ODB-Tools, il quale fornirà in output (sempre sotto forma di documento XML da interpretare) tutte le relazioni di equivalenza, ereditarietà ed aggregazione trovate; fra queste sono significative, ai fini della validazione, solo quelle che interessano le *view* precedentemente aggiunte, in quanto esse rappresentano delle relazioni fra i domini degli attributi coinvolti nei legami da validare. Come già precedentemente anticipato, tali informazioni

vengono memorizzate in una HashMap (chiamata *mapparel*) sulla quale si basa la parte conclusiva della procedura. La suddetta mappa associa, ad ogni *view*, l'insieme delle relazioni (con altre *view*) che la coinvolgono e che sono state estratte da ODB-Tools. Ricordo che a ciascuna *view* è associato il dominio di un particolare attributo e che quindi, in pratica, la mappa riporta, per quel dominio, tutte le possibili relazioni (valide) con altri domini (associati a loro volta ad altri attributi). A questo punto si dispone di tutte le informazioni per eseguire la vera e propria fase di validazione: per verificare la compatibilità dei domini con le relazioni che li interessano, è sufficiente consultare la suddetta mappa.
Supponiamo, ad esempio, di dover validare una relazione NT tra due attributi: la validazione ha esito positivo solo se il dominio del primo attributo è equivalente o contenuto nel dominio del secondo. Consultando la mappa *mapparel* si giunge velocemente ad una risposta: entrando con la *view* associata al primo attributo si verifica se, nella lista di valori ad essa associati, esiste una relazione SYN o NT in cui il secondo elemento è proprio la *view* associata al secondo attributo: in caso affermativo la relazione è valida.
Le relazioni fra attributi che non superano la fase di validazione non sono cancellate dal Common Thesaurus (o trasferite in una sorta di Thesaurus di uscita, come avveniva nel primo prototipo), ma semplicemente marcate come "relazioni non valide" settando a *false* un apposito flag (chiamato *validated*), presente in tutte le istanze della classe ThesRelation.
In Figura 5.6 è riportata la traduzione OLCD degli schemi che verranno passati ad ODB-Tools durante la fase di validazione (l'esempio è sempre quello di riferimento). Da notare la presenza delle *virt* (che traducono le *view* del linguaggio ODL_{I^3}) che si aggiungono alla descrizione già riportata in Figura 5.4.

Osservazione: Come già ribadito in paragrafi precedenti, ogni attributo è rappresentato da un Vector di oggetti Attribute omonimi, appartenenti ciascuno ad una diversa interface body della classe considerata. Validare una relazione fra una coppia di attributi significa, quindi, validare un legame fra due Vector di oggetti Attribute: essendo le diverse IntBody in alternativa fra loro (è un OR che le lega e non un AND), si considera valida la relazione esaminata se esiste almeno una coppia di oggetti Attribute (ciascuno appartenente ad un diverso Vector fra i due considerati) che supera con successo la procedura di validazione.

5.4.2 La classe ThesaurusByObject

La classe **ThesaurusByObject** (package *tools*) permette di velocizzare il trattamento delle relazioni presenti nel Common Thesaurus. Grazie ad un ricco insieme di metodi che essa mette a disposizione, risultano molto più rapide le operazioni di ricerca di relazioni soddisfacenti determinati requisiti, o il recupero delle classi che ne sono coinvolte. Per istanziare la classe ThesarusByObject può essere uti-

lizzato uno dei due costruttori disponibili: il primo richiede come parametro un oggetto della classe Schema (da esso si otterranno tutte le relazioni utilizzando il metodo *getThesRelations*, della classe TypeContainer), il secondo richiede come parametro direttamente un set di relazioni. In entambe i casi il costruttore utilizza il metodo *fillMaps* che, preso in ingresso un array di oggetti ThesRelation, carica le informazioni ad essi relative in due mappe (ciascuna istanza della classe Java HashMap).
Ogni elemento della prima mappa (chiamata *_firstEleMap*) prevede, come campo chiave, un'istanza della classe Object che rappresenta un attributo o una classe. Il valore associato a tale chiave è la lista di tutti gli oggetti della classe ThesRelation aventi come primo elemento la classe (o l'attributo) corrispondente alla chiave stessa.
Ogni elemento della seconda mappa (chiamata *_secondEleMap*) prevede come campo chiave un'istanza della classe Object: essa rappresenta il secondo elemento di ogni relazione che compare nella lista associata (come campo valore) a tale chiave.
È evidente come l'uso delle mappe appena descritte renda immediato, ad esempio, il recupero di tutte le relazioni in cui una certa classe (o attributo) compare al primo (o al secondo) membro. I metodi della classe ThesaurusByObject più significativi nel nostro contesto (in quanto più frequentemente utilizzati da **SIM**) sono:

- *getRelation*: questo metodo si propone in due diverse definizioni. La prima, preso in ingresso un elemento (classe o attributo), restituisce un set di tutti gli oggetti ThesRelation che lo coinvolgono; la seconda prevede come parametri due elementi e fornisce un set di tutte le relazioni che li comprendono entrambi;

- *getSynRelations*: dato un elemento, restituisce tutte le relazioni di sinonimia presenti nel Common Thesaurus che lo riguardano;

- *getInterfaceParentRelations*: fornito in ingresso un oggetto della classe Interface, restituisce un set di oggetti InterfaceRel in cui il tipo di relazione è NT e l'Interface data è al primo membro. Dalle relazioni ottenute è possibile ricavare tutti i padri della classe indicata semplicemente estraendo il secondo membro delle stesse.

