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

Un Approccio Intelligente all'Integrazione di Sorgenti Eterogenee di Informazione

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Simone Montanari

Correlatore
Dott. Silvana Castano

Anno Accademico 1996 - 97

Parole chiave:
Intelligent Information Integration
Integrazione Semantica
Mediatore
Database eterogenei
ODMG-93

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi e la Dottoressa Silvana Castano per l'aiuto fornito alla realizzazione della presente tesi.
Un ringraziamento particolare va inoltre all'Ing. Maurizio Vincini per la preziosa collaborazione e la costante disponibilità.

Indice

Introduzione

A causa dell'esplosione dei dati disponibili (tanto sulla rete Internet, quanto all'interno di un'azienda), ritrovare ed integrare informazioni provenienti da differenti sorgenti è al giorno d'oggi divenuto un problema critico. D'altra parte, 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.

Il reperimento delle informazioni desiderate, distribuite su una molteplicità di sorgenti, richiede generalmente familiarità con il contenuto, le strutture ed i linguaggi di interrogazione propri di queste risorse separate. L'utente deve quindi scomporre la propria interrogazione in una sequenza di sottointerrogazioni dirette alle varie sorgenti di informazione, e deve poi *trattare* ulteriormente i risultati parziali ricevuti, al fine di ottenere una risposta unificata, considerando le possibili trasformazioni che devono subire questi dati, le relazioni che li legano, le proprietà che possono avere in comune, le discrepanze che ancora sussistono tra loro. Avendo a disposizione un numero sempre maggiore di sorgenti, e di dati da manipolare, è quindi molto difficile individuare persone che posseggano tutte le conoscenze necessarie a portare a termine un compito di questo tipo, e un processo di automazione dell'intera fase di reperimento ed integrazione delle informazioni diviene indispensabile.

Obiettivo della presente tesi si è stato quindi progettare e realizzare un sistema mediatore che, sfruttando le informazioni semantiche proprie di ogni schema (col termine *schema* si intende l'insieme di metadati che descrive un deposito di dati), realizzi in modo semi-automatico l'integrazione di informazioni provenienti da sorgenti eterogenee.

Considerata la relativa *giovinezza* di questa area di ricerca, si è pensato fosse utile presentare, all'interno della tesi, una introduzione alle problematiche dell'Integrazione di Informazioni, nonché una breve rassegna delle soluzioni più interessanti già proposte in letteratura.

Elemento caratterizzante del progetto è stato inoltre l'utilizzo di tecniche di Intelligenza Artificiale e di tecniche di Analisi degli Schemi che erano state precedentemente sviluppate e che si è ripreso, e dove necessario ampliato, allo scopo di arrivare alla realizzazione del sistema Mediatore. Il mediatore costituisce un notevole avanzamento rispetto ai sistemi pre-esistenti proposti in letteratura, in quanto include alcune fasi significative del processo di integrazione realizzate in maniera automatica.

Le tecniche utilizzate all'interno del progetto, ma che non sono state sviluppate parallelamente ad esso, vengono comunque descritte nella tesi: in particolare, sono stati utilizzati gli ODB-Tools, un insieme di moduli software basati su tecniche di Intelligenza Artificiale e sviluppati presso l'Università di Modena, e tecniche di Analisi degli Schemi, sviluppate presso il Dipartimento di Scienze dell'Informazione dell'Università di Milano.

A partire da queste, è stato sviluppato il modulo MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources), che porta, avendo a disposizione un insieme di schemi di sorgenti locali, alla definizione di uno schema *globale*, che include tutti i concetti che le diverse fonti di informazioni vogliono condividere, e che sarà l'unico col quale l'utente dovrà interagire. In particolare, per realizzare il mediatore, è stato definito un linguaggio dichiarativo (denominato ODL_{I^3}) che permetta l'inserimento di relazioni terminologiche che legano le diverse sorgenti, nonché la definizione di regole di *mapping* che favoriscano il processo di integrazione stesso. Le tecniche di Analisi degli schemi saranno utilizzate per individuare concetti simili presenti in fonti di informazioni differenti, mentre si farà uso delle tecniche di Intelligenza Artificiale sia per automatizzare alcuni passaggi della fase di integrazione, sia per ottimizzare la fase di trattamento delle interrogazioni dell'utente. È stato realizzato un modulo software, $\textbf{SIM}_1$, ancora in versione prototipale, che realizza le funzionalitá di Integrazione degli schemi.

La struttura della tesi è la segunte.

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

Il Capitolo 2 rappresenta lo stato dell'arte, ed è costituito dalla descrizione dei sistemi di Mediazione più interessanti già presenti in letteratura.

Nel Capitolo 3 viene data una descrizione sia degli ODB-Tools, sia delle tecniche di Analisi degli Schemi, che sono state utilizzate nello sviluppo di MOMIS, ma che comunque non sono state sviluppate parallelamente ad esso. La loro descrizione è riportata anche per mettere in evidenza, successivamente, le modifiche che è stato necessario apportarvi per meglio utilizzarle all'interno del sistema di mediazione.

Il Capitolo 4 riporta la descrizione dell'intero progetto MOMIS, descrivendo, con

l'aiuto di un esempio di riferimento, tutti i passaggi che portano alla unificazione di un insieme di sorgenti eterogenee.
Il Capitolo 5 descrive il modulo software **SIM**$_1$ che è stato realizzato, riportandone sia l'architettura, sia gli algoritmi più significativi.

Parte dei risultati di questa tesi sono stati presentati nell'articolo *"An Intelligent Approach to Information Integration"* [1], già valutato positivamente dalla Comunità Scientifica Internazionale ed accettato per la presentazione al Convegno Formal Ontology in Information Systems FOIS98 (la documentazione ufficiale che ne certifica l'accettazione è riportata nell'Appendice D. Ulteriori avanzamenti del progetto (già comunque inclusi nella presente tesi) sono inoltre stati presentati in [2, 3], entrambi sottomessi per la pubblicazione.

Capitolo 1

L'Integrazione Intelligente di Informazioni

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

consistenza a fronte di cambiamenti nelle sorgenti originali. Con *Integration of Information* invece, come è descritto in [4], si rappresentano in letteratura tutti quei sistemi in grado di combinare tra di loro dati provenienti intere sorgenti o parti selezionate di esse, senza fare uso della replicazione fisica delle informazioni, bensì basandosi sulle loro descrizioni. Quando inoltre questa integrazione utilizza tecniche di intelligenza artificiale, sfruttando le conoscenze acquisite, possiamo parlare di Intelligent Integration of Information (I^3), che si distingue quindi dalle altre forme di integrazione prefiggendosi non una semplice aggregazione di informazioni, bensì anche un aumento del loro valore, ottenendo nuove informazioni dai dati ricevuti.
Con questi obiettivi si è quindi inserita, nell'ambito dell'integrazione, l'Intelligenza Artificiale (IA), che già aveva dato buoni risultati in domini applicativi più limitati. Naturalmente, è ovvio come sia pressocché impossibile pensare ad un sistema che vada bene per tutti i domini applicativi, e che magari integri un numero altissimo di sorgenti. Per questo motivo, per realizzare sistemi molto ampi, è stata proposta una partizione delle risorse e dei servizi che questi sistemi devono supportare, e che si articola su due dimensioni:

1. orizzontalmente, in tre livelli: livello utente, moduli intermedi che fanno uso di tecniche di IA, risorse di dati;

2. verticalmente: molti domini, con un numero limitato (e minore di 10) di sorgenti.

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

- *facilitator* e *mediator* (le differenze tra i due sono flebili ed ancora ambigue in letteratura), che ricercano le fonti "interessanti" e combinano i dati da esse ricevuti;

- *query processor*, che riformulano le query aumentando le probabilità di successo di quest'ultime;
- *data miner*, che analizzano i dati per estrarre informazioni intensionali implicite.

Nell'impostazione del progetto di Integrazione di Sorgenti Eterogenee presentato nella tesi, abbiamo seguito i principi ispiratori citati, sia per la loro completezza, sia per la riconosciuta validità del modello proposto. Oltre alla architettura di riferimento, muovendosi questo progetto in un campo di ricerca particolarmente giovane e in evoluzione, è riportato in appendice il glossario, a cui rifarsi per termini che risultino ambigui o poco chiari, definito nell'Appendice A.

1.1 Architettura di riferimento per sistemi I^3

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

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

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

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

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

- pianificazione e supporto della logistica;
- sistemi informativi nel campo sanitario;
- sistemi informativi nel campo manifatturiero;
- sistemi bancari internazionali;
- ricerche di mercato.

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

- *la grande differenza tra le fonti di informazione*:
  - le fonti informative sono semanticamente differenti, e si possono individuare dei livelli di differenze semantiche [6];
  - le informazioni possono essere memorizzate utilizzando differenti formati, come possono essere file, DB relazionali, DB ad oggetti;
  - possono essere diversi gli schemi, i vocabolari usati, le ontologie su cui questi si basano, anche quando le fonti condividono significative relazioni semantiche;
  - può variare inoltre la natura stessa delle informazioni, includendo testi, immagini, audio, media digitali;

  - infine, può variare il modo in cui si accede a queste sorgenti: interfacce utente, linguaggi di interrogazione, protocolli e meccanismi di transazione;

- *la semantica complessa ed a volte nascosta delle fonti*: molto spesso, la chiave per l'uso delle informazioni di vecchi sistemi sono i programmi applicativi su di essi sviluppati, senza i quali può essere molto difficile dedurre la semantica che si voleva esprimere, specialmente se si ha a che fare con sistemi molto vasti e quasi impossibili da interpretare se visti solo dall'esterno;

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

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

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

Passiamo ora ad analizzare l'architettura vera e propria di un sistema I^3, riportata in Figura 1.1. L'architettura di riferimento dà grande rilevanza ai Servizi di Coordinamento. Questi servizi giocano infatti due ruoli: come prima cosa, possono localizzare altri servizi I^3 e fonti di informazioni che possono essere utilizzati per costruire il sistema stesso; secondariamente, sono responsabili di individuare ed invocare a run-time gli altri servizi necessari a dare risposta ad una specifica richiesta di dati.
Sono comunque in totale cinque le famiglie di servizi che possono essere identificati in questa architettura: importanti sono i due assi della figura, orizzontale e verticale, che sottolineano i differenti compiti dei servizi I^3.
Se percorriamo l'asse verticale, si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i servizi di *wrapping* provvedono ad estrarre

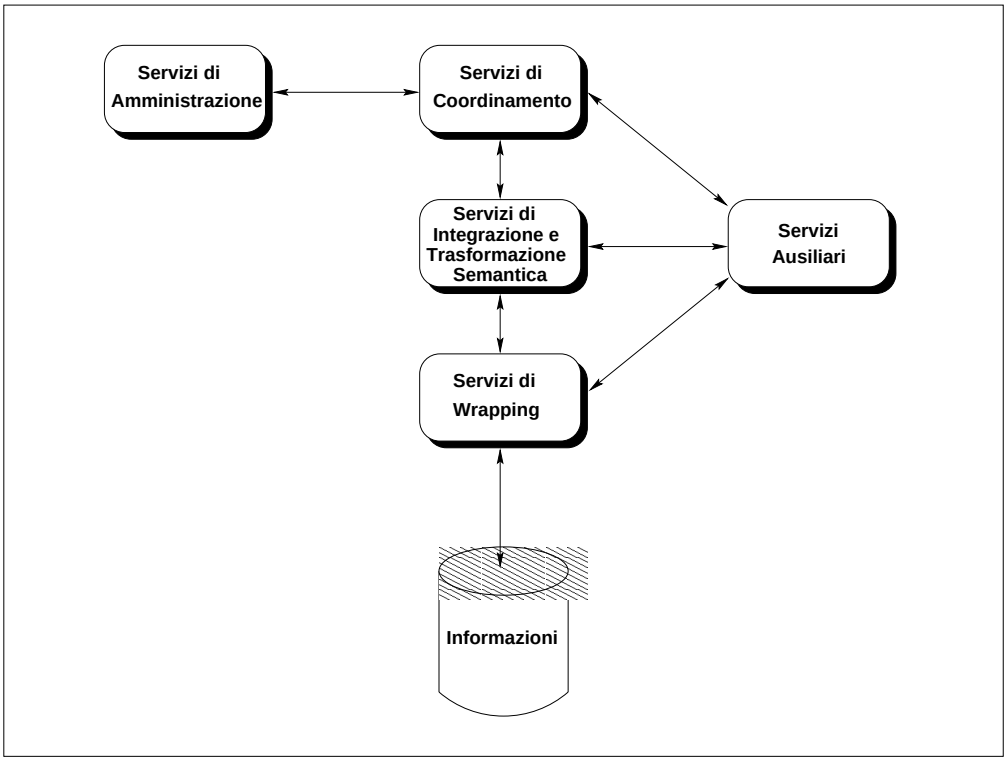


Figura 1.1: Diagramma dei servizi I^3

le informazioni dalle singole sorgenti, che sono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passati ai servizi di Coordinamento che ne avevano fatto richiesta. L'asse orizzontale mette invece in risalto il rapporto tra i servizi di Coordinamento e quelli di Amministrazione, ai quali spetta infatti il compito di mantenere informazioni sulle capacità delle varie sorgenti (che tipo di dati possono fornire ed in quale modo devono essere interrogate). Funzionalità di supporto, che verranno descritte successivamente, sono invece fornite dai Servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti.
Analizziamone in dettaglio funzionalità e problematiche affrontate.

1.1.2 Servizi di Coordinamento

I servizi di Coordinamento sono quei servizi di alto livello che permettono l'individuazione delle sorgenti di dati *interessanti*, ovvero che probabilmente possono dare risposta ad una determinata richiesta dell'utente. A seconda delle possibilità dell'integratore che si vuole realizzare, vanno dalla selezione dinamica delle sor-

genti (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.

1.1.3 Servizi di Amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti *utili*, per determinare le loro capacità, e per creare ed interpretare TEMPLATE. I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task. Sono quindi utilizzati dai sistemi meno "intelligenti", e consentono all'operatore di predefinire le azioni da eseguire a seguito di una determinata richiesta, limitando al minimo le possibilità di decisione del sistema.
In alternativa a questi metodi dei Template, sono utilizzate le **Yellow Pages**: servizi di directory che mantengono le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata). Consultando queste Yellow

Pages, il mediatore sarà in grado di spedire alla giusta sorgente la richiesta di informazioni, ed eventualmente di rimpiazzare questa sorgente con una equivalente nel caso non fosse disponibile. Fanno parte di questa categoria di servizi il Browsing: permette all'utente di "navigare" attraverso le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste. Il servizio si basa sulla premessa che queste descrizioni siano fornite esplicitamente tramite un linguaggio dichiarativo leggibile e comprensibile dall'utente. Potrebbe fornirsi a sua volta dei servizi Trasformazione del Vocabolario e dell'Ontologia, come pure di Integrazione Semantica. Da citare sono pure i servizi di Iterative Query Formulation: aiutano l'utente a rilassare o meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

1.1.4 Servizi di Integrazione e Trasformazione Semantica

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

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

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

3. Servizi di **supporto al processo di integrazione**. Sono utilizzati nel momento in cui una query è scomposta in molte subquery, da inviare

a fonti differenti, ed i loro risultati devono essere integrati. Comprendono inoltre tecniche di *caching*, per supportare la materializzazione delle viste (problematica molto comune nei sistemi che vanno sotto il nome di datawarehouse).

1.1.5 Servizi di Wrapping

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

1. permettere ai servizi di coordinamento e di mediazione di manipolare in modo uniforme il 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.

1.1.6 Servizi Ausiliari

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

1.2 Il mediatore

L'obiettivo del progetto di ricerca di cui la tesi fa parte è la progettazione e realizzazione di un **mediatore**, ovvero del modulo intermedio dell'architettura precedentemente descritta, che si pone tra l'utente e le sorgenti di informazioni. Secondo la definizione proposta da Wiederhold in [7] "un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore... Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti."
Compiti di un mediatore sono allora:

- assicurare un servizio stabile, anche quando cambiano le risorse;
- amministrare e risolvere le eterogeneità delle diverse fonti;
- integrare le informazioni ricavate da più risorse;
- presentare all'utente le informazioni attraverso un modello scelto dall'utente stesso.

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

1. utente: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un'unica risposta, come se stesse interrogando un'unica sorgente di informazioni;
2. mediatore: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio di interrogazione) comune a tutte le fonti;
3. wrapper: ogni wrapper gestisce una singola sorgente, convertendo le richieste del mediatore in una forma comprensibile dalla sorgente, e le informazioni da essa estratte nel modello usato dal mediatore.

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

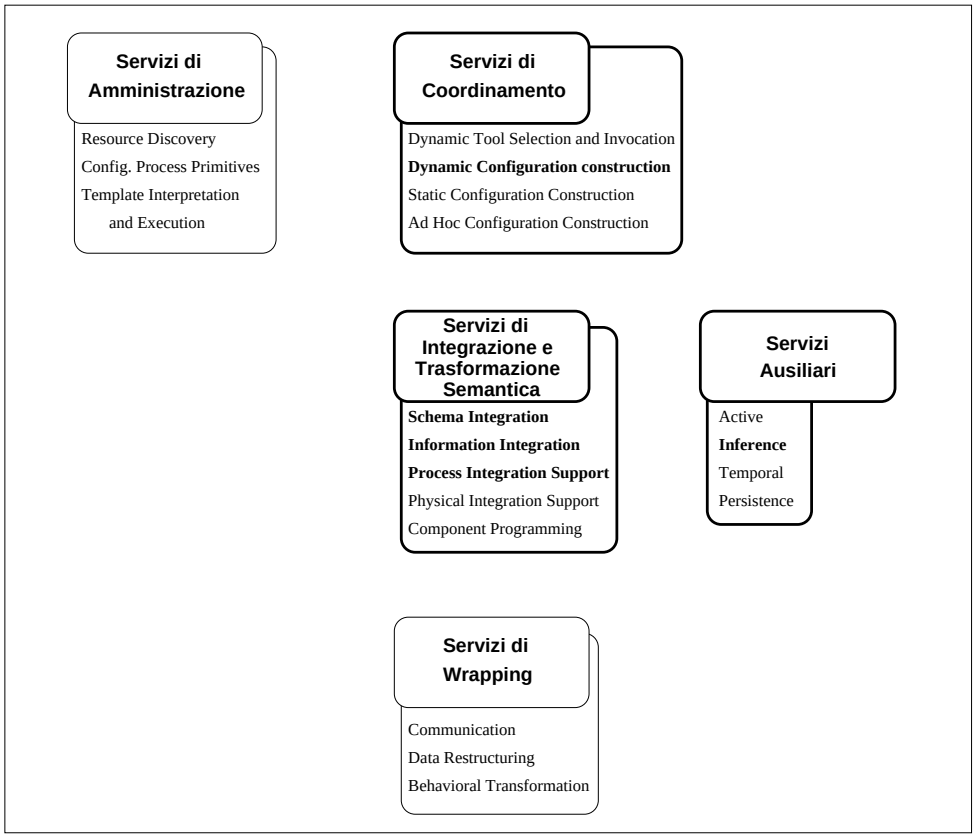


Figura 1.2: Servizi I^3 presenti nel mediatore

- 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, ed eventualmente di scomporre la richiesta in un insieme di sottointerrogazioni diverse da inviare alle differenti fonti di informazione;

- 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 integrazione (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.

Parallelamente a questa impostazione architetturale inoltre, il nostro progetto si vuole distaccare dall'approccio *strutturale*, cioé sintattico, tuttora dominante tra i

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

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

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

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

- il mediatore deve conoscere, per ogni sorgente, lo schema concettuale (metadati);

- informazioni semantiche sono codificate in questi schemi;

- deve essere disponibile un modello comune per descrivere le informazioni da condividere (e dunque per descrivere anche i metadati);

- deve essere possibile una integrazione (parziale o totale) delle sorgenti di dati.

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

1.2.1 Problematiche da affrontare

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

1. problemi ontologici;
2. problemi semantici.

Vediamoli più in dettaglio.