- *getAttributeParentRelations*: analogo al precedente metodo, ma applicato ad un attributo. Speculare ad esso è il metodo *getAttributeChildRelation*, che fornisce tutte le relazioni fra attributi di tipo BT in cui il parametro dato compare al primo membro.

- *getInterfaceRtToTheGiven*: dato un oggetto della classe Interface, restituisce il set di tutte le classi legate ad esso da una relazione RT.

- *getGroupSynInterface*: dato un oggetto Interface, fornisce un Vector contenente tutte le classi ad esso sinonime (direttamente o indirettamente).

Si farà uso dei suddetti metodi sia nella successiva procedura di identificazione dei cicli, sia nella fase di modifica degli schemi per l'inferenza.

5.4.3 Identificazione dei cicli

Prima eseguire la fase di inferenza di nuove relazioni sulla base di quelle già presenti nel Common Thesaurus, il modulo **SIM** controlla che in esso non vi siano relazioni di ereditarietà cicliche. L'algoritmo implementato è basato su una particolare tecnica di ricorsione, chiamata *back tracking*, tipicamente utilizzata per l'esplorazione di alberi. Il problema di ricerca di relazioni NT cicliche può essere, infatti, ricondotto (o meglio assimilato) a quello di esplorazione di un albero al fine di trovare tutti i percorsi che, partendo da un certo nodo, ritornano al nodo stesso. Interessiamoci ora solo dei cicli NT che coinvolgono oggetti della classe InterfaceRel (quindi relazioni fra classi). In questo caso ogni nodo dell'albero rappresenta un oggetto Interface e gli archi orientati esprimono le relazioni NT fra gli stessi (la freccia sarà rivolta verso la classe più generale). La tecnica del *backtracking* considera tutti i cammini che partono da un determinato nodo e li percorre, uno alla volta, fino alla loro fine; terminato un cammino si torna indietro per esaminare i rimanenti, e solo dopo che si sono valutati tutti i percorsi che hanno come padre un certo nodo, questo può essere eliminato. Vediamo ora i passi implementati.

Supponendo di chiamare:

- SC: schema esaminato;

- i: generica istanza della classe Interface appartenente allo schema esaminato;

- *list*: lista contenente le classi ancora da valutare appartenenti allo schema esaminato;

- *storia*: vettore di relazioni rappresentanti il cammino esaminato fino ad un dato momento.

```
Procedure findCycle ():
  begin
     for each i in SC do
       if ( i in list)
          then
             {crea ed azzera il Vector storia;
              esegui la procedura esplora(i,storia,list) e
                               metti il risultato nel Vector vet;
              if(vet <> null)
             }   then segnala il ciclo;
  end
Vector Procedure esplora(Interface nodo,Vector storia,HashSet list)
  begin
     if (nodo compare in storia come primo elemento di
                                            una relazione)
       then {trovato ciclo in storia;
             termina ritornando storia;
            }
       else
           { for each (relazione r=(s nt d) dove s=nodo) do
               { if(d in list)
                   then{storia = storia + r;
                        v = esplora(d,storia,list);
               }       }
             list = list - nodo;
             ritorna 'null';
  end      }
```

Figura 5.7: Algoritmo di ricerca dei cicli di relazioni NT

l'algoritmo di *back tracking* è riportato in Figura 5.7.
In pratica, si ha un ciclo ogni volta in cui si aggiunge al Vector *storia* una relazione NT il cui secondo elemento coincide col primo termine di una relazione già presente nel Vector stesso.

5.4.4 Software per la creazione dello schema virtuale

L'ultimo obiettivo che il modulo **SIM** realizza è l'inferenza di nuove relazioni sulla base di quelle già presenti e validate nel Common Thesaurus. Affinchè questo sia possibile si utilizza il motore inferenziale ODB-Tools, al quale vengono passati gli schemi opportunamente modificati per esprimere tutta la conoscenza racchiusa nel Common Thesaurus. Come già anticipato nel capitolo precedente (Sezione 4.3.2), gli schemi che subiscono tale intervento non sono quelli originali: le relazioni semantiche inserite dal progettista potrebbero, infatti, essere in conflitto con la struttura delle classi coinvolte.
Il primo prototipo **SIM** racchiudeva tutte le informazioni dedotte dagli schemi all'interno di liste (ricordo che il linguaggio utilizzato era l'ANSI C); durante la fase di inferenza tali liste erano duplicate e proprio su queste copie si operavano le mo-

difiche. Oggi le informazioni relative agli schemi sono contenute all'interno di un numero piuttosto considerevole di classi Java: risulta quindi più complicato produrne una copia su cui riportare tutte le modifiche. La soluzione che si è ritenuta più soddisfacente è stata quella di definire un metodo che, nel tradurre gli schemi originali nel formalismo OLCD (operazione indispensabile per poter dialogare con ODB-Tools), tenga conto anche delle relazioni presenti nel Common Thesaurus. Tale funzione (chiamata *toOlcdSimBOnMap* ed inserita nella classe Schema) non traduce quindi gli schemi originali, ma ciò che risulta fondendo la conoscenza in essi contenuta con quella espressa dalle relazioni validate. In questo modo si crea uno schema virtuale e si presentano ad ODB-Tools tutte le informazioni di cui ha bisogno per realizzare la fase di inferenza.
Vediamo ora come le relazioni presenti nel Thesaurus influenzino la modifica degli schemi.

Utilizzo delle relazioni tra attributi La prima fase di questo processo consiste nel considerare le informazioni che possono derivare dalle relazioni terminologiche definite tra attributi. Per fare questo, tutti gli attributi coinvolti in una o più relazioni sono riorganizzati all'interno di una gerarchia che tiene conto del tipo di relazioni tra di essi definite, con la conseguente creazione di un insieme di alberi gerarchici i cui nodi sono costituiti dagli attributi. In particolare, sono considerate le relazioni `BT` e `NT` per generare legami di specializzazione, mentre le relazioni `SYN` danno luogo a legami di equivalenza.
Oltre alla creazione di queste gerarchie, l'algoritmo di riorganizzazione provvede a definire automaticamente i termini designati come *capo_albero*, ovvero i termini eletti come rappresentanti dell'albero di cui sono a capo, scegliendoli tra gli attributi che occupano il livello più alto di ogni gerarchia. I termini designati come *capo_albero* andranno quindi a sostituirsi, negli schemi originali, a tutti i termini che fanno parte dell'albero di cui sono a capo, essendo o sinonimi di questi, o comunque termini più generali (successivamente a questa fase, sarà allora necessaria una fase di rimozione degli attributi duplicati all'interno delle classi stesse).

Esempio 8 *Con riferimento all'esempio universitario, alcuni degli alberi gerarchici creati da* **SIMB** *sono mostrati in Figura 5.8. Si può notare che, per quanto riguarda gli attributi* `first_name` *e* `last_name` *(entrambi appartenenti alla classe* `CS_Person`*), dovranno entrambi essere sotituiti dall'attributo* `name`*, a capo del loro albero (infatti sia in Research_Staff che in School_Member il nome è identificato da un unico vocabolo: name). Di conseguenza, nella fase successiva, all'interno di* `CS_Person` *andrà rimosso un attributo tra i due designati con* `name`*, onde evitare inconsistenze nelle descrizioni degli schemi.*

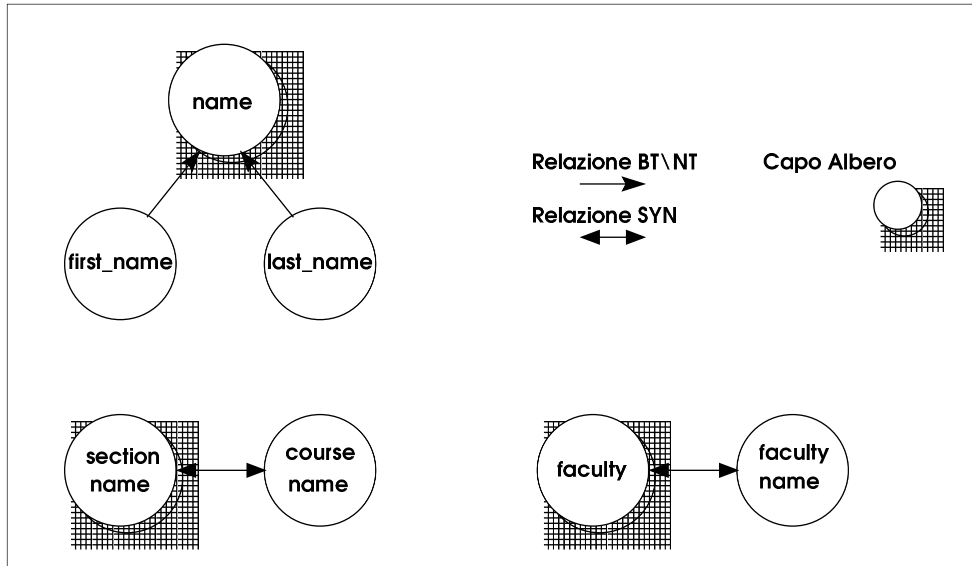

Figura 5.8: Alberi gerarchici

Si ricorda inoltre che, per semplicità, nel corrente esempio i nomi locali sono identificativi dell'attributo, mentre in realtà, all'interno del modulo, tutti i concetti sono identificati dalla coppia <nome_sorgente,nome_locale>*: saranno dunque presenti, sia nel Thesaurus, sia nelle gerarchie generate, relazioni di sinonimia del tipo:* ⟨`tax_position_xml.Student.name` SYN `University.School_Member.name`⟩

Da notare che, a questo livello, possono verificarsi problemi dovuti a notazioni incoerenti, o almeno non univoche, fornite dal progettista: è il caso in cui un determinato attributo fa parte contemporaneamente di due alberi diversi. Non potendo scegliere in modo automatico con quale termine questo vada sostituito, il sistema rileva e comunica all'utente questa incoerenza.
Tornando agli aspetti implementativi, si descriverà qui di seguito il procedimento seguito per ottenere i capialbero e le informazioni ad essi associate. Il software che realizza questa fase è racchiuso all'interno di un metodo (chiamato *getAttributeReplace* ed appartenente alla classe SIM) il quale, presa in ingresso un'istanza della classe Schema, restituisce un'oggetto HashMap che associa, ad ogni attributo, quello più generale o sinonimo con cui sostituirlo (se esiste). I passi attraverso i quali si giunge alla definizione della suddetta mappa sono:

1. creazione della mappa *map_nt*: si crea una mappa (che può essere definita "mappa dei padri") nella quale ad ogni attributo si associa la lista dei

suoi padri (saranno tutti gli attributi che compaiono al secondo membro di relazioni NT in cui quell'attributo compare al primo);

2. analisi delle relazioni di sinonimia: innanzitutto viene definita una funzione (chiamata *getSynGroup*) che restituisce tutti i gruppi di sinonimi presenti nello schema esaminato: ogni gruppo sarà un Vector e conterrà attributi che, sulla base delle relazioni presenti nel Common Thesaurus e validate, risultano sinonimi. Tra tutti questi sinonimi ne verrà scelto uno che andrà a rimpiazzare tutti i rimanenti attributi di quel gruppo: in questo modo si evita di mantenere nello schema attributi diversi (e con nomi diversi) che però sono sinonimi. In una mappa (chiamata *map_syn*) verranno memorizzate le informazioni relative all'attributo scelto ed all'insieme di attributi che esso rimpiazzerà.