1.2.2 Problemi ontologici

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

1. *top-level ontology*: descrivono concetti molto generali come spazio, tempo, evento, azione..., che sono quindi indipendenti da un particolare problema o dominio: si considera ragionevole, almeno in teoria, che anche comunità separate di utenti condividano la stessa top-level ontology;
2. *domain e task ontology*: descrivono, rispettivamente, il vocabolario relativo a un generico dominio (come può essere un dominio medico, o automobilistico) o a un generico obiettivo (come la diagnostica, o le vendite), dando una specializzazione dei termini introdotti nelle top-level ontology;

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

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

1.2.3 Problemi semantici

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

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

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

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

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

1.3 Soluzione proposta

Tenendo presente tutte le problematiche precedentemente esposte, e un insieme di progetti pre-esistenti che si analizzeranno nel prossimo capitolo, si è giunti alla progettazione di un sistema intelligente di Integrazione delle Informazioni denominato MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources). Contributo innovativo del progetto, rispetto ad altri similari, è sicuramente la fase di analisi ed integrazione degli schemi sorgenti, realizzata in modo semi-automatico, e che rappresenta l'argomento principale di questa tesi. È comunque stato svolto un approfondito lavoro di analisi anche per la fase di *query processing*, ovvero per il processo che dalla query posta sullo schema unificato provvede a generare automaticamente le subquery da inviare alle sorgenti, ed a reintegrare il risultato. Gli strumenti utilizzati, per il componente intelligente di questo mediatore (ed in particolare per lo sviluppo del modulo integratore di schemi), sono stati gli ODB-Tools (basati sulla logica descrittiva OCDL) che si descriveranno nel Capitolo 3. Inoltre, sempre nella fase di integrazione degli schemi, è stata portata avanti una collaborazione con la professoressa Castano del Dipartimento di Scienze dell'Informazione dell'Università di Milano, e con alcuni suoi collaboratori, che già avevano sviluppato un sistema automatico di individuazione di classi affini in schemi diversi. Per quanto riguarda il modulo integratore stesso, è inoltre stato realizzato un prototipo software, denominato $\mathbf{SIM}_1$ (**S**chemata **I**ntegrator **M**odule vers.**1**), che realizza tutti i passaggi che portano alla definizione dello schema unificato.

Capitolo 2

Stato dell'arte

Nonostante l'Integrazione Intelligente di Informazioni sia un campo di ricerca relativamente nuovo, esistono già in letteratura diversi sistemi che cercano di realizzare, in modo più o meno efficiente e flessibile, un modulo integratore (nei casi più riusciti) o un semplice modulo di ricerca di informazioni. Ancora prima di iniziare la fase di analisi del problema, si è ritenuto quindi utile esaminare alcuni di questi sistemi, alla ricerca delle soluzioni migliori. Per non appesantire questo capitolo, e per poter descrivere i sistemi presentati in modo sufficientemente esauriente, si è scelto di limitare a tre il numero di progetti presentati, cercando di selezionare i più significativi. Si rimanda comunque alla bibliografia per una analisi più approfondita sia di questi sistemi, sia di altri [10, 11, 12, 13].

2.1 TSIMMIS

TSIMMIS (**T**he **S**tanford- **I**BM **M**anager of **M**ultiple **I**nformation **S**ources) [14, 15] è sicuramente uno dei progetti più interessanti in questo campo: sviluppato presso l'Università di Stanford in collaborazione con il Centro di ricerca IBM di Almaden, si pone come obiettivo lo sviluppo di strumenti che facilitino la rapida integrazione di sorgenti testuali eterogenee, includendo sia sorgenti di dati strutturati che semi-strutturati. Questo obiettivo è raggiunto attraverso un'architettura comune a molti altri sistemi: i *wrapper* convertono i dati in un modello comune mentre i *mediator* combinano ed integrano i dati ricevuti dai wrapper. I wrapper inoltre forniscono un linguaggio di interrogazione comune per l'estrazione delle informazioni, mostrando un'interfaccia verso l'esterno uguale a quella dei mediatori: in questo modo, l'utente può porre le interrogazioni attraverso un unico linguaggio sia ai mediatori (ricevendo dati integrati da più sorgenti), sia direttamente ai wrapper (interrogando in questo modo un'unica fonte).

In Figura 2.1 è mostrata l'architettura: ad ogni sorgente corrisponde un wrap-

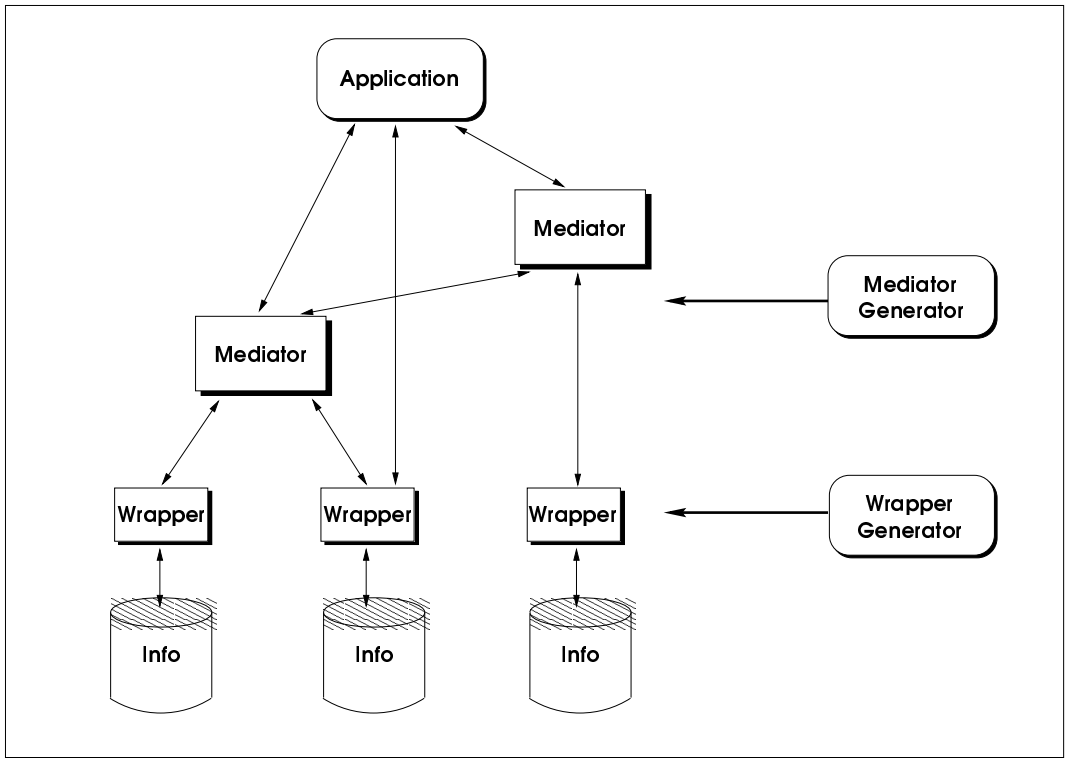


Figura 2.1: Architettura TSIMMIS

per (o *traduttore*) che converte nel modello comune i dati estratti dalla sorgente; sopra i wrapper stanno i mediatori. I wrapper convertono inoltre le query scritte utilizzando il modello comune in richieste comprensibili dalla particolare sorgente da loro servita. Il modello comune, peculiare di questo progetto, è il modello *OEM* (Object Exchange Model) [16], un modello a *etichette* basato sui concetti di identità di oggetto e annidamento. In aggiunta a questo, sono disponibili ed utilizzati dal sistema due linguaggi di interrogazione (*OEM-QL*) e *MSL* (Mediator Specification Language), per ottenere i dati dalle fonti ed integrarli opportunamente. Inoltre, per facilitare il compito dell'amministratore del sistema, sono stati progettati due moduli (*translator generator* e *mediator generator*) che lo aiutino nella realizzazione rispettivamente dei wrapper e dei mediatori, fornendogli un insieme di librerie di funzioni predefinite.

2.1.1 Il modello OEM

Si presenta brevemente il modello OEM, fondamentale per capire il tipo di approccio (*strutturale*) dell'intero progetto TSIMMIS: fa parte dei cosiddetti *self describing model*, dove ad ogni informazione è associata una etichetta che ne descrive il significato, e non si fa uso di un forte sistema dei tipi (in pratica sono ammessi solamente i tipi base). È un modello molto semplificato rispetto ai convenzionali modelli ad oggetti, non supportando direttamente né le classi, né i metodi, né l'ereditarietà, bensì solo l'identità e il nesting tra oggetti. Un esempio di descrizione dell'oggetto **persona** è il seguente:

```
<ob1:  person,  set,          {sub1,sub2,sub3,sub4,sub5}>
       <sub1:   last_name,    str,  'Smith'>
       <sub2:   first_name,   str,  'John'>
       <sub3:   role,         str,  'faculty'>
       <sub4:   department,   str,  'cs'>
       <sub5:   telephone,    str,  '32435465'>
```

Ogni oggetto possiede un identificatore, un'etichetta, un tipo ed un valore. Fondamentale, per la semantica dell'oggetto stesso, è l'etichetta, che ne descrive il ruolo all'interno del contesto in cui è utilizato. Nell'esempio riportato, sono descritti in totale sei oggetti: un oggetto che si potrebbe definire di top-level (**persona**) e cinque sotto-oggetti, che a **persona** sono collegati. È importante sottolineare che, usando l'OEM, non è necessario che oggetti che si descrivono tramite la stessa etichetta abbiano pure lo stesso schema: per esempio, potrebbe esistere un altro oggetto **persona** con un diverso insieme di sotto-oggetti. In questo modo risulta molto semplificata l'integrazione di oggetti provenienti da schemi diversi e semi-strutturati, non dovendo predefinire le strutture che questi oggetti dovranno seguire, ed anzi accettando tutti gli oggetti con una determinata etichetta, qualunque schema seguano.

2.1.2 Il linguaggio MSL

MSL è il linguaggio utilizzato per definire, in modo dichiarativo, i mediatori: attraverso l'uso di *rule*, si può specificare il punto di vista del mediatore, ed in questo modo definire lo "schema globale" in modo manuale. Ogni regola è costituita da una *testa* e da una *coda*, separate dal simbolo :-. La *coda* descrive dove andare a recuperare l'oggetto che si vuole ricevere, mentre l'intestazione definisce la struttura che questo oggetto dovrà avere, una volta estratto e ricostruito. Un esempio di regola MSL sarà presentato nella Sezione 2.1.4

2.1.3 Il generatore di Wrapper

TSIMMIS include un insieme di strumenti, chiamati *OEM Support Libraries*, per realizzare facilmente i wrapper, i mediatori e le interfacce utenti. Questi strumenti comprendono diverse procedure per lo scambio di oggetti OEM in un architettura Client/Server, dove un Client può essere indifferentemente un mediatore, una applicazione o un'interfaccia utente, mentre il server può essere sia un wrapper, sia un mediatore.
In questo modo, si è cercato di semplificare il più possibile la realizzazione dei wrapper, ed in particolare del modulo di Conversione, che ha il compito di convertire una query espressa nel linguaggio MSL in una interrogazione comprensibile dalla sorgente con la quale il wrapper interagisce. Per facilitare l'intero processo, e per poter servire anche sorgenti dalle limitate possibilità di risposta ad interrogazioni, nel progetto TSIMMIS si è ipotizzato di dover esprimere esplicitamente l'intero insieme delle query a cui la sorgente può rispondere (dunque un insieme comunque limitato di query) e di predisporre, attraverso i cosiddetti *query templates*, per ogni query supportabile in MSL, la rispettiva traduzione nel linguaggio di interrogazione proprio della sorgente stessa. Un esempio potrà sicuramente semplificare la descrizione di tutto il processo.

Supponiamo di interagire con una base di dati universitaria, *WHOIS*, che mantiene informazioni sugli studenti e sui professori di una data università, e con possibilità molto limitate: le uniche operazioni consentite sono il ritrovamento dei dati di una persona, conoscendone il nome o il cognome (od entrambi). Le interrogazioni supportate dalla sorgente, espresse nel suo linguaggio, potrebbero essere le seguenti:

1. ritrova le persone dato il cognome: `>lookup -ln 'ss'`
2. ritrova le persone dati cognome e nome: `>lookup -ln 'ss' -fn 'ff'`
3. ritrova tutte le informazioni della sorgente: `>lookup`

Ad ognuna di queste interrogazioni corrisponderà un *query template*: le query in entrata al wrapper saranno scritte in MSL, e dovranno essere tradotte, attraverso questi *query templates*, nel linguaggio locale. Ad esempio, alle interrogazioni 1 e 2 sopra descritte corrisponderanno le seguenti query MSL:

```
(QT2.1) Query ::= *O :- <O person {<last_name $LN>}>
(AC2.1)           {printf (lookup_query, 'lookup -ln %s',$LN);}
(QT2.2) Query ::= *O :- <O person {<last_name $LN>
                                  <first_name $FN>}>
```

```
(AC2.2)          {printf (lookup_query, 'lookup -ln %s -fn %s ',
                          $LN,$FN);}
```

Ad ogni *query template*, è associata una azione, scritta in questo caso in C, che realizza la traduzione da MSL a linguaggio "nativo". Naturalmente, il sistema sarebbe molto limitato se permettesse di rispondere solamente a questo insieme predefinito di interrogazioni: esiste dunque un modulo che permette di definire, data una query in input (che può essere ricevuta da un mediatore, come pure direttamente da una applicazione o da un'interfaccia utente), se ad essa la sorgente sia in grado di dare risposta. In particolare, oltre alle query predefinite, sono supportate tutte le query q tali che:

- q è equivalente ad una query q' direttamente supportata, ovvero gli insiemi delle risposte delle due query coincidono;
- q è sussunta da q' direttamente supportata, ovvero l'insieme risposta di q è incluso nell'insieme risposta di q'.

2.1.4 Il generatore di Mediatori

Il mediatore, in TSIMMIS, è il modulo che si preoccupa di dare una visione integrata dei dati, agendo su differenti sorgenti. Supponiamo di dover integrare due sorgenti, di cui la prima è una base di dati relazionale, *Computer_Science*, il cui schema è riportato di seguito:

```
employee(first_name,last_name,title,report_to)
student(first_name,last_name,year)
```

mentre la seconda è una sorgente ad oggetti, *WHOIS*. Ad ogni sorgente corrisponde, come già mostrato nell'architettura, un wrapper: **CS** esporta le informazioni della prima, **WHOIS** quelle della seconda (esempi di oggetti esportati da questi wrapper sono rispettivamente riportati in Figura 2.2 e in Figura 2.3).

Si vuole sviluppare un modulo mediatore, chiamato **MED**, che integri tutte le informazioni inerenti una persona, ricavate dai due wrapper. Per esempio, supponendo che la persona in questione si chiami 'Chung', ed appartenga al dipartimento 'CS' (ovvero Computer Science), la risposta che si vorrebbe ottenere è rappresentata in Figura 2.4.

Per realizzare questa integrazione, TSIMMIS fa uso di un sistema di regole, MSL (Mediator Specification Language), che permettono di specificare la struttura dell'oggetto integrato, nonché le fonti da cui recuperare i campi della struttura. Le regole devono essere quindi esplicitamente specificate dall'amministratore del sistema, che è l'unico che deve conoscere lo schema di *persona* delle sor-

```
<&e1,  employee,  set,        {&f1,&l1,&t1, &rep1}>
       <&f1,      first_name,  string,   'Joe'>
       <&l1,      last_name,   string,   'Chung'>
       <&t1,      title,       string,   'professor'>
       <&rep1,    reports_to,  string,   'John Hennessy'>

<&e2,  employee,  set,        {&f2,&l2,&t2}>
       <&f2,      first_name,  string,   'John'>
       <&l2,      last_name,   string,   'Hennessy'>
       <&t2,      title,       string,   'chairman'>
..............etc.

<&s3,  student,   set,        {&f3,&l3,&y3}>
       <&f3,      first_name,  string,   'Pierre'>
       <&l3,      last_name,   string,   'Huyn'>
       <&y3,      year,        integer,  3>
```

Figura 2.2: Oggetti esportati da **CS** in OEM

```
 <&p1,  person,   set,       {&n1, &d1, &rel1, &elem1}>
        <&n1,     name,      string,  'Joe Chung'>
        <&d1,     dept,      string,  'cs'>
        <&rel1,   relation,  string,  'employee'>
        <&elem1,  e_mail,    string,  'chung@cs'>
 ..............etc.
```

Figura 2.3: Oggetti esportati da **WHOIS** in OEM

genti e del mediatore. In particolare, riferendoci all'esempio, la rule che realizza il processo specificato sarà:

```
(MS1) Rule:
<cs_person {<name N> <rel R> Rest1 Rest2}>
           :- <person {<name N> <dept 'cs'> <relation R> | Rest1}>
              @whois
              AND decomp(N, LN, FN)
              AND <R {<first_name FN> <last_name LN> | Rest2}>@cs
```

External:

```
decomp(string,string,string)(bound,free,free) impl by name_to_lnfn
decomp(string,string,string)(free,bound,bound) impl by lnfn_to_name.
```

```
<&cp1,  cs_person,  set,         {&mn1, &mrel1, &t1, &rep1, &elm1}>
        <&mn1,      name,        string,  'Joe Chung'>
        <&mrel1,    relation,    string,  'employee'>
        <&t1,       title,       string,  'professor'>
        <&rep1,     reports_to,  string,  'John Hennesy'>
        <&elem1,    e_mail,      string,  'chung@cs'>
```

Figura 2.4: Oggetti esportati da **MED**

La *testa* della regola `MS1` specifica la struttura che avrà l'oggetto unificato: sarà esportato da **MED** con etichetta `cs_person` con un nome (`name`), un ruolo (`relation`), ed un insieme non specificato di altri attributi, ovvero con tutte le altre informazioni che sarà possibile recuperare dalle due sorgenti (rispettivamente `Rest1` e `Rest2`). Nella *coda* sono invece definiti i percorsi dove devono essere recuperate le informazioni: dalla sorgente **WHOIS** si ricavano oggetti di tipo `person` con un nome definito (lo stesso espresso nell'intestazione della rule), un ruolo, e l'attributo dipartimento uguale a 'cs'; dalla sorgente **CS** sono invece recuperati degli oggetti di tipo `R` (dove `R` è il ruolo specificato nell'intestazione, e può valere '`employee`' o '`student`), e di nome e cognome specificato. In ausilio alla rule vi è inoltre una funzione esterna, `decomp`, che realizza la trasformazione da `name` a `first_name` e `last_name` e viceversa.
In tutto il processo non permane alcuna ambiguità: per prima cosa, sono recuperati dalle sorgenti locali gli oggetti la cui struttura (e le cui etichette) sono conformi alla struttura specificata nella *coda* della rule, poi questi oggetti sono unificati e presentati in modo integrato come specificato nella *testa*.

2.1.5 Pregi e difetti

Il punto di forza di questo progetto è sicuramente il modello comune OEM adottato: permette l'integrazione di oggetti di struttura non predefinita (si veda nell'esempio l'uso degli attributi `Rest1` e `Rest2` in Sezione 2.1.4), rendendo possibile ciò che è negato in qualunque ambiente convenzionale orientato agli oggetti, che non aderisca ad una semantica di mondo aperto. Queste possibilità rendono l'intero sistema estremamente flessibile, permettendo l'integrazione di sorgenti il cui schema può essere sia parzialmente sconosciuto, sia variabile nel tempo. Interessante è inoltre il meccanismo di conversione da modello comune a linguaggi di interrogazione locali, attraverso i *query template*, che tiene conto delle diverse possibilità di rispondere ad interrogazioni delle sorgenti (è infatti inverosimile pensare che tutte possiedano le funzionalità di un DBMS). A questo processo può però essere mosso un appunto: esprimendo le capacità di risposta di una fonte di

informazioni attraverso query predefinite, è improbabile che si riescano a rappresentare completamente le funzionalità di questa, essendo frequente il caso in cui la fonte non risponde ad una query q ma è invece in grado di realizzare una query q' che non è sussunta ma sussume q. Basterebbe allora, per realizzare q, avere un modulo successivo di filtro presso il mediatore, ed eseguire una ulteriore selezione sui dati ricevuti in risposta a q'.
Rimane inoltre un problema aperto di fondo: il processo di integrazione di informazione è di per sè un processo ambiguo, in cui non si possono conoscere le sorgenti se non in modo approssimativo, e risulta senza dubbio una forzatura eccessiva l'ipotizzare di lasciare esclusivamente all'amministratore del sistema il compito di individuare i concetti comuni, e di integrarli opportunamente. Questo grosso svantaggio è imputabile anche all'approccio *strutturale* scelto, che non richiede (e quindi non utilizza) gli schemi concettuali delle sorgenti, a cui si contrappone l'approccio *semantico*, nel quale invece sono sfruttate le informazioni semantiche codificate negli schemi al fine di pervenire ad una totale integrazione di tutte le fonti di informazioni.

2.2 GARLIC

L'obiettivo del progetto **GARLIC** [17, 18] (sviluppato presso il centro di ricerca IBM di Almaden) è la costruzione di un Multimedia Information System (MMIS) in grado di integrare dati che risiedono in differenti basi di dati, nonché in un insieme di server di diversa natura. Questa integrazione deve essere realizzata mantenendo l'indipendenza dei data server ed evitando la replicazione fisica dei dati (soluzione tipica dei datawarehouse). *Multimedia* deve in questo contesto essere interpretato in senso molto ampio, indicando non solo immagini, video, e audio ma anche testi e tipi di dati specifici di alcune applicazioni (disegni CAD, mappe,...). Poiché molte di queste informazioni sono già modellate tramite oggetti, GARLIC fornisce un modello orientato ad oggetti che permette a tutte le differenti sorgenti di descriversi in modo uniforme. A questo modello si aggiunge inoltre un linguaggio di interrogazione, anch'esso orientato ad oggetti (ottenuto con un'estensione al linguaggio SQL) e utilizzato dal livello intermedio dell'architettura GARLIC, il cui compito è: presentare lo schema globale alle applicazioni, interpretare le interrogazioni, creare piani di esecuzione e riassemblare i risultati.