3. aggiornamento della mappa *map_nt*: Arrivati a questo punto, è necessario modificare la "mappa dei padri" sulla base della conoscenza espressa dalla mappa dei sinonimi. Le correzioni da operare sono di due tipi:

   - tutti gli attributi che hanno come padre un attributo rimpiazzato da un suo sinonimo, vedranno il padre sostituito con tale sinonimo;
   - si eliminano dalla "mappa dei padri" tutte le informazioni associate agli attributi rimpiazzati da un proprio sinonimo; allo stesso tempo, si associa ad ogni attributo "rimpiazzo" una lista contenente tutti i padri di ogni elemento da esso rimpiazzato.

   In questo modo si ottiene una mappa definitiva, in cui non compaiono termini sinonimi, e dalla quale si deduce la "mappa dei figli" (chiamata *map_bt*), che associa ad ogni attributo la lista dei propri figli.

Utilizzando le suddette mappe, è possibile individuare i cosidetti *capialbero*, ovvero quegli attributi che sono padri di qualche altro attributo e che non sono figli di nessuno.
La gerarchia sottostante ad ogni capoalbero, ovvero l'insieme di attributi che esso dovrà sostituire (figli, nipoti, ecc), può essere velocemente individuata consultando la "mappa dei figli".

Utilizzo delle relazioni tra classi Così come è stato fatto per le relazioni tra attributi, **SIMB** analizza le relazioni tra classi per modificare gli schemi sorgenti (quando necessario). In particolare, sono effettuate le seguenti modifiche:

- relazioni BT o NT: nella classe più specializzata viene inserita la classe più generale nelle lista di classi da cui la prima eredita (nel caso non sia già presente);

```
prim  Student = CS_Person &^[year:integer, name:string, tax_fee:string,
                            takes:<Course>, faculty_nam:string ,
                            rank:string,student_cod:string
                            rtArg:ListOfStude,rtArg0:Course];
prim Professor = CS_Person &^[belongs_to:Office,rank:string,
                              rtArg1:Office,rtArg2:Course];
...
prim  Student0= CS_Person &^[tax_fee:string,name:string,year:integer
                            faculty_nam:string,takes:<Course>,
                            student_cod:string,rank:string,
                            rtArg:ListOfStude,rtArg0:Course];
prim School_Memb = CS_Person&^ [year:integer,name:string,tax_fee:string
                                faculty_nam:string,takes:<Course> ,
                                student_cod:string, rank:string,
                                rtArg:ListOfStude,rtArg0:Course];
prim Research_Staff = CS_Person &^[name:string,section_cod:integer,
                                   dept_cod:integer;e_mail:string,
                                   rtArg15:Section,rtArg16:Department ];
prim Room = ^[room_code:integer,notes:string,seats_numbe:integer,
             rtArg17:Section ];
...
prim  Department = ^[dept_code:integer,budget:integer,
                    dept_name:string,rtArg22:Research_Staff  ].
```

Figura 5.9: Traduzione OLCD degli schemi per la fase di inferenza

- relazioni RT: viene inserito in una delle due classi un attributo con dominio complesso che *mappa* nell'altra, per instaurare tra le due una gerarchia di aggregazione;

- relazioni SYN: le classi legate da una relazione di sinonimia vengono rese strutturalmente equivalenti, duplicandone tutte le proprietà;

La Figura 5.9 mostra la traduzione OLCD degli schemi modificati (in neretto sono evidenziate le modifiche apportate). Da un confronto con gli schemi riportati in Figura 5.4, si possono notare alcuni cambiamenti significativi:

- le classi *Computer_Science.Student*, *University.School_Member* e *tax_position_xml.Student* hanno la stessa stuttura (ovvero lo stesso insieme di attributi): dalle relazioni presenti nel Common Thesaurus, esse risultano, infatti, sinonime. Il corpo di attributi ad esse associato è il risultato dell'unione dei singoli gruppi di attributi delle classi sinonime (gli attributi uguali compaiono una sola volta);

- Anche le classi *University.Research_Staff* e *University.School_Member* ereditano da *Computer_Science.CS_Person*;

- In alcune classi esistono attributi di tipo complesso che mappano in altre classi: è il caso, ad esempio, di *University.Room* che contiene un attributo di tipo Section, e di *University.Department* che ha un attributo di tipo Research_Staff: in entrambe i casi si esprime una relazione RT fra le due classi coinvolte aggiunta dal progettista.

5.5 Funzionalità dell'interfaccia

L'interfaccia di SI_Designer è già stata mostrata nel capitolo 3 (Figura 3.2): essa è formata da diversi pannelli, ciascuno dei quali racchiude le funzionalità di un modulo del progetto. Le relazioni inserite nel Common Thesaurus da **SIM**, e quelle lessicali estratte da SLIM, sono visualizzate all'interno del pannello denominato "Thes.Rel". In esso si possono distinguere tre parti:

- sulla sinistra è riportato uno schema ad albero che mostra le sorgenti, le classi e gli attributi degli schemi acquisiti attraverso il pannello "Source";
- al centro si colloca una tabella che racchiude tutte le informazioni sulle relazioni presenti nel Common Thesaurus. Per ciascuna relazione sono visualizzati gli elementi che la compongono (nome completo della classe sorgente e destinataria), il tipo di relazione, l'identificatore del produttore ed il flag di validazione;
- al di sotto della suddetta tabella, esiste un pannello di testo che mostra l'algoritmo di *tracing* del modulo lanciato (ovvero tutti i passi eseguiti per ottenere l'output riportato in tabella).