La Figura 2.5 rappresenta l'architettura di GARLIC. Alla base della figura stanno le sorgenti di informazioni che devono essere integrate (tra le quali basi di dati relazionali, ad oggetti, file system, document manager, image manager,...). Sopra ogni sorgente è posizionato un wrapper, che trasforma le informazioni sui dati (gli schemi), gli accessi ai dati e le interrogazioni, dal protocollo interno

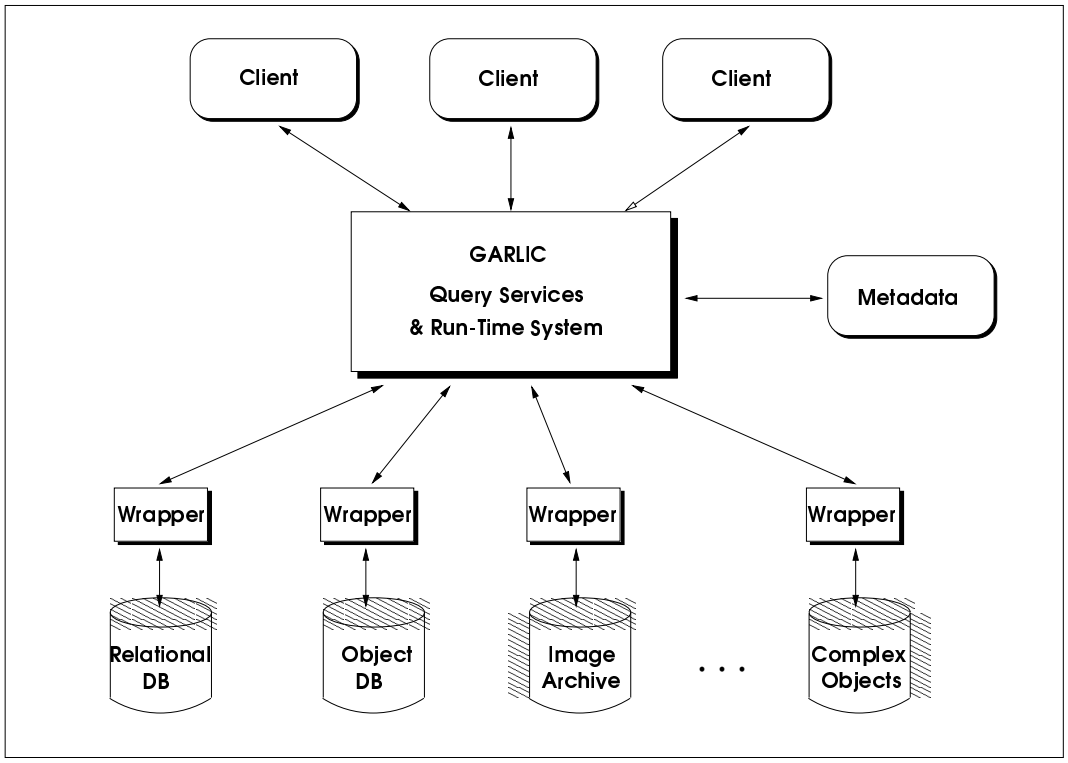


Figura 2.5: Architettura GARLIC

di GARLIC ai protocolli nativi delle sorgenti. Le informazioni riguardanti lo schema unificato sono mantenute nel deposito di metadati. L'altra sorgente di informazioni alla base della figura (Complex objects) serve invece per memorizzare gli *oggetti complessi* di GARLIC, utilizzati dalle applicazioni per unire i dati originariamente separati (siano essi appartenenti a schemi distinti, o allo stesso schema). I servizi di query processing sono forniti dal componente *Query Services & Run-Time System*: ad esso spetta il compito di presentare alle applicazioni una visione unificata, ad oggetti, del contenuto del sistema e di gestirne le richieste (interrogazioni o modifiche). Lo schema globale è presentato all'utente, e alle applicazioni, attraverso il modello di dati di GARLIC (**G**arlic **D**ata **M**odel): è costituito dalla unione degli schemi locali, uno per ogni sorgente, che a loro volta descrivono il contenuto della sorgente. Fra questi è pure disponibile lo schema degli *oggetti complessi*, creati ad-hoc dalle applicazioni per avere una visione integrata di oggetti preesistenti. Il fornire una descrizione dei dati attraverso il GDL (**G**arlic **D**ata **L**anguage) permette inoltre di identificare i dati che si vogliono integrare e parallelamente escludere dalla descrizione quelli che non si vogliono

Relational Repository Schema

```
interface Country {
   attribute string name;
   attribute string airlines_served;
   attribute boolean visa_required;
   attribute Image scene;
   }

interface City {
   attribute string name;
   attribute long population;
   attribute boolean airport;
   attribute Country country;
   attribute Image scene;
   }
```

Web Repository Schema

```
interface Hotel {
   attribute readonly string name;
   attribute readonly short class;
   attribute readonly doube daily_rate;
   attribute readonly string location;
   attribute readonly string city;
   }
```

Image Server Repository Schema

```
interface Image {
   attribute readonly string file_name;
   double matches (in string file_name);
   void display (in string device_name);
   }
```

Figura 2.6: GDL schema

rendere accessibili dall'esterno.

2.2.1 Il linguaggio GDL

Tra i vari compiti dei wrapper vi è quello di fornire una descrizione del contenuto della sorgente da loro servita, utilizzando il **G**arlic **D**ata **L**anguage, o **GDL**. GDL è un variante dell'ODMG **O**bject **D**escription **L**anguage [19]: attraverso le interface, ed un forte sistema dei tipi, si possono descrivere gli oggetti ed il loro comportamento, e memorizzare la loro descrizione in un *repository schema*. I vari *repositories* sono quindi registrati come parti di un GARLIC Database, e *fusi* nello schema globale presentato all'utente. Un esempio di GDL è riportato in Figura 2.6.

L'esempio considera una semplice applicazione per una agenzia di viaggio: l'agenzia gestisce informazioni sugli stati e sulle città per le quali organizza viaggi (in un db relazionale), nonché un sito web per la prenotazione di hotels, ed un image server che raccoglie immagini pubblicitarie. La base di dati relazionali ha due sole tabelle: `Country` e `City`. La tabella `Country` mantiene le informazioni sugli stati, ed ha come chiave l'attributo `name`. La tabella `City` invece possiede sia una chiave, `name`, sia una foreign key, `country`: è interessante vedere come sia compito del wrapper individuare le foreign key nello schema originale (in questo caso `country`), e riportarle in GDL come se fossero attributi con dominio complesso (il domonio di `country` diventa così la tabella `Country`), facendone quindi una traduzione da relazionale a visione orientata agli oggetti (e questo è sicuramente un punto a favore di GARLIC). Più improbabile è invece

la soluzione adottata per l'attributo `Scene`, che viene messo in relazione con la classe `Image`: ipotizzando infatti di sapere a priori che questo attributo si riferisce ad una tabella di un altro sistema, non dovrebbe comunque essere tra i compiti del wrapper il provvedere al *linking* di classi appartenenti a schemi diversi (il wrapper dovrebbe infatti occuparsi, ed essere a conoscenza, solo delle informazioni strettamente relative alla sorgente che gestisce, mentre dovrebbe essere un modulo di livello superiore a provvedere all'integrazione). A supporto di questa visione, viene la soluzione adottata nella descrizione della sorgente Web per le prenotazioni di hotels: l'attributo `city`, che andrebbe logicamente collegato alla tabella `City`, in realtà insiste su un dominio di tipo stringa. Questo perché, nell'esempio, si suppone che il sito web sia al di fuori del diretto controllo dell'agenzia, a differenza della base di dati relazionale e dell'image server. L'eventuale integrazione del sito con le altre sorgenti è quindi (giustamente) demandata alla creazione di un *oggetto complesso*, ad un livello superiore. In particolare, in GARLIC, con *oggetto complesso* si definisce una *vista* il cui obiettivo è arricchire (estendendo, semplificando o deformando) uno o più oggetti appartenenti a schemi locali. Questi oggetti complessi sono definiti in modo dichiarativo (allo stesso modo con cui in SQL è possibile definire una vista basata su altre tabelle) utilizzando il linguaggio di interrogazione interno di GARLIC (si tratta di un'estensione orientata agli oggetti dello **S**tandard **Q**uery **L**anguage) e sono visti dall'utente, e dalle applicazioni, come oggetti veri e propri.
Interessante è anche l'uso della parola chiave `readonly`, che permette di discriminare tra fonti aggiornabili e fonti da cui si può esclusivamente estrarre dati.

2.2.2 Query Planning

La fase di query planning porta, a partire dalla interrogazione posta sullo schema unificato, alla definizione di un insieme di query che le sorgenti locali devono eseguire. Parte attiva in questo processo la hanno i wrapper: le loro conoscenze sono utilizzate per formulare differenti piani di accesso, e per determinare il più efficiente tra questi. Durante la fase di pianificazione della query l'ottimizzatore di GARLIC identifica il frammento maggiore possibile che coinvolge una particolare sorgente, e lo spedisce al corrispondente wrapper: questo determina zero o più piani di accesso che realizzino, in toto o in parte, la query a lui assegnata. A questo punto l'ottimizzatore memorizza tutti i piani di tutti i wrapper interrogati, ed eventualmente aggiunge le operazioni da effettuare nel caso in cui una parte della query originale non sia eseguibile da alcun wrapper. È infatti da sottolineare come GARLIC sia in grado di gestire pure sorgenti con particolari restrizioni: oltre ad ipotizzare che una sorgente non sia in grado di effettuare, ad esempio, join a più vie, il wrapper può essere a conoscenza di particolarissime limitazioni

sulle operazioni realizzabili, come possono essere la lunghezza massima di una stringa, il valore massimo di una costante in una interrogazione,ecc . . . Il vantaggio consiste nel fatto di non dover comunicare tutte le restrizioni all'ottimizzatore, bensì di incapsularle a livello di wrapper. A sua volta l'ottimizzatore, in grado di realizzare funzioni tipiche di un DBMS, potrà effettuare quelle operazioni scartate dalle sorgenti. Oltre a questo, la possibilità di limitare le funzioni di risposta di una sorgente agevola notevolmente la fase di sviluppo di un wrapper: in un primo momento si possono realizzare solo le funzioni più semplici (rendendo utilizzabili da subito le sue informazioni), rimandando ad un secondo momento lo sviluppo di funzioni di ricerca più avanzate.

2.2.3 Pregi e difetti

Nonostante GARLIC sia da considerare la versione commerciale di TSIMMIS (sono entrambi stati sviluppati in ambienti IBM), l'intera impostazione del progetto è stata modificata. La differenza maggiore è senza dubbio l'abbandono del modello OEM (vedi Sezione 2.1.1), a cui è stato preferito l'utilizzo degli schemi locali (descritti attraverso il modello GDL): a fronte di una perdita di flessibilità dell'intero sistema (è ora praticamente impossibile gestire anche sorgenti semistrutturate), l'intera architettura risulta semplificata, così come pure è semplificata la fase di integrazione degli schemi. Purtroppo però non sono stati fatti passi avanti nella automazione della fase di integrazione: l'utente, o l'applicazione, deve definirsi una visione ad-hoc di tutti gli schemi, o deve considerarne semplicemente l'unione (con tutte le duplicazioni di informazioni che ne conseguono). Molto approfondita è invece la fase di query planning, che non si è potuto descrivere meglio in questi paragrafi per motivi di spazio, ma a cui si rimanda negli articoli in bibliografia [17, 18].
Sostanzialmente diverse da progetti analoghi sono comunque le funzioni del wrapper: da semplice traduttore di linguaggi e protocolli, in GARLIC il wrapper include molte delle funzioni demandate in altri sistemi al mediatore vero e proprio. Se dal punto di vista architetturale questo potrebbe anche essere considerato un errore, risulta sorprendente (e forse poco credibile) la sostanziale differenza dei tempi di sviluppo dichiarata da TSIMMIS e da GARLIC: mentre in TSIMMIS si considera ragionevole impiegare circa 6 mesi per realizzare un semplice wrapper, nel progetto GARLIC può essere sviluppato in poche settimane. . .

2.3 SIMS

SIMS [20, 21], sviluppato presso l'Università della Southern California, è un mediatore di informazioni che si occupa di fornire accesso e integrazione ad una

molteplicità di sorgenti eteorgenee. Il cuore del progetto, ed il suo punto di forza, è la sua dichiarata abilità nel ritrovare e trattare le informazioni in modo *intelligente*, ovvero utilizzando tecniche di intelligenza artificiale. Si basa sulle seguente idee di fondo:

- *Rappresentazione e Modellazione della conoscenza*: è usata per descrivere il dominio che accomuna le informazioni, come pure le strutture ed il contenuto delle sorgenti di informazioni stesse. Il modello di ogni sorgente deve indicare il modello dei dati da questa utilizzato, il linguaggio di interrogazione, la dimensione, e deve descrivere il contenuto di tutti i suoi campi usando la terminologia di un predefinito modello comune del dominio (denominato *domain model*, e che costituisce in pratica lo schema globale);

- *Pianificazione e Ricerca*: è utilizzata per costruire una sequenza di query dirette alle singole sorgenti, a partire dalla interrogazione dell'utente;

- *Riformulazione*: dopo aver determinato un insieme di piani di accesso alle sorgenti, SIMS identifica, applicando un modello dei costi ed un algoritmo di ottimizzazione semantica, il più efficiente tra questi. Questa ricerca del piano migliore è aiutata anche dal fatto di avere a disposizione gli schemi descrittivi, semanticamente ricchi, sia delle singole sorgenti, sia del modello globale;

SIMS si distingue quindi dai progetti precedentemente esposti per la sua abilità nell'utilizzare, nella fase di query processing, tecniche di intelligenza artificiale che utilizzano le descrizioni semantiche delle sorgenti. Rimane invece manuale la fase di integrazione delle informazioni (ovvero la costruzione dello schema globale a partire da quelli locali), nonostante sia stata automatizzata, attraverso il modulo LIM, la traduzione di tutti gli schemi locali dal modello originale al modello di conoscenza di SIMS, che si basa sulla logica descrittiva LOOM.

2.3.1 Gli strumenti utilizzati

Per adempiere a tutte le sue funzioni, e per utilizzare tecniche *intelligenti*, SIMS fa uso di una serie di strumenti:

1. **LOOM**: è il sistema di rappresentazione della conoscenza utilizzato per descrivere sia le fonti di informazioni, sia lo schema globale, nonché rappresenta una fonte di informazione esso stesso (costituita dalle risposte alle query già ottenute, e memorizzate, da una determinata applicazione). Si tratta di un linguaggio descrittivo derivato dal KL-ONE (modello sviluppato da Brachman nel 1976 in [22]).

2. **LIM**: è un'interfaccia (**L**OOM **I**nterface **M**odule) per mediare tra LOOM e le basi di dati. In particolare provvede a tradurre le descrizioni degli schemi delle sorgenti in linguaggio LOOM, nonché a tradurre query dirette alle sorgenti da LOOM al linguaggio di interrogazione proprio della singola sorgente. Deve essere sviluppata ad-hoc per ogni interfaccia che andrà a servire.

3. **Prodigy**: i problemi di selezionare le sorgenti interessate da una query, e di ordinare le subquery in cui viene decomposta, possono essere visti come problemi di pianificazione. Il modulo che risolve questi problemi all'interno di SIMS è Prodigy che, partendo da uno stato iniziale ed utilizzando una serie di operatori da applicare alla query, ottiene uno stato finale nel quale l'obiettivo della query è soddisfatto.

2.3.2 Il modello del dominio e delle sorgenti

Poichè SIMS fa uso di un approccio "semantico", è assolutamente necessario che possieda le descrizioni dettagliate di tutte le fonti informative, creandone un *modello*. Per ogni singola sorgente, il *modello* deve contenere le seguenti informazioni:

- descrivere il contenuto informativo della sorgente;
- specificare se si tratta di una sorgente "classica"(nel qual caso si dovrà utilizzare l'interfaccia LIM) o di una sorgente di conoscenza LOOM;
- descrivere le dimensioni della base di dati e delle sue tabelle, nonché la loro locazione, per poter stimare il costo di un determinato piano di accesso;
- definire le chiavi delle tabelle, se esistono.

In aggiunta ai modelli delle sorgenti, viene definito un modello del dominio applicativo, definito *domain model*, che costituirà lo schema globale, l'unico col quale l'utente dovrà interagire. Questo è costituito da una base di conoscenza

```
(retrieve (?depth)
          (:and (port ?port)
                (port.name ?port ‘‘SAN-DIEGO’’)
                (port.depth ?port ?depth)
```

Figura 2.7: Esempio di query SIMS

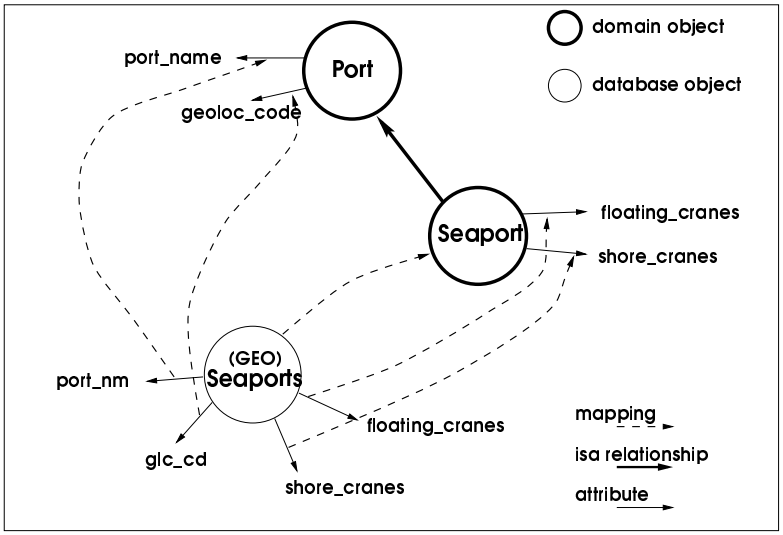


Figura 2.8: Mapping tra *domain model* e modello locale

terminologica organizzata in modo gerarchico (attraverso il linguaggio LOOM), dove i nodi rappresentano tutti gli oggetti, le azioni, gli stati, possibili all'interno del dominio.

Le entità del dominio non devono però necessariamente corrispondere a classi appartenenti ad una determinata sorgente: il *domain model* deve essere inteso come la descrizione del dominio applicativo dal punto di vista dell'utente, e solo con esso l'utente avrà a che fare. In particolare, per porre una query, l'utente compone uno statement LOOM (un esempio di interrogazione è riportato in Figura 2.7, che richiede la profondità del porto di SAN-DIEGO) usando solo la terminologia del *domain model*, essendo in questo modo esonerato dal dover conoscere tutte le sorgenti integrate nel sistema (benché possa pure interagire direttamente con alcune di esse, se ha particolare familiarità con i loro termini). La parte critica dell'intero processo è invece la fase di integrazione delle sorgenti, ovvero il collegare lo schema globale alle varie sorgenti locali. Questo consiste nel descrivere tutti i concetti e le relazioni di una sorgente attraverso i termini del *domain model*. Questo mapping deve essere fatto per tutte le sorgenti, ed in modo particolarmente accurato: un anello di mapping tra un concetto globale ed uno locale significa che i due concetti rappresentano la stessa classe di informazioni (ed analogamente per gli attributi). Un esempio qualitativo di questo mapping è riportato in Figura 2.8: se l'utente richiede tutti gli elementi della classe globale `Seaport`, andrà interrogata la classe `Seaports` della sorgente `GEO`.

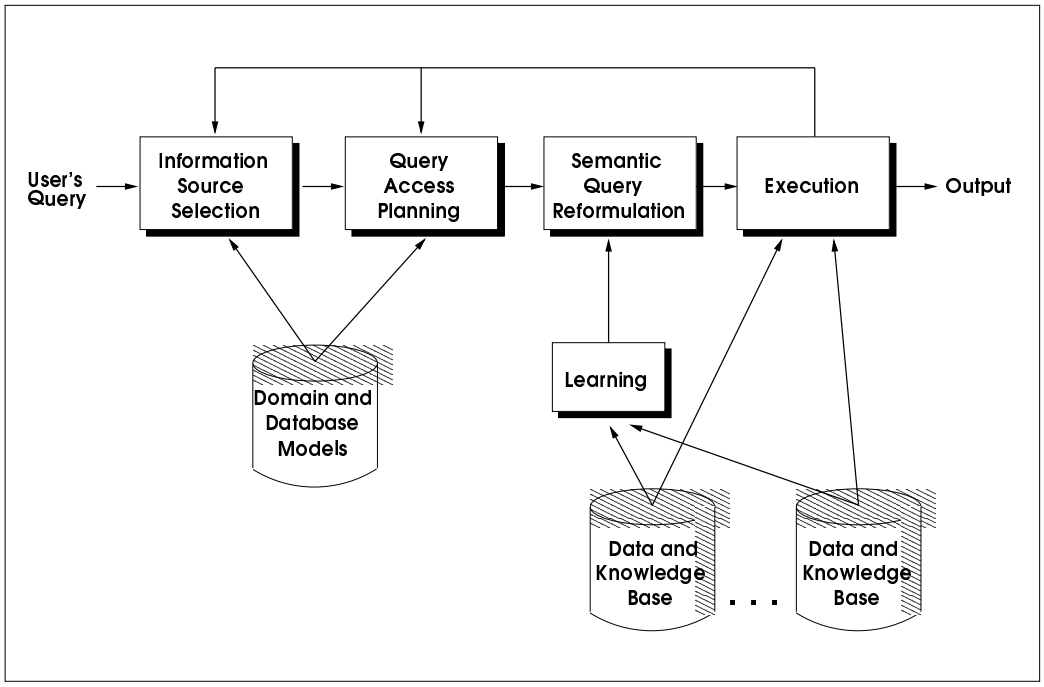


Figura 2.9: Query Processing

2.3.3 Query Processing

L'intero processo che porta dalla formulazione di una query da parte dell'utente, posta sullo schema globale, alla presentazione dei risultati, è rappresentato in Figura 2.9.