I bottoni presenti hanno le seguenti funzionalità:

- **runSIMA**: si utilizza prima dell'intervento del progettista e permette di eseguire la fase di estrazione automatica delle relazioni dagli schemi. Premendo tale pulsante, con riferimento all'esempio dell'Università, il risultato ottenuto è mostrato in Figura 5.10; le relazioni estratte sono contraddistinte dal codice '900', corrispondente al produttore **SIMA**.
- **runSIMB**: si utilizza dopo l'intervento del progettista per validare le relazioni fra attributi ed inferirne di nuove. A seguito della validazione si aggiorna l'ultima colonna della tabella. Premendo tale pulsante, con riferimento all'esempio dell'Università, il risultato ottenuto è mostrato in Figura 5.11: le relazioni inferite sono contraddistinte dal codice '902' (corrispondente al produttore **SIMB**);

- **Add**: permette di aggiungere manualmente una relazione;
- **Delete**: permette di eliminare una relazione;
- **fg_valid**: permette la validazione manuale di una relazione;
- **Clear**: permette di ”pulire” il pannello di testo sottostante.

Il software che implementa la suddetta interfaccia è racchiuso nel package **thesRelationEditor**, il quale utilizza le funzionalità della classe **Swing** appartenente alla libreria Java Foundation Classes.

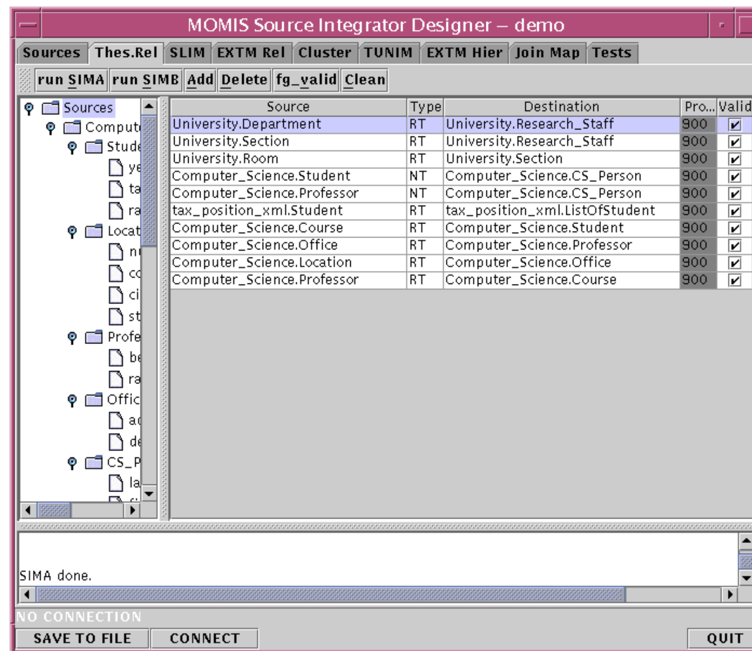

Figura 5.10: Visualizzazione del Thesaurus prima dell'intervento del progettista

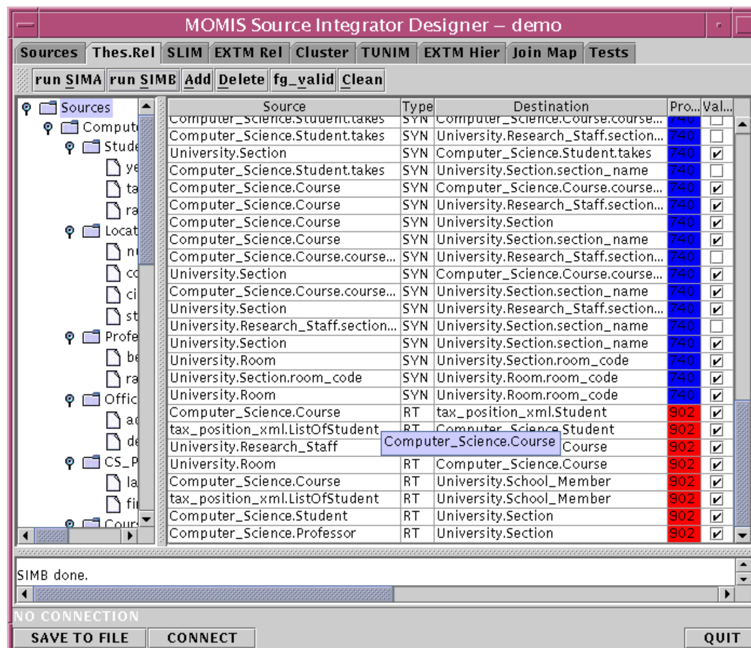

Figura 5.11: Visualizzazione del Thesaurus dopo l'intervento del progettista

Conclusioni

Il lavoro presentato in questa tesi si colloca all'interno di un progetto di integrazione di sorgenti eterogenee di dati: il sistema MOMIS. L'obiettivo di MOMIS è realizzare un mediatore in grado di integrare sorgenti eterogenee, al fine di poter interrogare uno schema *globale* integrato. Una delle principali caratteristiche, che distingue questo sistema da altri progetti simili, consiste nell'automazione della fase di integrazione degli schemi: partendo da questi ultimi è in grado di ottenere in modo semi-automatico una vista globale sulle sorgenti stesse. In questo modo l'utente finale può interrogare una moltitudine di sorgenti, pur essendo "svincolato"dalla conoscenza della loro organizzazione e della distribuzione fisica delle informazioni all'interno delle stesse. Ogni *query* viene formulata sullo schema della vista creata, lasciando al sistema il compito di interagire con le singole sorgenti nel linguaggio specifico.
Nel sistema MOMIS possiamo individuare due *macrofunzioni*: la creazione dello schema globale (realizzata dal modulo **Global Schema Builder**), e l'elaborazione di strumenti volti ad interrogarlo (a tale scopo si è progettato il modulo **Query Manager**).
Il lavoro presentato in questa tesi ha portato alla realizzazione di un modulo software che si inserisce all'interno del Global Schema Builder, e le cui funzioni sono molteplici: estrazione automatica di relazioni terminologiche dagli schemi, validazione delle relazioni fra attributi inserite dal progettista, inferenza di nuove relazioni sulla base degli schemi e delle relazioni già precedentemente estratte. Tutte le operazioni appena citate sono state realizzate con l'ausilio del componente esterno ODB-Tools, a cui si è acceduto attraverso una connessione CORBA.
Uno dei principali obiettivi che il modulo implementato si è posto è stato quello di individuare in modo *automatico* il maggior numero di relazioni tra le classi: maggiore sarà l'insieme di tali relazioni e minore il compito di inserimento manuale di nuove relazioni da parte del progettista. Per raggiungere tale obiettivo è stata necessaria un'approfondita analisi degli schemi sorgenti (specie di quelli relazionali), ed un'attenta fase di inferenza.

A livello implementativo si è cercato di realizzare tutto questo producendo un codice il più possibile modulare e documentato, in modo da rendere semplici eventuali modifiche future (questo aspetto riveste un'importanza notevole all'interno di un progetto di ricerca in continua evoluzione).

Un possibile sviluppo futuro consiste nell'adattare il modulo realizzato ad un ambito più dinamico, ovvero ad un contesto in cui si possa eliminare una sorgente già integrata in uno schema globale, oppure integrare in esso eventuali modifiche agli schemi delle sorgenti già acquisite (tale ambito è per ora solo in fase di studio).
Un'altra problematica che in futuro dovrà essere maggiormente approfondita ed integrata riguarda la conoscenza estensionale in MOMIS: ogni relazione estensionale *implica* una relazione intensionale che influenza il processo di generazione dello schema globale.

Approssimativamente sono state prodotte 3000 linee di codice commentato, implementate utilizzando la versione 1.2 del Java Development Kit della Sun (jdk1.2) disponibile sul Web presso http://java.sun.com.

Il lavoro realizzato in questa tesi è parte del progetto MOMIS presentato con successo alla conferenza internazionale Very Large Data Base ({VLDB200}, Cairo (Egitto), 10-14 Settembre 2000).

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⟩	::=	$= \mid \geq \mid \leq \mid > \mid <$
⟨rule_cast⟩	::=	**(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩	::=	⟨identifier⟩ | ⟨identifier⟩**.**⟨dotted_name⟩
⟨forall⟩	::=	**for all** | **forall**

Appendice C

Esempio di riferimento in ODL$_{I^3}$

Di seguito é riportata la descrizione, attraverso il linguaggio ODL$_{I^3}$, dell'esempio di riferimento citato nella trattazione della tesi.

Sorgente `University` (a volte abbreviata in UNI):

```
interface Research_Staff
( source relational University
  extent Research_Staff
  key  (name)
  foreign_key (dept_code)
  references Depatment (dept_code);
  foreign_key (section_code)
  references Section (section_code)
{ attribute string name;
  attribute string e_mail;
  attribute integer dept_code;
  attribute integer section_code; };
```

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

```
interface Department
( source relational University
  extent Departments
  key dept_code )
{ attribute string dept_name;
  attribute integer dept_code;
  attribute integer budget;};
```

```
interface Section
( source relational University
  extent Section
  key section_code
  foreign_key (room_code)
  references Room (room_code))
{ attribute string section_name;
  attribute integer section_code;
  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; };
```

Sorgente `Computer_Science` (a volte abbreviata in CS):

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

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

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

interface Professor : CS_Person
 ( source object Computer_Science
   extent Professors )
  {attribute Office belong_to;
  attribute string rank;};

interface Office
( source object Computer_Science
  extent Offices
  key description )
{ attribute string description;
  attribute Location address;};

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

Sorgente tax_Position_xml

```
interface Student
( source semistructured tax_Position
  extent Students
  key student_code )
{ attribute string name;
  attribute integer student_code;
  attribute string faculty_name;
  attribute integer tax_fee; };

interface ListOfStudent
( source semistructured tax_position_xml
  extent ListOfStudent)
  attribute set <Student> Student;
```

Appendice D

L’architettura CORBA

CORBA [21] (*Common Object Request Broker Architecture*) è un’architettura standard, distribuita e ad oggetti sviluppata dall’Object Management Group (OMG). Dal 1989 l’obiettivo del gruppo OMG è stato la progettazione di una architettura per un *software bus* aperto, chiamato *Object Request Broker* (ORB), sul quale oggetti diversi potessero interagire via rete, indipendentemente dal sistema operativo in cui sono stati implementati. Questo standard permette a più oggetti di invocarne altri senza conoscerne l’esatta locazione o in quale linguaggio sono stati implementati.
Il linguaggio utilizzato per definire un oggetto CORBA è l’IDL (*Interface Definition Language*) mentre gli ORB comunicano attraverso il protocollo standard IIOP (*Internet InterORB Protocol*), definito sempre dall’OMG.