Selezione delle fonti informative Il primo passaggio da realizzare, una volta in possesso dell'interrogazione dell'utente formulata usando la terminologia dello schema globale, è identificare le appropriate sorgenti che saranno interessate dalla query. Per esempio, se l'utente richiede informazioni sui `porti` ed esiste in una base di dati un concetto che contiene i porti, il mapping è facilissimo, come pure la riformulazione della query. In realtà, molto spesso non esisterà un mapping diretto e sarà necessario riformulare la query utilizzando termini propri delle sorgenti locali. A questo scopo, vengono utilizzati una serie di operatori di riformulazione, diretti a definire una query equivalente alla originale. Tra di essi, operatori di generalizzazione (che fanno uso delle relazioni tra classe e superclasse e spostano ad un livello superiore dell'albero gerarchico la query, magari aggiungendovi delle limitazioni ai domini degli attributi in modo da ottenere an-

cora una query equivalente), operatori di specializzazione (che attraverso tecniche di intelligenza artificiale cercano di riclassificare la query ad un livello gerarchico inferiore, in modo da determinare l'insieme di classi da interrogare), operatori di partizione.

Piano di accesso Il piano di accesso determina un ordine per le query trovate nella fase precedente, cercando di massimizzare il parallelismo delle operazioni. Per fare questo sono analizzati i costi di accesso alle sorgenti ed i costi di eventuali passaggi aggiuntivi che il sistema dovrà realizzare per produrre i risultati finali. L'ordine di esecuzione è determinato esaminando quali passi del piano di accesso si basano su risultati raccolti in altre sorgenti, facendo uso del modulo Prodigy.

Query Plan Reformulation Oltre ad una ottimizzazione basata sul modello dei costi di accesso, SIMS è pure in grado di realizzare una query reformulation di tipo semantico. L'idea di base è trasformare la query risultante dal piano di accesso in una query semanticamente equivalente che può essere eseguita in maniera più efficiente. Questa ottimizzazione è realizzata sia a livello di singola sorgente, sia a livello globale. Nella singola sorgente, il problema della riformulazione è analogo al problema dell'ottimizzazione semantica di una query all'interno di una base di dati. La riformulazione si basa quindi su un processo di inferenza che utilizza delle informazioni estratte dal database e codificate attraverso l'uso di *rule* e di limitazioni sui range dei domini. Analogamente viene effettuata una ottimizzazione della query globale, in modo da realizzare efficientemente la fase di processing dei risultati parziali ottenuti dalle singole sorgenti.

2.3.4 Pregi e difetti

Non sono pochi gli aspetti interessanti ed innovativi che caratterizzano questo progetto, dovuti fondamentalmente al massiccio uso di tecniche di Intelligenza Artificiale, che permettono di sfruttare tutti i pregi di un approccio semantico. Queste tecniche sono utilizzate sia in fase di identificazione delle sorgenti (in cui, in assenza di mapping diretti, si determinano a run-time le sorgenti che saranno interessate dalla query), sia in fase di ottimizzazione della query stessa.
Altro aspetto da sottolineare, assente in progetti analoghi, è l'utilizzo di una base di dati interna al mediatore stesso sia per riprocessare le risposte delle sorgenti (ma questo è un aspetto già incontrato), sia per memorizzare le risposte già ottenute, utilizzandole successivamente, in parte o in toto, come sorgente aggiuntiva (evitando in questo modo di andare a recuperare dati già presenti in memoria). Rimane invece intentata, come negli altri progetti, la automazione della fase di

integrazione degli schemi locali. Sarà allora su questa linea che si muoverà il progetto sviluppato in questa tesi.

Capitolo 3

Gli strumenti utilizzati

L'obiettivo di questa tesi è arrivare alla definizione di un modulo che, all'interno del mediatore, realizzi in modo semi-automatico la fase di integrazione delle sorgenti di informazione. Per fare questo si è ritenuto utile fare uso di tecniche di Intelligenza Artificiale, basate sulla logica descrittiva **OCDL** [23], nonché di tecniche di integrazione di schemi, che sfruttano un predefinito dizionario semantico.

In particolare, per quanto riguarda l'utilizzo di algoritmi di inferenza, erano a disposizione alcuni moduli preesistenti basati su **OCDL**, che vanno sotto il nome di ODB-Tools, sviluppati presso il Dipartimento di Ingegneria dell'Università di Modena. La teoria sulla quale si basano invece le tecniche di integrazione strutturale degli schemi, così come verrà presentata in questo capitolo (e come è stata esposta in [24]), è stata sviluppata presso l'Università di Milano, con la quale è stata avviata una collaborazione. La descrizione di questi strumenti è stata riportata per sottolineare, nei prossimi capitoli, le estensioni, teoriche e applicative, che è stato necessario apportarvi per arrivare a definire il modulo integratore di schemi semi-automatico $\mathbf{SIM}_1$.

3.1 ODB-Tools

ODB-Tools è un sistema per l'acquisizione e la verifica di consistenza di schemi di basi di dati e per l'ottimizzazione semantica di interrogazioni nelle basi di dati orientate agli oggetti (OODB). È stato sviluppato presso il Dipartimento di Scienze dell'Ingegneria dell'Università di Modena [25, 26]. L'ambiente teorico su cui è basato ODB-Tools include due elementi fondamentali:

1. **OCDL**(**O**bject **C**onstraint **D**escription **L**ogics), proposto come formalismo comune per esprimere descrizioni di classi, vincoli di integrità, ed interrogazioni e dotato di tecniche di inferenza basate sul calcolo della sus-

sunzione introdotte per le *Logiche Descrittive* nell'ambito dell'Intelligenza Artificiale;

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

3.1.1 Aspetti generali

Il formalismo **OCDL** deriva dal precedente **ODL**, proposto in [23], che già estendeva l'espressività di linguaggi di logica descrittiva al fine 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. La caratteristica principale di **ODL** 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). Sulla base di questi tipi di base si possono definire i *tipi valore*, attraverso gli usuali costruttori di tipo definiti nei *CODMs*, quali *tuple*, *insiemi* e *tipi classe*, che denotano insiemi di oggetti con una identità ed un valore associato. Ai tipi può poi 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*). **ODL** 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 [27], **ODL** è stato esteso per permettere la formulazione dichiarativa di un insieme rilevante di vincoli di integrità definiti sulla base di dati. L'estensione di **ODL** con vincoli è stata denominata **OCDL**(**O**bject **C**onstraint **D**escription **L**ogics). 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 sottoforma di regole *if-then* i cui antecedenti e conseguenti sono espressioni di tipo **ODL**. In tale modo, è possibile descrivere correlazioni tra proprietà strutturali della stessa classe, o condizioni sufficienti per il popolamento di sottoclassi di una classe data. In [23] è stato presentato il sistema OCDL-Designer, per l'acquisizione e la validazione di schemi OODB descritti attraverso **OCDL**, che preserva la consistenza della tassonomia di concetti ed effettua inferenze tassonomiche. In particolare, il sistema prevede un algoritmo di *sussunzione* che determina tutte le relazioni di specializzazione tra tipi, e un algoritmo che rileva gli eventuali tipi *inconsistenti*,

cioé tipi necessariamente vuoti.
In [27] l'ambiente teorico sviluppato in [23] è stato esteso per effettuare l'ottimizzaziome semantica delle interrogazioni, dando vita al sistema ODB-QOptimizer. Sono state inoltre sviluppate delle interfacce software per tradurre descrizioni ed interrogazioni espresse rispettivamente nei linguaggi ODL e OQL (proposti dal gruppo di standardizzazione ODMG-93 [19]) in **OCDL**.

La nozione di ottimizzazione semantica di una query è stata introdotta, per le basi di dati relazionali, da King [28, 29] e da Hammer e Zdonik [30]. 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.
Sia il processo di consistenza e classificazione delle classi dello schema, che 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 e 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 [31].
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: dimin-

uendo l'insieme degli oggetti da controllare per dare risposta all'interrogazione, ne viene effettuata una vera ottimizzazione indipendente da qualsiasi modello di costo.

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

Riportiamo in questa sezione la sintassi del linguaggio **OCDL**, nonché una breve descrizione dei concetti di *espansione semantica* e *relazione di sussunzione*.
Nel corso della descrizione, si farà inoltre riferimento ad un esempio esplicativo, di un dominio `Magazzino`, di cui riportiamo la descrizione in sintassi ODMG-93 nella tabella 3.1, e le regole di integrità su di esso definite in tabella 3.2.

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

Schema e istanza del database

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

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

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

Classi dello Schema

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

Tabella 3.1: Schema del dominio `Magazzino` in sintassi ODMG–93

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

Tabella 3.2: Regole di integrità sul dominio `Magazzino`

Gli identificatori di oggetti sono associati a valori tramite una *funzione totale* δ da $\mathcal{O}$ a $\mathcal{V}(\mathcal{O})$; in genere si dice che il valore $\delta(o)$ è lo *stato* dell'oggetto identificato dall'oid o.
Sia $\mathbf{N}$ l'insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che $\mathbf{A}$, $\mathbf{B}$, e $\mathbf{N}$ siano a due a due disgiunti. $\mathbf{N}$ è partizionato in tre insiemi $\mathbf{C}$, $\mathbf{V}$ e $\mathbf{T}$, dove $\mathbf{C}$ consiste di nomi per *tipi-classe base* ($C, C' \ldots$), $\mathbf{V}$ consiste di nomi per *tipi-classe virtuali* ($V, V' \ldots$), e $\mathbf{T}$ consiste di nomi per *tipi-valori* ($t, t', \ldots$). Un *path* p è una sequenza di elementi $p = e_1 \,.\, e_2 \,.\, \ldots \,.\, e_n$, con $e_i \in \mathbf{A} \cup \{\triangle, \forall, \exists\}$. Con ϵ si indica il path vuoto.
$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$[1] indica l'insieme di tutte le *descrizioni di tipo finite* (S, S', $\ldots$), dette brevemente *tipi,* su di un dato $\mathbf{A}, \mathbf{B}, \mathbf{N}$, ottenuto in accordo con la seguente regola sintattica:

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

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

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

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

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

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

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

Tabella 3.3: Schema con Regole di Integrità in linguaggio OCDL

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

Uno *schema con regole* è una coppia $(\sigma, \mathbf{R})$, dove σ è uno schema e $\mathbf{R}$ un insieme di regole. Ad esempio, il dominio Magazzino rappresentato in Tabella 3.1 ed esteso in 3.2 con l'aggiunta di alcune regole di integrità, è descritto in OCDL tramite lo schema con regole mostrato in Tabella 3.3.

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

$$
\mathcal{I}[[a_1\colon S_1, \ldots, a_p\colon S_p]] = \left\{ [a_1\colon v_1, \ldots, a_q\colon v_q] \,\middle|\, \begin{array}{l} p \leq q, v_i \in \mathcal{I}[S_i], 1 \leq i \leq p, \\ v_j \in \mathcal{V}(\mathcal{O}), p{+}1 \leq j \leq q \end{array} \right\}
$$

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

$$\begin{aligned}
\mathcal{I}[\forall\{S\}] &= \left\{\{v_1,\ldots,v_p\}\,\middle|\, v_i \in \mathcal{I}[S], 1 \le i \le p\right\} \\
\mathcal{I}[\exists\{S\}] &= \left\{\{v_1,\ldots,v_p\}\,\middle|\, \exists i, 1 \le i \le p, v_i \in \mathcal{I}[S]\right\} \\
\mathcal{I}[\triangle S] &= \left\{o \in \mathcal{O}\,\middle|\, \delta(o) \in \mathcal{I}[S]\right\} \\
\mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S']
\end{aligned}$$

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

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

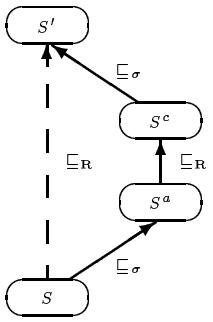
$$\mathcal{I}[(\forall\colon S)] = \mathcal{I}[\forall\{S\}],\ \mathcal{I}[(\exists\colon S)] = \mathcal{I}[\exists\{S\}]$$

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

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

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

Si noti come, in una istanza legale $\mathcal{I}$, l'interpretazione di un nome di classe base *è contenuta* nell'interpretazione della sua descrizione, mentre per un nome di classe virtuale, come per un nome di tipo-valore, l'interpretazione *coincide* con l'interpretazione della sua descrizione. In altri termini, mentre l'interpretazione di una classe base è fornita dall'utente, l'interpretazione di una classe virtuale è calcolata sulla base della sua descrizione. Pertanto, in particolare, le classi virtuali possono essere utilizzate per rappresentare interrogazioni effettuate sulla base di dati. In presenza di classi virtuali cicliche questa computazione non è univoca: fissata un'interpretazione per le classi base, una classe virtuale ciclica può avere più di una interpretazione possibile. Tra tutte queste interpretazioni, noi selezioniamo come istanza legale quella *più grande*, cioè adottiamo una semantica di *massimo punto fisso*. Le definizioni formali di tale semantica e le motivazioni della scelta sono discusse in [23]; in questa sede si vuole soltanto evidenziare il fatto che il modello ammette delle classi virtuali cicliche e quindi il metodo di ottimizzazione proposto è applicabile anche alle query cicliche o ricorsive [32].



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

Figura 3.1: Relazione di sussunzione dovuta alle regole

Sussunzione ed Espansione Semantica di un tipo

Introduciamo la relazione di sussunzione in uno schema con regole.

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

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

È importante notare che la relazione di sussunzione rispetto al solo schema σ, cioè considerando $\mathbf{R} = \emptyset$, denotata con $\sqsubseteq_\sigma$, è simile alle relazioni di *subtyping* o *refinement* tra tipi definite nei *CODMs* [33, 34]. Questa relazione può essere calcolata attraverso una comparazione sintattica sui tipi; per il nostro modello tale l'algoritmo è stato presentato in [23].

È immediato verificare che, per ogni S, S', se $S \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_R S'$. Comunque, il viceversa, in generale, non vale, in quanto le regole stabiliscono delle relazioni di inclusione tra le interpretazioni dei tipi che fanno sorgere *nuove* relazioni di sussunzione. Intuitivamente, come viene mostrato nell'esempio di Figura 3.1, se $S \sqsubseteq_\sigma S^a$ e $S^c \sqsubseteq_\sigma S'$ allora $S \sqsubseteq_R S'$.

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

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

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

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

L'espressione $EXP(S)$ permette di esprimere la relazione esistente tra $\sqsubseteq_{\mathrm{R}}$ e $\sqsubseteq_\sigma$: per ogni $S, S' \in \mathbf{S}$ si ha $S \sqsubseteq_{\mathrm{R}} S'$ se e solo se $EXP(S) \sqsubseteq_\sigma S'$. Questo significa che, dopo aver determinato l'espansione semantica, anche la relazione di sussunzione nello schema con regole può essere calcolata tramite l'algoritmo presentato in [23].

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

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

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

Esempio applicativo

Riprendiamo l'esempio descritto all'inizio di questo capitolo (in Sezione 3.1.2, e vediamo a quali vantaggi pratici può portare l'uso della logica **OCDL**.
L'attività di inferenza iniziale da parte del sistema, effettuauta da OCDL-Designer, consiste nel determinare l'espansione semantica di ogni classe dello schema. Questo permette di rilevare relazioni *isa* non esplicitate nella definizione dell classi stesse. Ad esempio, nel calcolo dell'espansione semantica della classe `DMaterial` si rileva che tale classe è sussunta (implica) l'antecedente della regola R1: congiungendone quindi il conseguente si determina che `DMaterial` è una specializzazione di `SMaterial`. Nel caso in cui vengano applicate più regole durante l'espansione semantica di una classe, le relazioni *isa* ricavate sono meno ovvie di quella appena descritta. Ad esempio, nel calcolo dell'espansione semantica della classe `DStorage` viene applicata prima la regola R1 e successivamente la regola R2, concludendo che `DStorage` è una specializzazione di `SStorage`.

L'altra componente del sistema, ODB-QOptimizer, si occupa invece dell'ottimizzazione semantica delle interrogazioni a run-time, anch'essa basata sull'espansione semantica, in modo da specializzare le classi interessate dall'interrogazione. Prendiamo ad esempio la seguente query, espressa in *OQL*, che vuole

selezionare dallo schema tutti i magazzini che contengono materiali che hanno tutti rischio maggiore o uguale a 15.

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

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

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

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

3.1.3 Architettura degli ODB-Tools

Tutti gli strumenti software che si basano sull'uso della logica descrittiva **OCDL**, e che sono già stati sviluppati presso il Dipartimento di Scienze dell'Ingegneria dell'Università di Modena, sono visibili ed utilizzabili attraverso il sito web *http://www.sparc20.dsi.unimo.it*.
ODB-Tools, la cui architettura è mostrata in Figura 3.2, fornisce i seguenti servizi:

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

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

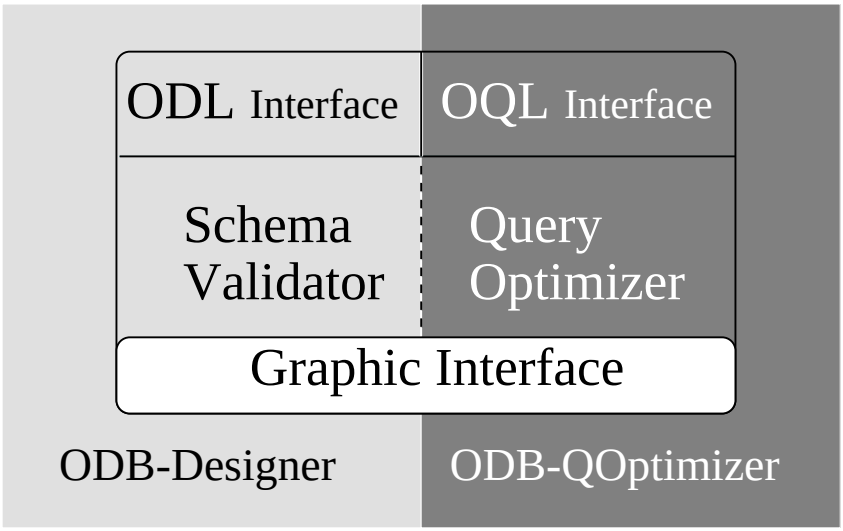


Figura 3.2: ODB-Tools

3.2 Tecniche di Integrazione di Schemi

In questo progetto di tesi è stata iniziata una collaborazione con il Dipartimento di Scienze dell'Informazione dell'Università di Milano, in particolare con la Dottoressa Silvana Castano ed i suoi collaboratori.

In questo paragrafo si darà una breve descrizione delle basi teoriche che abbiamo utilizzato all'interno del progetto mediatore, e che erano state precedentemente sviluppate da questo gruppo. In particolare, si è ritenuto utile fare uso delle loro tecniche di analisi di schemi eterogenei di basi di dati e di identificazione di concetti simili all'interno di questi.

3.2.1 Introduzione

Il problema critico che si deve affrontare nella fase di integrazione di diverse sorgenti di informazione è il superamento delle loro differenze semantiche, dovute principalmente all'uso di diverse terminologie, o di differenti contesti applicativi (dal punto di vista geografico, funzionale,...).

L'idea di fondo è che la presenza di un dizionario semantico all'interno di un dominio che si vuole integrare possa facilitare sia la fase identificazione dei concetti comuni a più schemi, sia la fase di reperimento delle informazioni. Sono quindi state sviluppate tecniche che supportino la definizione e l'organizzazione di questo *dizionario semantico*, attraverso l'analisi degli schemi sorgenti. Il progetto originale è stato sviluppato in collaborazione con l'Information System Author-

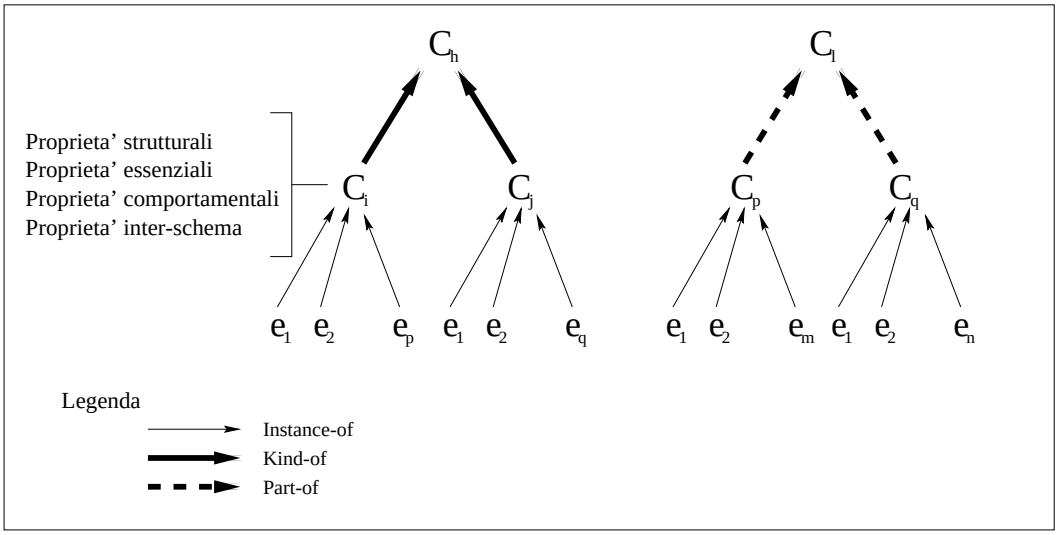


Figura 3.3: Gerarchia di concetti nel dizionario semantico

ity for Public Amministration (AIPA), per riorganizzare e ridisegnare i processi lavorativi della Pubblica Amministrazione.

3.2.2 Architettura del Dizionario Semantico

Nel dizionario semantico sono definite delle *Gerarchie di concetti*, dove i concetti di livello più alto sono o più generali o composti dai concetti di livello inferiore, come è mostrato in Figura 3.3.

Ad ogni concetto C_k appartenente al livello più basso della gerarchia (definito concetto *bottom*) è associato un *cluster* di entità (con termine entità si intendono le tabelle relazionali, essendo questo progetto nato in questo ambito) che sono semanticamente simili ma appartengono a schemi differenti. Nel dizionario semantico, i concetti sono uniti attraverso degli anelli di collegamento, che possono essere di tre tipi:

- **instance-of**: collegamento tra un concetto bottom C_k ed ogni entità e_i presente nel cluster associato con C_k.
- **kind-of**: collegamento tra coppie di concetti bottom utilizzato per rappresentare la relazione *is-a* (ad esempio `Public Organization` **kind-of** `Organization`).
- **part-of**: collegamento che unisce coppie di concetti bottom per rap-

presentare la relazione di *aggregazione* (ad esempio `Unit` **part-of** `Organization`).

In aggiunta a questi collegamenti, per i concetti bottom è mantenuto un insieme di *proprietà*, che ne descrive ulteriori peculiarità. In particolare, devono essere memorizzate le seguenti proprietà:

- **proprietà strutturali**: sono l'unione degli attributi di tutte le istanze e_i appartenenti ad un cluster C_k;
- **proprietà essenziali**: sottoinsieme delle proprietà strutturali, che specifica gli attributi che caratterizzano un cluster C_k e tutte le sue istanze. Possono essere identificate esaminando gli attributi comuni a tutte le entità del cluster;
- **proprietà comportamentali**: sono l'unione delle operazioni che si possono fare su tutte le entità del cluster (in un ambiente ad oggeti, possono essere identificate con i metodi di una classe);
- **proprietà inter-schema**: proprietà che specificano mutue relazioni esistenti tra le entità appartenenti ad uno stesso cluster. In particolare, indicando con $EXT(e_i)$ l'insieme delle istanze (l'estensione) dell'entità e_i, e date due entità e_i ed e_j appartenenti al cluster C_k, possono essere specificate le seguenti proprietà inter-schema:
  1. e_i EQUIVALENT e_j: sse $EXT(e_i) \equiv EXT(e_j)$;
  2. e_i CONTAINED e_j: sse $EXT(e_i) \subset EXT(e_j)$;
  3. e_i OVERLAPPING e_j: sse $EXT(e_i) \cap EXT(e_j) \neq \emptyset$ e e_i CONTAINED e_j non è verificato;
  4. e_i DISJOINTNESS e_j: sse $EXT(e_i) \cap EXT(e_j) = \emptyset$

3.2.3 Costruzione del Dizionario Semantico

Per la costruzione vera e propria del dizionario semantico occorre basarsi sugli schemi concettuali delle sorgenti da integrare, nonché sull'aiuto dell'utente. In particolare, si possono distinguere due fasi, delle quali la prima è ancora principalmente manuale, mentre la seconda è automatizzata:

1. analisi degli schemi: utilizzata per identificare relazioni terminologiche presenti tra le diverse entità. Queste relazioni possono essere derivate dalla struttura delle entità o dal contesto in cui queste si trovano;

2. identificazione delle entità affini: effettuata sfruttando le relazioni terminologiche derivate dalla fase precedente per dare una valutazione del livello di similarità tra le diverse entità. Le entità simili sono poi raggruppate utilizzando tecniche di clustering.

Analisi degli schemi

Il compito della fase di analisi degli schemi sorgenti è l'identificazione delle relazioni terminologiche tra nomi, rilevanti per la valutazione della similiraità tra entità. In particolare, sono considerate:

- relazioni derivate dalla analisi dei nomi assegnati agli elementi degli schemi, come possono essere relazioni di sinonimia;

- relazioni derivate dal contesto delle entità negli schemi relazionali, riferendosi quindi ad attributi, gerarchie di aggregazione e relazioni che caratterizzano una entità: spesso, entità che rappresentano lo stesso concetto in schemi diversi sono caratterizzate dalla presenza di attributi e relazioni simili.

Al fine di rappresentare e codificare queste informazioni, sono state introdotte le seguenti relazioni binarie che coinvolgono coppie di termini:

- SYN (synonyms terms): definita tra nomi che sono considerati sinonimi. Sono termini che possono essere scambiati tra loro in tutti gli schemi senza alcuna modifica nel loro significato, poiché rappresentano lo stesso concetto (ad esempio `Worker` SYN `Employee`). Possono pure rappresentare la relazione tra un termine ed il suo acronimo.

- BT (broader terms): definita per le coppie di nomi in cui il primo ha un significato più generale del secondo. Possono essere dedotte anche dalle gerarchie di specializzazione definite negli schemi concettuali (ad esempio `Organization` BT `Public Organization`).

- RT (related terms): definita tra nomi che sono generalmente usati nello stesso contesto (ad esempio `Organization` RT `Employee`).

Tutte le relazioni presenti nello schema sono quindi memorizzate nel Thesaurus, che diventa in questo modo la base di conoscenza terminologica dell'intero sistema.

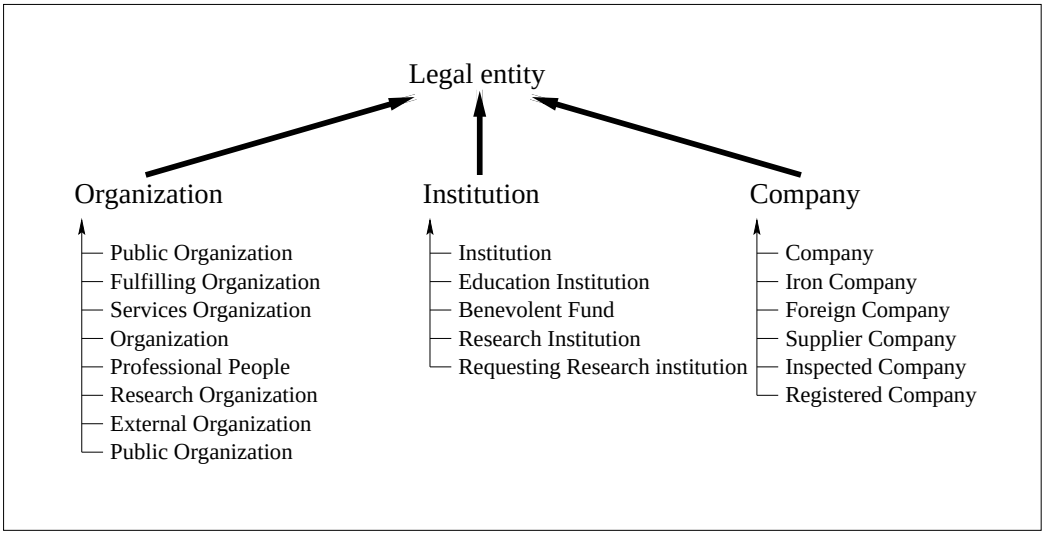


Figura 3.4: Esempio di gerarchia di concetti

Identificazione delle entità affini

In questa fase le relazioni terminologiche tra termini sono utilizzate per determinare il livello di affinità tra le entità. In particolare, per valutare la similarità semantica tra termini, viene assegnato un valore compreso tra zero e uno (definito *forza* e identificato dalla lettera σ) a ciasun tipo di relazione, valore che rispecchia l'importanza ed il peso di quella relazione nel sistema di valutazione introdotto: più sarà stringente la relazione tra due termini diversi, più alto sarà il valore ad essa associato (in particolare, sarà $\sigma_{SYN} \geq \sigma_{BT} \geq \sigma_{RT}$). Attraverso una organizzazione dei termini sul modello delle reti associative [35], sono poi calcolati i livelli di affinità tra tutti i termini appartenenti al Thesaurus, dati dal prodotto delle forze delle relazioni che li uniscono (per una descrizione più dettagliata dell'intero procedimento, si veda la Sezione 4.6 del prossimo capitolo). Vengono in questo modo calcolati (e memorizzati in una matrice) i coefficienti di affinità tra tutte le entità che devono essere classificate, sfruttando sia le affinità tra i nomi di tutte le entità, sia le affinità tra i nomi degli attributi di queste. Sulla base di questa matrice quadrata (dove ogni riga, ed ogni colonna, corrisponde ad una entità, mentre ogni nodo interno corrisponde al livello di affinità tra le entità in questione), attraverso un algoritmo iterativo, vengono riorganizzate in un cluster le entità che hanno tra di loro un livello di affinità più alto. In questo modo, tutte le entità appartenenti ad un singolo cluster faranno verosimilmente riferimento a concetti simili della realtà, di cui il cluster stesso potrà essere considerato il nuovo unico rappresentante. Un esempio dell'intero processo di integrazione è riportato in Figura 3.4: un insieme di entità, estratte da database diversi della Pubblica

Amministrazione, sono stati riorganizzati, dopo aver computato i loro coefficienti di affinità, all'interno di tre cluster, ognuno dei quali è quindi ora rappresentativo di un insieme definito e limitato di entità locali. Da notare che, una volta identificati i cluster, è possibile riorganizzarli in una struttura simile ad un vero e proprio schema concettuale, attraverso la definizione di ulteriori anelli di collegamento tra concetti *bottom* (Sezione 3.2.2), per definire relazioni di specializzazione e di aggregazione, arrivando in questo modo alla creazione di una gerarchia di concetti (in Figura 3.4), i cluster ottenuti sono definiti specializzazioni del concetto `Legal Entity`).

Capitolo 4

Il progetto MOMIS

MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) è un progetto nato parallelamente allo sviluppo di questa tesi, tra il gruppo di lavoro della Professoressa Bergamaschi e lo staff della Dottoressa Silvana Castano (del Dipartimento di Scienze dell'Informazione dell'Università di Milano) con lo scopo di sviluppare un sistema di integrazione di sorgenti eterogenee di informazione. In particolare, con questa tesi, si è approfondito l'ambito in cui questo progetto va a posizionarsi (vedi Capitolo 2), è stata eseguita un'analisi generale dell'intero problema, ed è stata proposta una soluzione (attraverso anche la realizzazione di un modulo software) per la prima parte di questo progetto.

4.1 Il mediatore

Come ampiamente presentato nel capitolo introduttivo, la continua crescita di informazioni accessibili (sia sulla rete Internet, sia all'interno di una azienda) ha portato all'esigenza di costruire sistemi che facilitino il loro reperimento, ovvero che si preoccupino di ritrovare le informazioni (e questo può essere realizzato attraverso i cosiddetti motori di ricerca) ma che diano anche una visione integrata di queste, in modo da eliminare incongruenze e ridondanze necessariamente presenti tra di loro. A questo fine, è stato introdotto da Wiederhold [7] il concetto di *mediatore*, un modulo il cui compito è appunto reperire ed integrare informazioni da una molteplicià di sorgenti eterogenee. I problemi che devono essere affrontati in questo ambito sono principalmente dovuti ad eterogeneità strutturali e realizzative, nonché alla mancanza di una ontologia comune, che porta a differenze semantiche tra le fonti di informazione. A loro volta, queste differenze semantiche possono originare diversi tipi di conflitti, che vanno dalle semplici incongruità nell'uso dei nomi (quando nomi differenti sono utilizzati attraverso le sorgenti per identificare gli stessi concetti) a conflitti strutturali (quando modelli diversi sono

utilizzati per rappresentare le stesse informazioni). Come è già stato descritto nel Capitolo 2, in letteratura sono già stati presentati diversi sistemi diretti all'integrazione di database convenzionali, come pure applicabili all'integrazione di dati semi-strutturati. Di questi, seguendo quanto esposto in [36], si può realizzare una categorizzazione sulla base del diverso tipo di approccio utilizzato, distinguendoli in *semantici* e *strutturali*. Per quanto riguarda l'approccio *strutturale* (e tra questi un esempio è costituito dal progetto TSIMMIS, descritto in Sezione 2.1) si possono sottolineare i seguenti punti caratterizzanti:

- un modello self-describing è utilizzato per trattare tutti i singoli oggetti presente nel sistema, rendendo inutili gli schemi concettuali delle diverse sorgenti;
- le uniche informazioni semantiche sono inserite da un operatore attraverso l'utilizzo di regole (ed in particolare, in TSIMMIS, attraverso le MSL rule);
- l'utilizzo di un linguaggio self-describing facilita l'integrazione anche e soprattutto di dati semi-strutturati;
- l'assenza degli schemi concettuali non permette, in caso di database di grandi dimensioni, una ottimizzazione semantica delle interrogazioni;
- sono eseguibili solo le interrogazioni predefinite dall'operatore (per le quali addirittura è già memorizzato un piano di accesso), limitando in questo modo la libertà dell'utente del sistema.

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

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

Diverse sono le motivazioni che possono portare alla adozione di un approccio di questo tipo, unito all'utilizzo di un modello ad oggetti:

1. la presenza di una schema globale permette all'utente di formulare qualsiasi interrogazione che sia consistente con lo schema, e le informazioni semantiche in esso comprese possono contribuire ad una eventuale ottimizzazione di queste interrogazioni;

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

4. ampi sforzi sono sono già stati realizzati per lo sviluppo di standard rivolti agli oggetti: CORBA [37] per lo scambio di oggetti attraverso sistemi diversi; ODMG-93 [19] (e con esso i modelli ODM e ODL per la descrizione degli schemi, e OQL come linguaggio di interrogazione) per gli object-oriented database;

5. 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 semi-strutturati: 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.

In questa tesi, viene proposto un approccio semantico alla integrazione di informazioni, nel quale sono considerati (ed utilizzati) gli schemi concettuali delle sorgenti che devono essere integrate, e che fa uso di un modello di dati comune e di un linguaggio di definizione comune (ODL_{I^3}) per descriverli. Come verrà più ampiamente descritto nella Sezione 4.4, ODL_{I^3} costituisce un'estensione del corrispondente linguaggio di definizione ODL proposto dal gruppo di standardizzazione ODMG-93. Inoltre, la logica descrittiva **OCDL**(vedi Sezione 3.1.2) è utilizzata come linguaggio *kernel* del sistema, sia per automatizzare la fase di integrazione, sia per la fase do ottimizzazione delle interrogazioni, attraverso l'ausilio degli ODB-Tools.

L'approccio consiste principalmente di due fasi:

1. *unificazione degli schemi*: è la fase più originale del sistema, in rapporto ai sistemi già esistenti, ed è quella approfondita e realizzata in questa tesi. L'uso della logica descrittiva **OCDL**, insieme alle tecniche di clustering

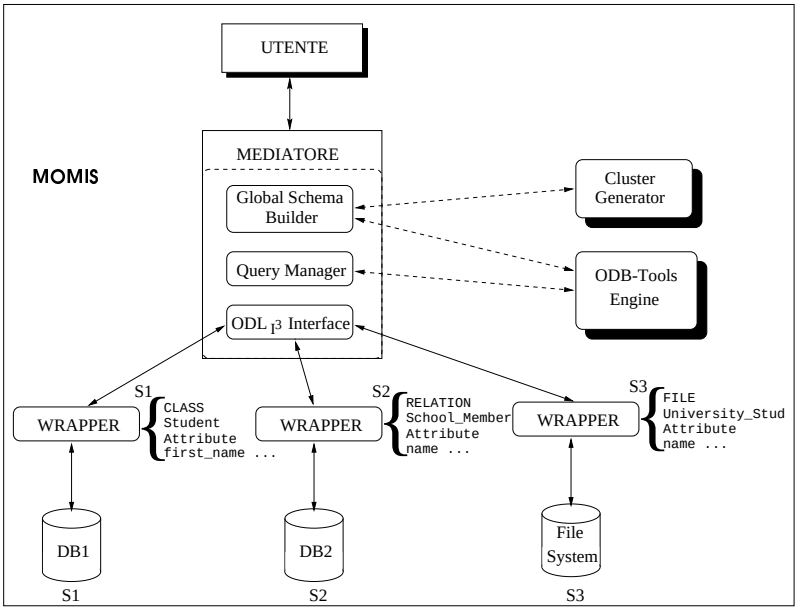


Figura 4.1: Architettura del sistema MOMIS

riportate nella Sezione 3.2, permettono la realizzazione di una fase semi-automatica di integrazione degli schemi, fino a pervenire alla definizione di uno schema globale, direttamente interrogabile dall'utente, che rappresenti l'unione di tutti gli schemi locali, rimuovendone differenze e ripetizioni;

2. *query processing*: è la fase che porta, a partire da una interrogazione dell'utente posta sullo schema globale, alla definizione di un insieme di interrogazioni dirette alle varie sorgenti, nonché alla presentazione di un'unica risposta unificata. In questa tesi è stata realizzata una analisi di questa fase, e sono riportate alcune proposte di soluzione, soprattutto per quanto riguarda la fase di generazione automatica delle interrogazioni locali, mentre rimane ancora aperto il problema della unificazione dei dati da esse estratti.

4.2 L'architettura di MOMIS

In Figura 4.1 è mostrata l'architettura del sistema MOMIS.

È stata mantenuta la struttura classica di un sistema I^3 (si veda ad esempio la corrispondente architettura del progetto TSIMMIS 2.1). I livelli dell'architettura sono quattro:

1. sorgenti: sono le fonti di informazioni che devono essere integrate dal sistema, e sono rappresentate al livello più basso. Possono essere dei database

tradizionali (sia ad oggetti, sia basati sul modello relazionale), oppure dei semplici file system. A differenza di altri progetti analoghi, in questa prima fase del progetto, non si è ipotizzato di avere a che fare con dati di tipo multimediale, mentre è in corso di definizione una estensione al linguaggio di definizione ODL_{I^3} che permetta il supporto anche di dati semi-strutturati [3];

2. wrapper: sono i moduli che rappresentano l'interfaccia tra il mediatore vero e proprio e le sorgenti locali di dati, e devono essere appositamente sviluppati per la sorgente che andranno a servire (o, nel migliore dei casi, si differenzieranno solo per il tipo di sorgente). Ad ogni singola sorgente deve comunque corrispondere un singolo wrapper. La loro funzione è duplice:

   - in fase di integrazione, forniscono, consultando il catalogo della sorgente, la descrizione delle informazioni in essa contenute. Questa descrizione sarà fornita attraverso il linguaggio ODL_{I^3} (descritto in Sezione 4.4);
   - in fase di query processing, devono tradurre la query ricevuta dal mediatore (espressa quindi nel linguaggio comune di interrogazione OQL_{I^3}, che sarà 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;

3. mediatore: è il cuore del sistema, ed è composto da diversi sottomoduli.

   - Global Schema Builder: 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 degli ODB-Tools (sfruttati dal modulo software $\mathbf{SIM}_1$) e delle tecniche di clustering;
   - Query Manager: 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, e a *ricomporre* le informazioni da esse ricevute. Tra i suoi compiti vi è pure l'ottimizzazione semantica delle interrogazioni, realizzata utilizzando gli ODB-Tools.
   - ODL_{I^3} interface: è l'interfaccia che serve al mediatore per interfacciarsi con i wrapper, con i quali comunica soltanto utilizzando il modello

comune. Questa interfaccia permette al mediatore di comunicare con tutte le sorgenti in maniera uniforme, lasciando quindi l'onere delle traduzioni (sia delle descrizioni degli schemi, sia delle interrogazioni, sia dei dati) ai corrispondenti wrapper;

4. utente: è l'utilizzatore del sistema, colui che interagisce con il mediatore. A lui viene presentato lo schema globale, all'interno del quale sono rappresentati tutti i concetti interrogabili: dal suo punto di vista, è come se si trovasse di fronte ad un unico database, da interrogare in modo tradizionale, e le sorgenti locali rimangono completamente trasparenti.