Gli oggetti CORBA si differenziano dagli oggetti creati con altri linguaggi per i seguenti aspetti:

- possono essere localizzati in qualsiasi punto della rete;
- possono interagire con oggetti implementati su piattaforme HW/SW diverse, purché ovviamente siano sempre oggetti CORBA;
- possono essere scritti in qualsiasi linguaggio di programmazione per il quale è stato definito il *mapping* con il linguaggio standard IDL (attualmente i linguaggi utilizzabili includono Java, C++, C, Smalltalk, COBOL e ADA).

Come funziona

Il diagramma di Figura D.1 mostra come un client manda un *messaggio* (inteso come esecuzione di un metodo di un altro oggetto) ad un oggetto CORBA

implementato in un server, chiamato *servant-object*. Un client può essere un qualsiasi programma (anche un oggetto CORBA) che invoca un metodo di un servant-object.
Per invocare il metodo, il client utilizza un *object reference* dell’oggetto CORBA che vuole invocare. Se l’oggetto CORBA è locale, l’object reference è un puntatore ad un oggetto altrimenti, se l’oggetto CORBA è remoto, l’object reference punta ad una *stub function* senza che il client se ne accorga: per il client l’object reference è *sempre* un puntatore ad un oggetto. É l’apparato basato sugli ORB che rende possibile tutto questo.
L’invocazione di un metodo di un oggetto CORBA remoto da parte di un client avviene in questo modo:

1. il client invoca un metodo dell’oggetto CORBA utilizzando l’object reference;
2. la stub function puntata dall’object reference identifica, attraverso l’ORB locale, la macchina sulla quale si trova il servant-object CORBA, che è in attesa di ricevere messaggi;
3. l’ORB locale chiede all’ORB remoto di stabilire una connessione con l’oggetto CORBA;
4. ottenuta la connessione, l’ORB locale manda all’ORB remoto l’object reference della stub function e i parametri per il metodo da invocare;
5. l’ORB remoto passa la richiesta di esecuzione del metodo, assieme ai parametri, al servant-object che eseguirà il metodo invocato;
6. i risultati ed eventuali eccezioni vengono ritornate all’ORB locale lungo lo stesso percorso.

Il client non sa dove si trova il servant-object CORBA, non ne conosce i dettagli implementativi e nemmeno quale ORB è stato usato per stabilire la connessione.

Il Naming Service

Un client può invocare un oggetto CORBA remoto attraverso un object reference: ma come fa ad ottenere l’object reference? L’architettura CORBA mette a disposizione diversi modi per ottenere il reference. Uno di questi (semplice e flessibile) è il *Naming Service*, uno dei servizi standard implementato negli ORB. Il principio su cui si basa è semplice: assegnare un nome ad ogni oggetto CORBA creato e memorizzarlo in un *registro* di nomi.

In particolare, quello che occorre fare è attivare un *naming server* (un'applicazione fornita assieme alle librerie che permettono di creare oggetti CORBA in uno specifico linguaggio) sulla macchina in cui si vogliono creare oggetti CORBA accessibili in remoto: ogni oggetto CORBA creato dovrà poi registrarsi nel naming server, che gestisce il registro degli oggetti CORBA su quella macchina.
I nomi degli oggetti possono essere organizzati in una struttura ad albero proprio come i file sono organizzati in directory. Per accedere ad un determinato oggetto CORBA , il client esegue due sole operazioni:

1. chiedere all'ORB locale di connettersi ad un naming server (naturalmente, il naming server gira su una macchina remota collegata in rete e il client dovrà indicare all'ORB l'indirizzo e la porta per accedere al servizio);

2. ottenuta la connessione, attraverso l'ORB chiedere al naming server un object reference all'oggetto CORBA registrato sotto un certo nome.

Per esempio, nella Figura D.2 è rappresentata la struttura ad albero memorizzata presso un naming server: si notano i *naming context* (equivalenti alle directory per i file system) *Initial Naming Context* (sempre presente) e *Personal*, mentre gli oggetti CORBA registrati sono `plans`, `calendar` e `schedule`. Per accedere all'oggetto CORBA `calendar` il client dovrà prima chiedere al naming server di accedere al naming context *Personal* e poi l'oggetto di nome `calendar`.

La creazione di oggetti CORBA in Java

Per creare un oggetto CORBA occorre definire quali sono le loro interfaccie. Per fare questo si utilizza il linguaggio IDL, **I**nterface **D**efinition **L**anguage. Con un semplice tool messo a disposizione dalla Sun (`idltojava`) le definizioni delle interfaccie IDL vengono tradotte nelle corrispondenti espresse in linguaggio Java, assieme ad una serie di classi che permetteranno l'implementazione dell'oggetto CORBA desiderato in modo semplice. La Figura D.3 mostra la dichiarazione IDL di un oggetto CORBA **Wrapper** e la corrispondente traduzione in Java. Definendo poi una classe Java che implementa l'interfaccia creata si può creare l'oggetto CORBA: naturalmente, occorrerà scrivere il codice dei metodi dichiarati nell'interfaccia.

Occorre sottolineare che gli oggetti CORBA non hanno proprietà *pubbliche* ma solo *private* ed accessibili solo tramite i metodi messi a disposizione dall'interfaccia.

La figura D.4 mostra la connessione CORBA realizzata dal modulo **SIM** per poter interfacciarsi ad ODB-Tools.