Utilizzando questa architettura, lo scopo che ci si è preposti con MOMIS è la realizzazione di un sistema di mediazione che, a differenza di molti altri progetti analizzati, contribuisca a realizzare, oltre alla fase di query processing, una reale integrazione delle sorgenti. Per fare questo, l'idea da cui si è partiti è stata l'arricchire di una componente *intelligente* le tecniche di integrazione basate sul clustering, ampliandole con fasi automatiche realizzate sfruttando gli ODB-Tools, in modo da diminuire il più possibile l'apporto manuale del progettista del sistema in questa fase. Sono state inoltre rifinite queste tecniche di clustering, al fine di ampliare quella che prima (vedi Sezione 3.2) era una semplice analisi terminologica di nomi di classi e attributi, e che ora è diventata una analisi sia dei nomi che dei domini dei tipi che devono essere integrati.

L'approccio *semantico* utilizzato da MOMIS si articola allora nei seguenti punti:

1. *generazione di un Thesaurus comune*: l'obiettivo di questa fase è la costruzione di un Thesaurus di *relazioni terminologiche*, che si riferiscono a termini di classi appartenenti a sorgenti diverse. Queste relazioni terminologiche sono derivate in maniera semi-automatica, analizzando la struttura e il contesto delle classi degli schemi, attraverso l'interazione del sistema con il progettista, ed utilizzando il modulo $\mathbf{SIM}_1$ (a sua volta basato sugli ODB-Tools);

2. *analisi delle affinità delle classi*: le relazioni terminologiche memorizzate nel Thesaurus sono utilizzate per dare una valutazione delle *affinità* tra le classi di una stessa sorgente, o di sorgenti diverse. Il concetto stesso di affinità è stato introdotto per specificare il tipo di relazione che può intercorrere tra classi diverse dal punto di vista dell'integrazione: la affinità tra due classi è stabilita attraverso appropriati *coefficienti di affinità*, che prendono in considerazione sia i nomi delle classi, sia i loro attributi (analizzando per ogni attributo sia il nome che il dominio);

3. *generazione dei cluster*: sulla base dei coefficienti di affinità calcolati nella fase precedente, i concetti *più affini* tra loro vengono raggruppati all'interno di un *cluster*, attraverso l'uso di un algoritmo iterativo. Lo scopo di questa fase è l'individuazione delle classi che, dopo l'analisi terminologica, risultino rappresentare lo stesso concetto del mondo reale;

4. *generazione dello schema globale del mediatore*: l'organizzazione delle classi all'interno di cluster porta direttamente alla creazione dello schema globale del mediatore. Per ogni cluster ottenuto viene definita una *classe globale*, che sarà rappresentativa di tutte le classi appartenenti al cluster. Lo schema globale sarà in questo modo rappresentato da tutte le classi globali, e sarà la base dell'utente per porre le interrogazioni sui dati delle sorgenti, in modo completamente trasparente.

Una volta ottenuto lo schema globale (operazione da svolgersi solo in fase di iniziazione del sistema, o di acquisizione di una nuova sorgente da integrare), il mediatore è pronto per essere interrogato: l'utente può porre su questo schema globale ottenuto delle interrogazioni, senza sapere in quale sorgente particolare andare a recuperare determinate informazioni. Inoltre, l'utente sarà pure esonerato dal conoscere i diversi linguaggi di interrogazione locali delle fonti integrate: è il compito del Query Manager ottimizzare le query ricevute e spedirle alle sorgenti che da queste devono essere interrogate (durante la fase di Query Reformulation). Nel seguito di questo capitolo saranno approfonditamente analizzati tutti i passi del processo che porta alla costruzione dello schema globale, nonché saranno dati esempi di funzionamento del Query Manager.

4.3 Esempio di riferimento

In questo capitolo verrà utilizzato, per meglio chiarire tutti i passaggi che vengono effettuati sia nella fase di integrazione che in quella di query processing, un esempio di riferimento, riportato in modo schematico in Figura 4.2.

L'esempio si riferisce ad una realtà universitaria: le sorgenti da integrare sono tre. La prima sorgente, `University` (S_1), è un database di tipo relazionale, che contiene informazioni sullo staff e sugli studenti di una determinata università. È composta da cinque tabelle: `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Per ogni professore (presente nella tabella `Research_Staff`), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key `dept_code`), sul suo indirizzo di posta elettronica (`email`), e sul corso da lui tenuto (`section_code`). Per il corso, viene memorizzata pure l'aula (`Room`) dove questo si svolge, mentre del dipartimento sono descritti, oltre al nome (`dept_name`) ed al codice (`dept_code`), il

Sorgente University (S_1)

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

Sorgente Computer_Science (S_2)

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

Sorgente Tax_Position (S_3)

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

Figura 4.2: Esempio di riferimento

budget (`budget`) che ha a disposizione e l'area (`dept_area`) a cui appartiene, sia essa Scientifica, Economica,... Per gli studenti presenti nella tabella `School_Member` sono invece mantenuti il nome (nella coppia `first_name` e `last_name`), la facoltà di appartenenza (`faculty`) e l'anno di corso (`year`). La sorgente `Computer_Science` (S_2) contiene invece informazioni sulle persone afferenti a questa facoltà, ed è un database ad oggetti. Sono presenti sei classi: `CS_Person`, `Professor`, `Student`, `Division`, `Location` e `Course`. I dati mantenuti sono comunque abbastanza simili a quelli della sorgente S_1: per quanto riguarda i professori, sono memorizzati il titolo (`title`), e la divisione di appartenenza (`belongs_to`), che a sua volta fa parte di un dipartimento (e ne può quindi essere considerata una specializzazione); per gli studenti sono memorizzati i corsi seguiti (`takes`) e l'anno di corso (`year`). Il corso ha poi un attributo complesso che lo lega al professore che ne è titolare (`taught_by`), mentre per la divisione si tiene l'indirizzo (`address`), i fondi (`fund`) e il numero di impiegati (`employee_nr`).
È presente inoltre una terza sorgente, `Tax_Position` (S_3), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (`tax_fee`). In questo caso (S_3), non si tratta di un database ma di un file system, che contiene quindi semplici tracciati record.

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

Il linguaggio ODL$_{I^3}$ è il linguaggio di definizione attraverso il quale i wrapper comunicano al mediatore (ed in particolare al Global Schema Builder) le descrizioni delle sorgenti da loro servite. Spunto di partenza per la definizione di questo linguaggio, è stato l'attenersi alle raccomandazioni della proposta di standardizzazione per linguaggi di mediazione [38], risultato del lavoro di un workshop I^3svoltosi presso l'Università del Maryland. Parallelamente a questa proposta (secondo la quale i diversi sistemi di mediazione avrebbero potuto supportare sorgenti con modelli complessi, come quelli ad oggetti, e sorgenti molto più semplici, come file di strutture), si è cercato di discostarsi il meno possibile dal linguaggio ODL, a sua volta proposto dal gruppo ,di standardizzazione ODMG-93.
È stato quindi definito il linguaggio ODL$_{I^3}$ come una estensione allo standard ODL, per raggiungere i seguenti scopi:

- si ipotizzi che il nostro sistema di mediazione debba integrare database relazionali, ad oggetti, e file system;
- tutte le fonti di informazioni devono essere descritte in maniera uniforme, come depositi di oggetti, e quindi utilizzare il concetto di *classe*, indipendentemente dal modello originale utilizzato. Sarà compito del wrapper provvedere alla traduzione dal modello originale di descrizione all'ODL$_{I^3}$, aggiungendo eventualmente in questa descrizione tutte le informazioni necessarie al mediatore (il nome e il tipo della sorgente,. . .);
- il linguaggio supporta la dichiarazione di regole di integrità (*if then rule*), siano esse definite sugli schemi locali (e magari da questi ricevute), siano esse riferite allo schema globale, e quindi inserite dal progettista del mediatore;
- il linguaggio supporta la dichiarazione di regole di mediazione, o *mapping rule*, utilizzate per meglio specificare l'accoppiamento tra i concetti globali e i concetti locali originali;
- è utilizzata una *semantica di mondo aperto*, che permette quindi alle classi descritte di cambiare formato (magari aggiungendo attributi agli oggetti) senza necessariamente cambiarne la descrizione;
- il linguaggio può essere automaticamente tradotto nella logica descrittiva OCDL, e quindi utilizzarne le capacità nei controlli di consistenza nonché nell'ottimizzazione semantica delle interrogazioni.

Utilizzando questo linguaggio, sono fornite al mediatore le descrizioni di tutte le classi da integrare: da sottolineare che le descrizioni ricevute rappresentano

tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema, e quindi interrogabili. Non è quindi detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente, bensì ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa. La sintassi del linguaggio ODL$_{I^3}$è riportata, in BNF, nell'Appendice B. Mostriamo invece di seguito un esempio di descrizioni di classi ODL$_{I^3}$, rimandando invece all'Appendice C per l'intera descrizione dell'esempio di riferimento di Sezione 4.3.

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

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

La nuova definizione aggiunge, dopo il nome della classe, le informazioni sul tipo e sul nome della sorgente di origine, sulla sua estensione, sugli attributi chiave, ed eventualmente sulle foreign key. Nell'esempio sono riportate una classe `Student` derivata da una sorgente ad oggetti (`Computer_Science`) ed una classe `School_Member` proveniente da una sorgente relazionale, rappresentata quindi originariamente da una tabella di tuple, con attributo chiave `name`.

4.5 Generazione del Thesaurus comune

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. Si è inoltre ipotizzato che, per identificare in modo univoco all'interno del mediatore un nome (sia esso di attributo, sia esso di classe), sia rispettivamente necessaria la coppia *nome_sorgente,nome classe* e *nome_sorgente,nome_attributo*. Per semplicità nel nostro esempio, in assenza di ambiguità, sarà usato, per identificare un concetto, solo il nome locale (quando esso è univoco, cioé presente in un'unica sorgente, o quando, pur appartenendo a più sorgenti, denota in tutte lo stesso concetto). Si parlerà inoltre genericamente di *termine* t_i indicando con esso un nome di classe

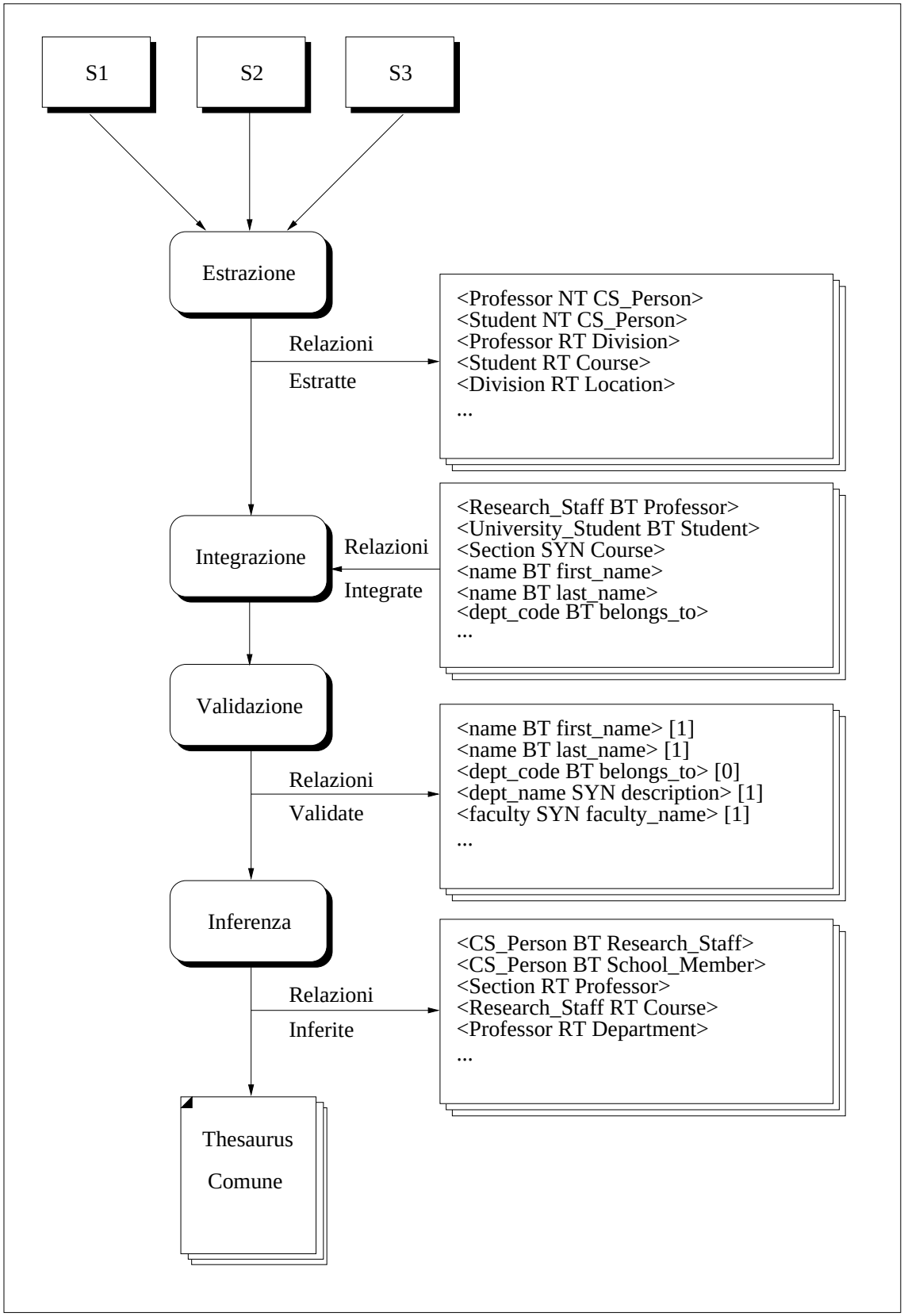


Figura 4.3: Il processo di generazione del Thesaurus comune

o di attributo.

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

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

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

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

- RT (related-term): definita tra due termini t_i e t_j che sono generalmente usati nello stesso contesto, tra i quali esiste comunque un legame non meglio specificato. Per esempio, possiamo avere la seguente $\langle$`Student` RT `Course`$\rangle$.

La scoperta di relazioni terminologiche presenti all'interno degli schemi è un processo semi-automatico, caratterizzato dalla interazione tra il progettista del sistema e gli ODB-Tools. Lo sforzo fatto in questa fase è stato diretto a limitare il più possibile l'intervento dell'operatore, al fine di aumentare la porzione di definizione del Thesaurus realizzabile in modo realmente automatico. L'intero processo che porta, partendo dalle descrizioni degli schemi in ODL$_{I^3}$, alla definizione del un Thesaurus comune è riportato in Figura 4.3, e si articola in quattro passi.

Estrazione automatica di relazioni dagli schemi ODL$_{I^3}$ **(step1)** Sfruttando le informazioni semantiche presenti negli schemi strutturati (sia basati sui modelli ad oggetti, sia relazionali) può essere identificato in modo automatico un insieme di relazioni terminologiche. In particolare il modulo **SIM**$_1$, durante questa preliminare fase di analisi, è in grado di estrarre:

1. **schemi ad oggetti**:

- relazioni BT e NT derivate dalle gerarchie di generalizzazione;
- relazioni RT derivate dalle gerarchie di aggregazione;

2. **schemi relazionali**:

- relazioni RT derivate dalle foreign key.

Esempio 1 Si considerino le sorgenti S_1 (relazionale) e S_2 (ad oggetti) riportate in Figura 4.2. Le relazioni terminologiche automaticamente estratte sono le seguenti:

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

Integrazione delle relazioni (step2) Interagendo con il modulo, il progettista deve inserire tutte le relazioni terminologiche che non sono state estratte nel passo precedente, ma che devono comunque essere presenti per pervenire ad una esatta integrazione delle sorgenti. In particolare, possono essere inserite relazioni che coinvolgono classi appartenenti a schemi diversi, come pure relazioni che si riferiscono a nomi di attributi. Da sottolineare che, durante questa fase, tutte le relazioni inserite dal progettista hanno carattere esclusivamente *terminologico*, a priori quindi da qualunque considerazione sulle estensioni. È quindi una fase che, nei prossimi sviluppi del progetto, potrebbe facilmente essere coadiuvata dall'uso di un dizionario in linea che evidenzi eventuali termini sinonimi, o correlati tra loro.

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

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

Validazione delle relazioni (step3) Come è stato precedentemente sottolineato, le relazioni inserite del progettista si basano esclusivamente sull'analisi dei termini che identificano un concetto. È stata quindi inserita a questo livello una fase di *validazione* delle relazioni definite tra attributi, che verifichi se anche i domini di questi attributi possano essere compatibili con il tipo di relazione definita tra loro. In particolare il modulo **SIM**$_1$, utilizzando gli ODB-Tools, eseguirà i seguenti controlli: siano $a_t = \langle n_t, d_t \rangle$ e $a_q = \langle n_q, d_q \rangle$ i due attributi coinvolti in una relazione.

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

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

Esempio 3 Facendo sempre riferimento all'esempio universitario, si riporta di seguito l'output di questa fase: ad ogni relazione che coinvolga due attributi, definita dal progettista, viene affiancato un flag di controllo che riporta l'esito della validazione.

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

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

- classi: sono modificate in modo da conformarsi alle informazioni aggiunte attraverso le relazioni. Sono quindi aggiunte relazioni di ereditarietà negli schemi (per rappresentare BT o NT non presenti originariamente), relazioni di aggregazione (per rappresentare RT tra classi);

- attributi: tutti gli attributi coinvolti in relazioni validate (che cioé abbiano superato positivamente la fase di controllo dei domini) sono riorganizzati in gerarchie (strutturate sul modello di alberi gerarchici). Per ogni attributo sono mantenuti gli attributi equivalenti (identificati da nomi definiti sinonimi), quelli più specializzati e quelli più generali. Al termine di questa riorganizzazione in gerarchie di termini, ogni attributo è sostituito con il termine più generale della gerarchia di appartenenza.

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

Esempio 4 In Figura 4.4 sono riportate tutte le nuove relazioni inferite, messe a confronto con quelle definite esplicitamente negli schemi (includendo in quelle definite esplicitamente sia quelle inserite manualmente dal progettista, sia quelle dedotte automaticamente durante la fase preliminare di analisi). In Figura 4.5 sono invece riportate tutte le classi di tutte le sorgenti, così come risultano riorganizzate alla fine della fase di generazione del Thesaurus comune, facendo quindi

Relazioni Esplicite (Step 1,2)

⟨CS_Person BT Student⟩
⟨CS_Person BT Professor⟩
⟨School_Member BT Student⟩
⟨Research_Staff BT Professor⟩
⟨Section SYN Course⟩
⟨Department BT Division⟩
⟨Student RT Course⟩
⟨Course RT Professor⟩
⟨Research_Staff RT Department⟩
⟨Research_Staff RT Section⟩
⟨Professor RT Division⟩
⟨Division RT Location⟩
⟨University_Student BT Student⟩
⟨Section RT Room⟩

Relazioni Inferite (Step 4)

⟨CS_Person BT Research_Staff⟩
⟨CS_Person BT School_Member⟩
⟨Section RT Professor⟩
⟨Research_Staff RT Course⟩
⟨Professor RT Department⟩
⟨Professor RT Section⟩
⟨Course RT Room⟩
⟨Student RT Section⟩
⟨CS_Person BT University_Student⟩

Figura 4.4: Relazioni Esplicite ed Inferite

riferimento alle classi modificate. È quindi una rappresentazione grafica delle relazioni che sussistono tra queste: le linee tratteggiate mettono in evidenza le relazioni inferite, le linee unite rappresentano invece quelle esplicite; le linee dotate di frecce rappresentano inoltre relazioni di generalizzazione, mentre quelle in cui le frecce sono assenti denotano le relazioni di aggregazione.