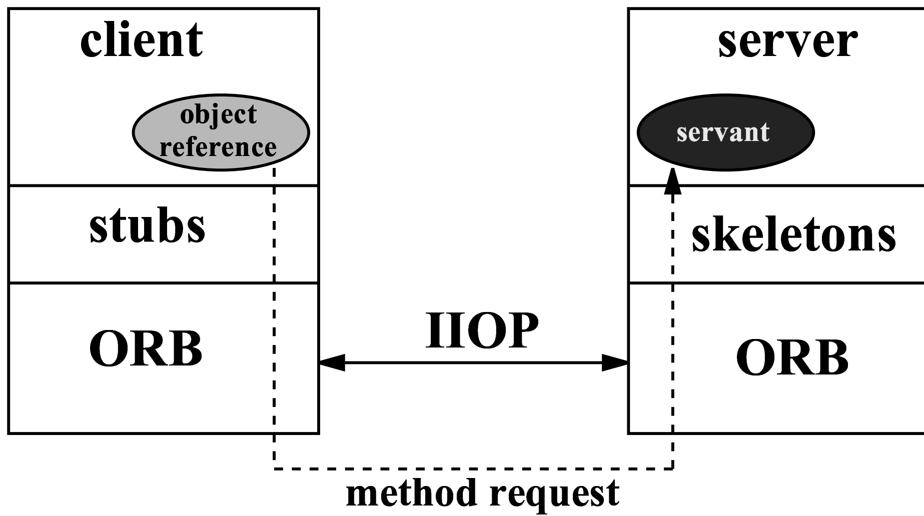

Figura D.1: La invocazione di un metodo di un oggetto CORBA remoto

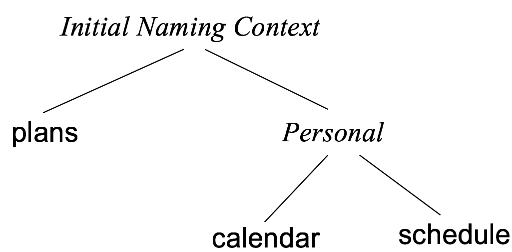

Figura D.2: Esempio di albero creato dal naming server

```
// definizione dell'interfaccia IDL
module MomisApplic {
  interface Wrapper {
     string getType() raises (momisOqlException);
     string getDescription() raises (momisOqlException);
     MomisResultSet runQuery( in string oql ) raises (momisOqlException);
     string getSourceName() raises (momisOqlException);
  };
}

// la stessa interface tradotta in Java
package MomisApplic;
    public interface Wrapper
        extends org.omg.CORBA.Object,
                org.omg.CORBA.portable.IDLEntity {

        String getType()
            throws MomisApplic.momisOqlException;
        String getDescription()
            throws MomisApplic.momisOqlException;
        MomisApplic.MomisResultSet runQuery(String oql)
            throws MomisApplic.momisOqlException;
        String getSourceName()
            throws MomisApplic.momisOqlException;
}
```

Figura D.3: Traduzione in Java di una interfaccia IDL di un oggetto CORBA

```
ORB orb;
org.omg.CORBA.Object objRef;  // Get the root naming context
NamingContext ncRef;
NameComponent nc;
String serverName;
String orbPort;
String orbServerName;
String connArray[];                 //array used to initialize the ORB
OdbToolsFactory factoryObj          //reference all’oggetto factory
OdbTools odbt = null;               //refernce all’oggetto servant

orbServerName = (String)configuration.get("odbt_orbServer");
orbPort       = (String)configuration.get("odbt_orbPort");
serverName    = (String)configuration.get("odbt_namingName");

connArray = new String[4];          //creating the array for ORB initialization
connArray[0] = "-ORBInitialHost";
connArray[1] = orbServerName;
connArray[2] = "-ORBInitialPort";
connArray[3] = orbPort;

orb = ORB.init(connArray, null);      //create and initialize the ORB;
objRef = orb.resolve_initial_references("NameService");   //get the root naming context;
ncRef = NamingContextHelper.narrow(objRef);
nc = new NameComponent(ServerName, "");   // resolve the object reference in naming ;
NameComponent path[] = {nc};              //creating path;

factoryObj = OdbToolsFactoryHelper.narrow(ncref.resolve(path)); //narrowing ODB-Tools object
odbt = factoryObj.newServent("DemoClient");  //get a new servant;
```

Figura D.4: Collegamento all’oggetto ODB-Tools realizzato da SIM

Bibliografia

[1] 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*.

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

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

[4] Simone Montanari. Un approccio intelligente all' integrazione di sorgenti eterogenee di informazione. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1996-1997.

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

[6] S. De Capitani di Vimercati S. Bergamaschi, S. Castano and M. Vincini. Momis: An intelligent system for the integgration of semistructured data, November 1998.

[7] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. Technical report, Bell Laboratories, 1996.

[8] Sap hone page. Available at http://www.sap.com.

[9] Sonia Bergamaschi. Extraction of informations from highly heterogeneous source of textual data. In *First International Workshop CIA-97 COOPERATIVE INFORMATION AGENTS - DAI meets Database Systems, University of Kiel, Kiel, Germany*, 1997.

[10] W.H. Inmon and C. Kelley. Rdb/vms: Developing the data warehouses, 1993.

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

[12] R.G.G. Cattell. *The Object Database Standard: ODMG-93, Release 1.1*. Morgan Kaufmann Publishers, Inc., 1994.

[13] Francesco Venuta. Il componente query manager di momis: esecuzione di interrogazioni, 1999-2000.

[14] J. P. Ballerini, D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. A semantics driven query optimizer for OODBs. In A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, editors, *DL95 - Intern. Workshop on Description Logics*, volume 07.95 of *Dip. di Informatica e Sistemistica - Univ. di Roma "La Sapienza" - Rapp. Tecnico*, pages 59–62, Roma, June 1995.

[15] J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.

[16] J. J. King. *Query optimization by semantic reasoning*. PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.

[17] M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.

[18] L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.

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

[20] 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.

[21] Lesson: Introducing java idl. disponibile all'indirizzo http://web2.java.sun.com/docs/books/tutorial/idl/intro/index.html.