4.6 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

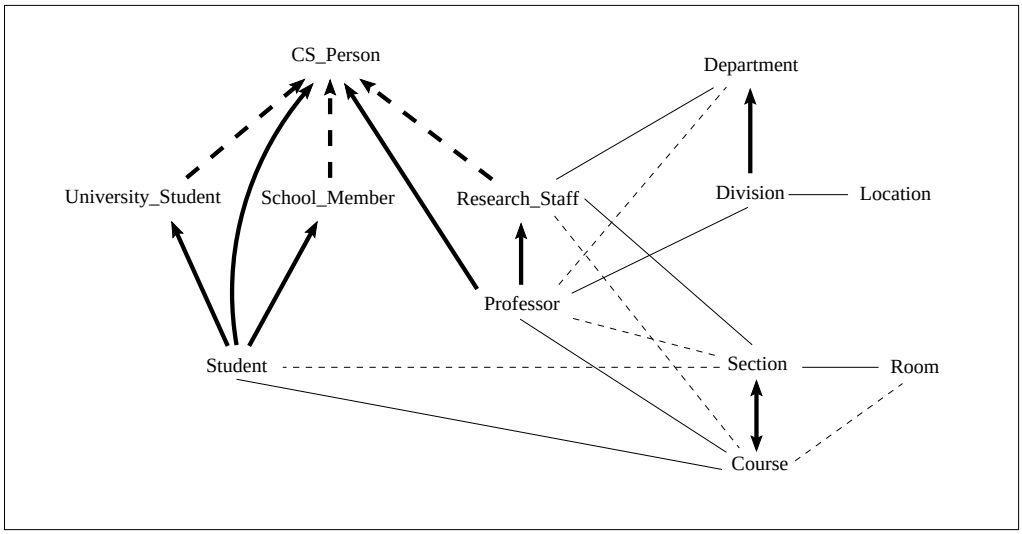


Figura 4.5: Thesaurus comune per le sorgenti $\mathbf{S_1}$, $\mathbf{S_2}$, e $\mathbf{S_3}$.

una struttura simile alle Associative Networks [35], 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 4 (Funzione di Affinità) Presi due termini, t_i e t_j, possono essere presenti nel Thesaurus zero o più cammini che li uniscono, formati da relazioni. Ad ognuno di questi cammini corrisponde naturalmente un valore, dato dal prodotto dei pesi delle relazioni in esso coinvolte. La Funzione di Affinità $A_{thes}(t_i, t_j)$ tra due termini, t_i e t_j, restituisce il valore maggiore tra questi, corrispondente al cammino più *stringente*, che unisce questi termini (che non sempre coincide col cammino più breve), definito come segue:

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

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

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

Esempio 5 Si consideri il Thesaurus illustrato in Figura 4.5. Esiste un cammino tra le classi `University_Student` e `School_Member`, ovvero, `University_Student` $\rightarrow^{nt}$ `CS_Person` $\rightarrow^{bt}$ `School_Member`. Da questo si deduce che $A_{thes}(\texttt{University\_Student}, \texttt{School\_Member}) = 0.8 \cdot 0.8 = 0.64$.

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

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

Per esempio, si supponga $\alpha = 0.4$.
In questo caso, dal momento che $A_{thes}(\texttt{UniversityStudent}, \texttt{School\_Member}) = 0.64$, possiamo concludere che `University_Student` $\sim$ `School_Member`.

4.6.1 Coefficienti di Affinità

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

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

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

Esempio 6 Si considerino le classi `University_Student` e `School_Member`. Si avrà che $NA(\texttt{University\_Student}, \texttt{School\_Member}) = 0.64$

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

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

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

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

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.
Rispetto alla definizione originaria di SA precedentemente sviluppata (si veda [24]), è stata proposta in questa tesi l'estensione realizzata dal fattore di controllo F_c. Il termine F_c realizza un controllo sui domini degli attributi coinvolti nella relazione da esaminare (il controllo è quello esposto nel terzo passo della Sezione 4.5), 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*.

Esempio 7 Si considerino gli schemi delle sorgenti riportati in Figura 4.2. Per quanto riguarda le classi `University_Student` in S_3 e `School_Member` in S_1, abbiamo che $SA(\texttt{University\_Student}, \texttt{School\_Member}) = \frac{2 \cdot 3}{4+4} \cdot \frac{1}{1} = 0.75$.

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

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

dove i pesi w_{NA} e w_{SA}, con $w_{NA}, w_{SA} \in [0, 1]$ e $w_{NA} + w_{SA} = 1$, sono stati introdotti per dare al progettista la possibilità di variare caso per caso l'importanza dovuta ad ognuno dei due coefficienti rispetto all'altro. Nel nostro esempio di riferimento, abbiamo considerato ugualmente rilevanti ai fini dell'integrazione i due coefficienti, ponendo quindi $w_{NA} = w_{SA} = 0.5$.
Durante il calcolo di questo coefficiente globale, è comunque data implicitamente una maggiore rilevanza al *Name Affinity coefficient*, e quindi ai nomi delle classi stesse: per classi i cui nomi non hanno nulla in comune non è neppure valutata la affinità rispetto ai loro attributi, e conseguentemente il loro GA risulterà nullo.

Esempio 8 Partendo dai coefficienti riportati negli Esempi 6 e 7, il Global Affinity Coefficient delle classi `University_Student` e `School_Member` è calcolato nel seguente modo:

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

4.7 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 4.6) 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.

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

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

end procedure.

Figura 4.6: Procedura di clustering

M è così aggiornata dopo ogni operazione di fusione tra cluster, cancellando le righe e le colonne corrispondenti ai cluster unificati, e inserendo una nuova riga ed una nuova colonna che rappresenti il nuovo cluster determinato. Vengono quindi calcolati i coefficienti $GA()$ tra questo cluster aggiunto e tutti quelli già presenti nella matrice: in particolare, viene mantenuto il valore $GA()$ massimo tra i due che erano stati già calcolati tra i cluster rimossi ed il corrispondente cluster col quale si vuole determinare il nuovo valore del coefficiente globale. La procedura termina quando tutte le classi appartengono ad un unico cluster.

L'output di questa fase non è comunque il cluster finale, contenente tutte le classi: ben più importante è l'albero che si è definito attraverso questa procedura di clustering, riportato, sempre relativamente all'esempio universitario, in Figura 4.7. In questo albero, i nodi sono rappresentanti di tutte le classi che sono state analizzate: nodi contigui sono caratterizzati da alta affinità, nodi tra loro molto lontani rappresenteranno invece concetti differenti. In questo modo, scegliendo un valore di soglia di riferimento, si possono formare non un unico bensì un insieme di cluster, all'interno dei quali sono raggruppate tutte le classi (alla prima iterazione) o i cluster (nelle iterazioni successive) tra i quali esiste una affinità (rappresentata dal valore di GA) maggiore del valore soglia predefinito. Come mostrato in figura, abbiamo scelto come soglia il valore 0.5: tutte le classi di partenza delle sorgenti locali S_1, S_2 e S_3 sono state raggruppate iterativamente (attraverso la procedura esposta poco sopra) in cluster finali Cl_i.

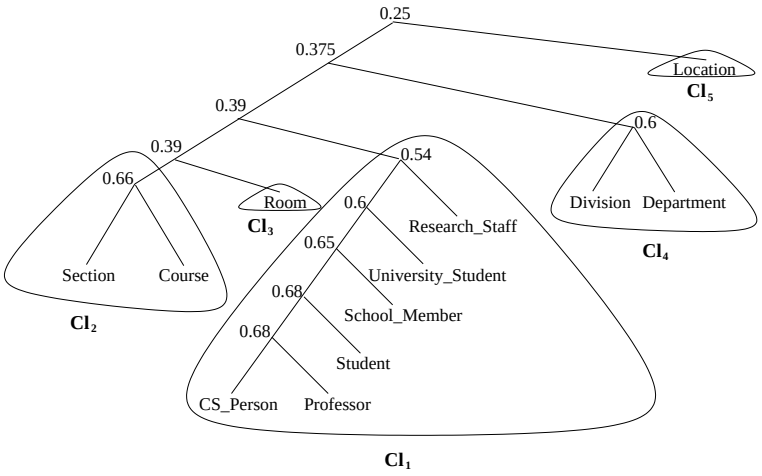


Figura 4.7: Albero di affinità

4.8 Generazione dello Schema Globale

In questa sezione viene presentato il processo che porta, a partire dai cluster precedentemente determinati, alla definizione dello *Schema Globale* del Mediatore, ovvero della visione dei dati che sarà presentata all'utente in fase di Query Processing.

La prima fase di questo processo viene realizzata automaticamente e genera, per ogni cluster, una *classe_globale*$_i$ rappresentativa di tutte le classi che fanno parte del cluster (ovvero una classe che costituisca una visione unificata di queste classi). Sia Cl_i un cluster determinato nella fase precedente: ad esso viene associata la *classe_globale*$_i$, alle quale corrisponderà quindi un insieme di attributi globali. La fase di determinazione degli attributi è realizzabile in modo automatico, basandosi sulle relazioni tra attributi memorizzate nel Thesaurus (vedi Sezione 4.5 step3) e seguendo i seguenti criteri:

- ad ogni *classe_globale*$_i$ è associata l'unione degli attributi di tutte le classi appartenenti al cluster Cl_i dal quale è stata generata;
- all'interno dell'unione degli attributi sono identificati tutti gli insiemi di termini definiti sinonimi, e ne viene riportato solo uno tra essi (rimuovendo quindi tutti gli altri);
- all'interno dell'unione degli attributi sono identificati tutti gli insiemi di termini legati da relazioni di specializzazione (tra i quali erano quindi state definite relazioni di BT e NT) e vengono riorganizzati all'interno di gerarchie:

per ognuna di queste gerarchie è mantenuto solamente il termine più generale (che quindi ne sta a capo e ne può essere considerato il rappresentante) mentre sono rimossi tutti gli altri.

Esempio 9 Riferendoci al cluster Cl_1 di Figura 4.7, viene definita automaticamente la seguente classe globale:

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

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

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

- nome della *classe_globale*$_i$: il sistema propone un insieme di nomi candidati per la classe globale, sfruttando le relazioni terminologiche memorizzate nel Thesaurus ed i nomi delle classi appartenenti al cluster da cui questa classe globale deriva. Basandosi su questi suggerimenti, il progettista deve definire un nome che ben rappresenti il concetto di cui questa classe globale è rappresentante. Nel nostro esempio, riferendoci alla classe `GC`$_1$, il progettista decide di denominarla `University_Person`, essendo questa classe comprensiva di informazioni riguardanti sia studenti, sia professori di una data università;

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

– corrispondenza in *and*: l'attributo globale corrisponde all'unione, in uno specificato ordine, degli attributi locali ad esso corrispondenti.

Esempio 10 Riprendendo l'esempio dell'attributo globale `name`, è evidente come, in fase di interrogazione delle classi presenti nella sorgente S_1, l'attributo debba essere contemporaneamente tradotto dalla coppia `first_name` e `last_name`, e non in uno solo tra entrambi, in quanto è *equivalente* all'unione dei due. In questo modo, la query Q1 che richiede i nomi di tutte le persone presenti nella classe globale `University_Person`,

```
Q1: select name
    from University_Person
```

verrà tradotta, per quanto riguarda per esempio la classe `Research_Staff` di S_1, nella corrispondente query Q2:

```
Q2: select first_name, last_name
    from Research_staff
```

– corrispondenza in *or*: l'attributo globale è equivalente ad ogni singolo attributo locale con cui è messo in corrispondenza, e deve quindi originare più interrogazioni contemporaneamente per quella classe in cui vi è questo tipo di corrispondenza;

Esempio 11 Supponiamo di avere a che fare con l'integrazione di database in un dominio automobilistico. In particolare, dobbiamo integrare due tabelle appartenenti a due depositi di auto, in cui è memorizzata la quantità di auto a disposizione di una determinata marca, suddivise per anno di immatricolazione. Gli schemi delle due tabelle sono:

```
Cars(manufacturer,year,qty);
Bmw&Fiat(year,qty_bmw,qty_fiat).
```

Si può supporre che gli attributi `qty_bmw` e `qty_fiat` siano da considerare NT di `qty`, arrivando a definire la seguente classe globale:

```
Car(manufacturer,year,qty)
```

In questo caso, la corrispondenza tra l'attributo globale `qty` e la coppia `qty_bmw` e `qty_fiat` è ben diversa dal caso riportato nell'Esempio 10: l'attributo globale è infatti equivalente a ciascuno dei locali, e corrisponderà ad ognuno di essi presi singolarmente.
La seguente query Q3,

```
Q3: select  qty
    from    Car
    where   year = 1980
```

che richiede la quantità di auto dell'anno 1980, non deve infatti essere tradotta nella query Q4 (in riferimento soltanto alla classe `Bmw&Fiat`)

```
Q4: select  qty_bmw, qty_fiat
    from    Bmw&Fiat
    where   year = 1980
```

bensì nella coppia di query Q5 e Q6:

```
Q5: select  qty_bmw
    from    Bmw&Fiat
    where   year = 1980
```

```
Q6: select  qty_fiat
    from    Bmw&Fiat
    where   year = 1980
```

che devono essere spedite entrambe alla sorgente `Bmw&Fiat`;

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

  Esempio 12 Nella classe globale $\texttt{GC}_1$, il progettista è sicuramente in grado di specificare che l'attributo globale `rank` vale sempre `Student` nella

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

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

classe `School_Member`, nonostante questa informazione non sia esplicitamente memorizzata in alcuna struttura, bensì sia presente come metadato nel nome stesso dalla classe. Ci si avvarrà di questa funzionalità di MOMIS per inserire l'informazione;

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

Al fine di poter specificare in modo dichiarativo questo insieme di informazioni aggiuntive, è stata proposta in ODL_{I^3} una estensione alla classica definizione di classe attraverso il linguaggio ODL. Un esempio di descrizione di classe globale in ODL_{I^3} è dato nella firgura 4.8, in riferimento alla GC_1.

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

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

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

Figura 4.9: Mapping table di University_Person e Workplace

tipo *and* per la classe `University.Research_Staff` (si usa la *dot notation* per evidenziare al sorgente di origine).

In MOMIS, questa definizione di classe, e le informazioni in essa contenute, dà origine ad una struttura dati tabellare, definita *mapping table*, che sarà la base per tutto il processo di Query Processing. Come esempio, sono riportate in Figura 4.9 le mapping table dei cluster Cl_1 e Cl_4 di Figura 4.7, rappresentanti rispettivamente delle classi globali `University_Person` e `Workplace`.

È ancora in fase di studio il processo che porterà, partendo dalle definizioni di queste classi globali, alla determinazione delle relazioni che si possono instaurare tra queste classi: in particolare, devono essere analizzati criteri che permettano almeno l'instaurarsi di gerarchie di aggregazione, in presenza di attributi il cui dominio fa riferimento a classi appartenenti ad un'altra classe globale. Da approfondire anche il rapporto tra un attributo complesso, in uno schema ad oggetti, che fa riferimento ad una classe ed una corrispondente foreign key, in uno schema relazionale, che va a referenziare una classe equivalente a quella referenziata dall'attributo con dominio complesso.

4.9 Query Reformulation

In questa sezione verrà descritto il processo che, sfruttando le informazioni memorizzate nelle mapping table, realizza la Query Reformulation. Scopo di questo processo, che verrà attivato ogniqualvolta l'utente pone una interrogazione sullo

schema globale (esprimendola dunque in termini di classi e attributi globali), è arrivare alla definizione automatica delle interrogazioni che devono essere spedite alle diverse fonti di informazione per dare risposta alla query originale.
Mantenendo una impostazione simile ad altri approcci *semantici*, questo processo consiste di tre fasi distinte:

1. *ottimizzazione semantica*: sfruttando le informazioni semantiche presenti a livello di schema globale, ed eventuali regole di integrità definite dal progettista, è realizzata una ottimizzazione semantica dell'interrogazione posta dall'utente;

2. *formulazione del query plan*: basandosi sulla mapping table, e su un algoritmo di eliminazione delle sorgenti *inutili*, il sistema genera automaticamente un insieme di query da spedire alle sorgenti locali;

3. *ottimizzazione basata sulla conoscenza estensionale*: nel caso in cui siano disponibili all'interno del sistema delle regole definite tra le estensioni delle classi locali, queste informazioni sono sfruttate dal sistema per limitare ulteriormente il numero di sorgenti da interrogare, mantenendo inalterato l'insieme di risposta da presentare all'utente.

4.9.1 Ottimizzazione Semantica

In questa fase MOMIS opera sulla query dell'utente sfruttando le tecniche di ottimizzazione semantica [39, 40] supportate dagli ODB-Tools (descritti in Sezione 3.1), al fine di ridurre il costo del piano di accesso che sarà generato. Lo scopo è quello di estendere la query originale con ogni possibile restrizione che è logicamente implicata dalla query stessa e dalle regole definite sullo schema, per poter poi sfruttare eventuali indici ausiliari presenti nelle sorgenti locali.
Naturalmente, condizione necessaria per questa fase di ottimizzazione è la presenza di regole di integrità definite sullo schema globale: sebbene questa sembri un ipotesi molto forte (queste regole in fin dei conti le deve definire il progettista, il quale deve quindi essere a conoscenza di condizioni valide su tutte le estensioni di tutte le classi appartenenti al cluster da cui deriva una determinata classe globale), non è improbabile che in determinati domini applicativi la definizione di queste regole sia possibile e quindi auspicabile (facendo per esempio riferimento a database della Pubblica Amministrazione, o comunque appartenenti a domini fortemente regolamentati).

Supponiamo, per esempio, che nel nostro dominio universitario esista, a livello globale, una relazione che unisce i fondi di ricerca (rappresentati dall'attributo globale `budget`) all'area (`area`) di appartenenza del dipartimento a cui questi

fondi sono stati assegnati. Attraverso il linguaggio ODL$_{I^3}$, il progettista può quindi definire la seguente regola di integrità sulla classe globale `Workplace` (di cui in Figura 4.9 è riportato un frammento della mapping table):

rule R1 **for all** `X` **in** `Workplace`: (`X.budget` $>$ 60000)
then `X.area` = '`Scientific`'

Si consideri ora la seguente interrogazione: "Ritrova i nomi dei professori che lavorano in un dipartimento che ha un budget di ricerca maggiore di 60000 dollari", espressa dalla query Q7:

```
Q7: select  name
    from    University_Person
    where   rank = 'professor'
    and     works.budget > 80000
```

Il mediatore, sfruttando la regola R1, realizza l'espansione semantica della query (vedi Sezione 3.1.2) ed automaticamente ottiene la nuova query Q8:

```
Q8: select  name
    from    University_Person
    where   rank = 'professor'
    and     works.budget > 80000
    and     works.area = 'Scientific'
```

Come si vede, l'espansione semantica è stata realizzata al fine di aumentare il numero dei predicati presenti nella clausola *where*: questo processo, nonostante appesantisca la fase di query plan (ora deve essere riformulata una query più complessa rispetto all'originale), potrebbe alleggerire il lavoro di query processing delle singole sorgenti, nel caso in cui siano presenti indici secondari definiti sugli attributi aggiunti nella nuova query.
L'algoritmo di ottimizzazione utilizzato dagli ODB-Tools e presentato in [41] è polinomiale. Le prove effettuate hanno comunque evidenziato come l'*overhead* dovuto alla fase di ottimizzazione sia minimo comparato con i costi complessivi di trattamento della query. Su un insieme di 8 query realizzate su 10 differenti istanze di un database, l'ottimizzazione semantica delle query ha portato ad una riduzione media dei costi di esecuzione del 47%.

4.9.2 Formulazione del Query Plan

Una volta che il sistema MOMIS è in possesso della query globale, eventualmente ottimizzata, deve essere costruito un piano di accesso per andare a recuperare le informazioni da fornire in risposta all'utente. In questa sezione viene presentato il meccanismo attraverso il quale saranno definite le query da spedire alle sorgenti locali (meccanismo già abbastanza chiaro, almeno ad alto livello), mentre si rimanda alla Sezione 4.10 per la presentazione della fase di riunificazione dei dati (fase in cui sono tuttora rimasti aperti diversi problemi).

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

L'algoritmo di controllo ed eliminazione è riportato in Figura 4.10. Supponiamo che il cluster corrispondente alla classe globale interessata dalla interrogazione sia Cl_j: per ogni classe c_h appartenente al cluster sono controllati nella mapping table tutti i corrispondenti locali degli attributi globali sui quali,

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

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

Figura 4.10: Algoritmo di Controllo ed Eliminazione

nella query, sono state poste condizioni in **and** (denominate fattori boolean). Nel caso in cui, dopo la fase di controllo ed eliminazione, la query sia da spedire a quella classe, tutti i termini globali sono naturalmente tradotti nei corrispondenti termini locali, al fine di rendere *comprensibile* dalla sorgente la nuova query generata.

Si riprenda la query Q8, espressa nella Sezione 4.9.1:

```
Q8:  select  name
     from    University_Person
     where   rank = 'professor'
     and     works.budget > 80000
     and     works.area = 'Scientific'
```

Il passo 1 dell'algoritmo (controllo sui valori nulli) scarterebbe automaticamente dalla lista delle classi da interrogare le classi `School_Member`, `CS_Person`, `Student` and `University_Student` (in quanto in esse l'attributo `works` non ha alcun corrispondente), mentre il passo 2 (controllo sui valori di default) eliminerebbe le classi `School_Member` e `University_Student` (anche se in questo caso particolare erano già state eliminate), poiché il valore di `rank` non è compatibile con la condizione espressa nella query. Sono quindi automaticamente riformulate solamente due query, Q9 e Q10, da spedire rispettivamente alle sorgenti S_1 e S_2:

```
Q9:  select  first_name and last_name
     from    Research_Staff as R, Department as D
     where   R.dept_code = D.dept_code
     and     D.budget > 80000
     and     D.dept_area = 'Scientific'
```

```
Q10: select  name
     from    Professor as P
     where   P.rank = 'Professor'
     and     P.belongs_to.fund > 80000
     and     P.belongs_to.sector = 'Scientific'
```

Il risultato ottenuto non è certamente di poco conto: sono state scartate quattro classi dall'insieme di classi interrogabili (e dunque è sicuramente diminuito il numero di dati da analizzare), ed è stata eliminata una intera sorgente, alla quale quindi non occorre neppure accedere, avendo la sicurezza di non trovarvi informazioni richieste dalla query posta dall'utente.
Non è inoltre da escludere la presenza di regole di integrità definite questa volta non sullo schema globale, bensì sulle sorgenti locali (e quindi non fornite dal progettista del mediatore, ma fornite dal wrapper unitamente alla descrizione stessa dello schema in ODL_{I^3}). Poiché è improbabile che tutte le sorgenti siano in grado di realizzare una ottimizzazione semantica, e ritenendo *pesante* e fuori luogo includere queste funzionalità in tutti i wrapper che agiscono su sorgenti non *intelligenti*, la scelta è stata quella di spostare a livello di mediatore questa ulteriore ottimizzazione.

Supponendo che la seguente rule sia stata ricevuta assieme allo schema della sorgente `Computer_Science` (S_1):

```
rule R2  for all X in Division: (X.fund > 60000)
         then  X.employee_nr > 20
```

MOMIS potrà sfruttarla per effettuare una espansione semantica della query Q10, giungendo alla formulazione della query locale Q11:

```
Q11: select  name
     from    Professor as P
     where   P.rank = 'Professor'
     and     P.belongs_to.fund > 80000
     and     P.belongs_to.sector = 'Scientific'
     and     P.belongs_to.employee_nr > 20
```

Questa ulteriore condizione potrebbe essere utilmente impiegata nella sorgente interrogata, nel caso in cui sia presente un indice definito sull'attributo `employee_nr`.

4.9.3 Ottimizzazione basata sulla Conoscenza Estensionale

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

1. se fornite in modo manuale (dal progettista), comportano un'enorme mole di lavoro: è praticamente impensabile che il progettista controlli fisicamente tutte le estensioni di tutte le classi coinvolte nel sistema, per vedere se vi sono tra loro dati duplicati, o in qualche modo correlati;

2. se cercate in modo automatico, è comunque un compito non banale ideare un algoritmo di controllo e confronto dei dati memorizzati nelle varie sorgenti (si pensi ad esempio che questo confronto dovrebbe essere riattivato successivamente ad ogni operazione di aggiornamento, o di inserimento,...).

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

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

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

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

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

Si supponga che il progettista sia in grado di definire, a livello estensionale, la seguente proprietà:
`School_Member` $\subset_{ext}$ `University_Student`
valida tra le classi appartenenti al cluster Cl_1.

Il sistema potrebbe sicuramente utilizzare questa conoscenza durante la fase di Query Processing, per minimizzare il numero di accesso ai dati. In particolare, se l'utente vuole ritrovare tutti i nomi degli studenti iscritti alla facoltà di Economia (attraverso la seguente query):

```
Q12:  select  name
      from    University_Person
      where   faculty = 'Economics'
```

MOMIS, attraverso i passi appena descritti, arriverebbe alla formulazione di due query locali, da spedire rispettivamente alle classi `School_Member` and `University_Student` nelle sorgenti S_1 e S_3. Utilizzando però la conoscenza espressa dalla proprietà interschema definita dal progettista tra queste due classi, si può dedurre l'inutilità di spedire questa query alla classe `School_Member` in quanto, relativamente agli attributi specificati, i dati sono inclusi nell'insieme dei dati ricuperabili da `University_Student`. Il risultato sarà dunque la generazione (e spedizione) dell'unica query locale Q13:

```
Q13:  select  name
      from    University_Student
      where   faculty_name = 'Economics'
```

4.10 Unificazione dei dati

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

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

- supponendo che esista per esempio una chiave `nome` in tutte le tabelle interrogate, niente ci assicura che persone con nomi uguali siano effettivamente la stessa persona in contesti diversi;

- supponendo invece che come chiave interna delle tabelle si utilizzi un codice (pensiamo ad un esempio magazzino), il problema è più complicato: è molto probabile che si usino codici diversi per identificare lo stesso concetto, pur se il dominio degli attributi `codice` è uguale in tutte le sorgenti.

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

Capitolo 5

Il modulo software $\mathbf{SIM}_1$

In questo capitolo verrà presentato il modulo software $\mathbf{SIM}_1$, che realizza la prima parte del progetto MOMIS.

Insieme all'architettura del modulo, verranno presentate le procedure più interessanti utilizzate al suo interno, includendo inoltre la descrizione delle strutture dati e degli algoritmi più significativi realizzati. Il software è stato sviluppato in linguaggio ANSI C, utilizzando il compilatore 2.7.2 della GNU (Free Software Fundation, Inc.). La piattaforma hardware utilizzata è una SUN Sparc 20 con sistema operativo Solaris 2.5.

5.1 Obiettivi ed Architettura del Modulo

Il modulo $\mathbf{SIM}_1$ (**S**chemata **I**ntegrator **M**odule, prima versione) si pone come obiettivo la realizzazione della parte iniziale del progetto MOMIS, relativa alla fase di integrazione degli schemi delle sorgenti di informazione.
L'architettura del modulo è riportata in Figura 5.1: a partire dagli schemi delle sorgenti che devono essere integrate, forniti in ODL_{I^3}, e attraverso l'interazione col progettista del sistema di integrazione, $\mathbf{SIM}_1$ porta alla definizione semi-automatica del Thesaurus di relazioni terminologiche. Dovrà dunque interfacciarsi, a monte, con un ulteriore modulo di integrazione (in fase di sviluppo presso il Dipartimento di Scienze dell'Informazione dell'Università di Milano), denominato Artemis-Tools, che calcoli i cluster veri e propri di classi (applicando le tecniche di clustering definite ed esposte nel Capitolo 4), e durante la sua esecuzione con gli ODB-Tools, moduli software preesistenti, utilizzati in diverse fasi del processo di derivazione del Thesaurus. In particolare, il processo complessivo si divide in due fasi ben distinte:

1. *estrazione automatica delle relazioni dagli schemi* ODL_{I^3}: corrispondente alla fase esposta in Sezione 4.5 step 3, il modulo deriva automaticamente un

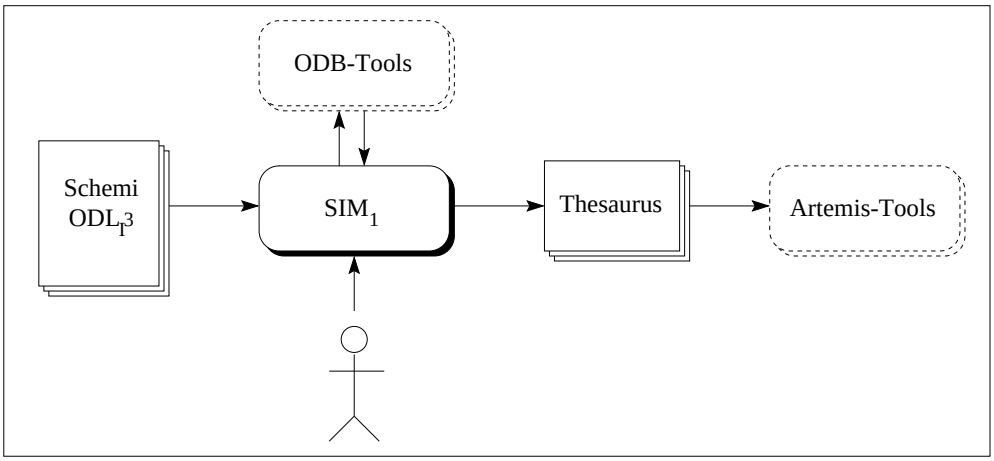


Figura 5.1: Il modulo **SIM**$_1$

insieme di relazioni, sfruttando la conoscenza codificata nelle descrizioni degli schemi;

2. *definizione del Thesaurus definitivo*: corrispondente agli step 3 e 4 di Sezione 4.5, il modulo, sulla base delle relazioni inserite manualmente dal progettista, definisce il Thesaurus definitivo (inferendone di nuove), che sarà la base per la successiva fase di determinazione dei cluster (realizzata invece dal modulo Artemis-Tools).

Come rappresentato in Figura 5.2, l'architettura di **SIM**$_1$ dovrà necessariamente rispecchiare queste due fasi, strutturandosi dunque in due moduli separati:

1. **SIM**$_1$**A**: realizza la prima fase, analizzando in particolare gli schemi provenienti da sorgenti relazionali e ad oggetti;

2. **SIM**$_1$**B**: realizza la seconda fase, ricevendo in ingresso sia le relazioni dedotte da **SIM**$_1$**A**, sia quelle inserite dal progettista, e fornendo in uscita un file in cui sono rappresentate tutte le relazioni terminologiche facenti parte del Thesaurus.

Nelle sezioni seguenti saranno analizzate l'architettura e gli algoritmi più interessanti di entrambi i sottomoduli.

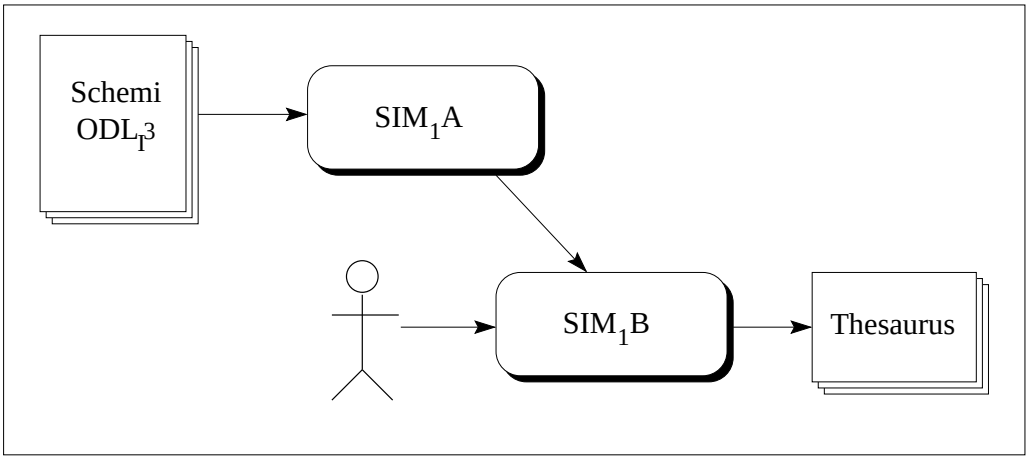


Figura 5.2: Architettura del modulo **SIM**$_1$

5.2 Il modulo SIM$_1$A

Il modulo **SIM**$_1$**A** ha il compito di realizzare una analisi preliminare degli schemi delle sorgenti, al fine di esplicitare un insieme di relazioni terminologiche in realtà già implicitamente presenti negli schemi ricevuti. L'architettura del modulo è riportata in Figura 5.3.

Il primo passo del processo è l'acquisizione degli schemi delle sorgenti, espressi attraverso il linguaggio ODL$_{I^3}$: per fare questo, è stato necessario realizzare una estensione al preesistente parser di ODL, al fine di includervi le modifiche apportate a questo linguaggio descrittivo, per supportare le nuove esigenze di un ambiente di integrazione di informazioni. Il parser è stato realizzato attraverso due strumenti software, FLEX e BISON, che facilitano la scrittura di programmi in linguaggio C per l'analisi e l'interpretazione di sequenze di caratteri che costituiscono un dato testo sorgente. In particolare, sono state realizzate le seguenti funzioni:

- **yylex**: effettua l'analisi lessicale del testo sorgente, riconoscendo nelle serie di caratteri ricevuti in input dei predefiniti *token* (ovvero delle espressioni regolari definite dal programmatore);

- **yyparse**: interpreta una sequenza di token e riconosce la sintassi definita nel file di input; in questo modo viene automaticamente realizzato un parser semplicemente definendo la sintassi da riconoscere, a sua volta espressa in una notazione molto simile alla *Bakus-Naur Form* (BNF).

Fase successiva all'acquisizione degli schemi è l'invocazione di una procedura di controllo di coerenza, che agisce sulle strutture dati appena *caricate* dal parser,

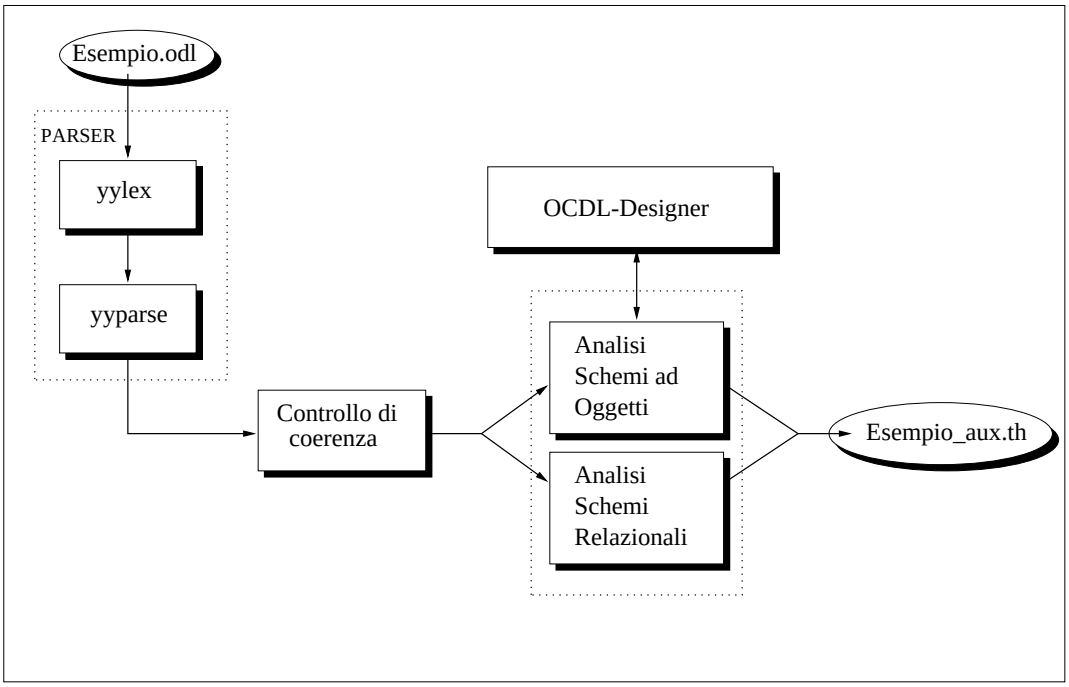


Figura 5.3: Architettura del modulo **SIM_1A**

che realizza un'ulteriore serie di controlli (per rilevare, ad esempio, se all'interno di una stessa classe esistano più attributi con lo stesso nome) impossibili da realizzare parallelamente all'acquisizione stessa degli schemi.

Esempio 13 Si riporta di seguito la struttura dati che il parser provvede a memorizzare a seguito del riconoscimento di una classe, sulla quale quindi andranno ad agire tutte le procedure successive.

```
struct s_interface_type
   {
   char         *name;
                /* nome interfaccia */
   char         fg_interf_view;
                /* indica se l'"interfaccia" e' una classe
                oppure una view */
   struct       s_iner_list  *iner;
                /* lista delle superclassi (ereditarieta')*/
   struct       s_typeprop *typeprop;
                /* lista delle proprieta' della classe:
```

```
                sorgente, extent, chiave, foreign key */
struct          s_prop_list  *prop; */
                /* lista degli attributi e relazioni  */
struct s_interface_type *next;
                /* lista globale delle interface */
char            fg_used;
                /* flag usato per evirtare i cicli */
}   *Interface_type;
```

La fase di analisi degli schemi vera e propria, allo scopo di derivare relazioni terminologiche, inizia però al passo successivo, e viene realizzata separatamente da due procedure diverse:

- *analisi degli schemi relazionali*: sono analizzati gli schemi provenienti da sorgenti relazionali, per dedurre relazioni di tipo RT dalle definizioni di foreign key.

  Esempio 14 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 )
  { 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 di foreign key sull'attributo `room_code`, generando in questo modo la relazione:

  $\langle$`Section` RT `Room`$\rangle$

- *analisi degli schemi ad oggetti*: per la fase di analisi degli schemi ad oggetti, **SIM$_1$A** fa uso del modulo OCDL-Designer(facente parte dei preesistenti ODB-Tools) a cui è demandata l'analisi delle gerarchie che legano le classi, siano esse esplicitamente indicate negli schemi (ereditarietà dirette, attributi con domini complessi), siano esse dedotte attraverso l'algoritmo di sussunzione ed espansione semantica.

Esempio 15 Si consideri ora la classe `Professor`, proveniente dalla sorgente ad oggetti `Computer_Science`:

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

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

⟨`Professor` NT `CS_Person`⟩
⟨`Professor` RT `Division`⟩

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

Il compito del modulo **SIM_1A** termina con la scrittura di un file di output (che terminerà con l'appendice `_aux.th`), che il progettista del sistema dovrà a sua volta integrare.

5.3 Il modulo SIM_1B

Più complessa, rispetto a quella appena presentata, risulta essere l'architettura del modulo successivo, **SIM_1B**, riportata in Figura 5.4.

Il punto di partenza è l'insieme delle relazioni terminologiche sino a quel punto memorizzate nel Thesaurus (comprendente dunque sia quelle estratte dal modulo **SIM_1A**, sia quelle aggiunte dal progettista). Il suo compito è modificare gli schemi sorgenti, sulla base delle informazioni che si possono dedurre dalle relazioni già memorizzate, ottenendo nuovi schemi da analizzare attraverso OCDL-Designer, in modo da inferire nuove relazioni terminologiche.

Per arrivare alla definizione del Thesaurus di relazioni definitivo sono realizzati dal modulo diversi passaggi, alcuni dei quali saranno semplicemente descritti in questa sezione, mentre altri saranno approfonditi nelle successive.

Le relazioni terminologiche ricevute in input sono in una prima fase suddivise in tre liste separate:

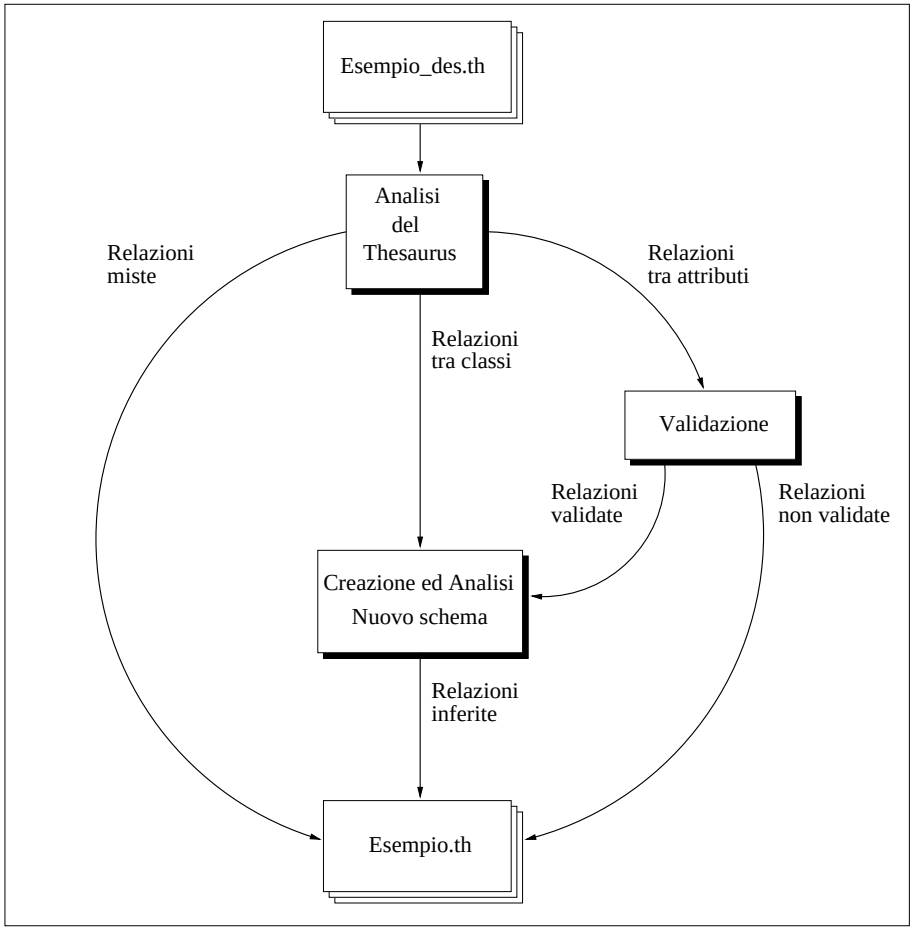


Figura 5.4: Architettura del modulo **SIM$_1$B**

1. *relazioni tra attributi*: relazioni che coinvolgono coppie di attributi sono posizionate in una apposita lista, che dovrà essere analizzata e processata per prima; questo tipo di relazioni dovranno infatti, prima di concorrere alla modifica degli schemi originali, essere validate;

2. *relazioni tra classi*: le relazioni che coinvolgono coppie di classi saranno analizzate in un secondo momento (e dunque vengono posizionate in una apposita lista), successivamente alla analisi delle relazioni tra attributi validate;

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

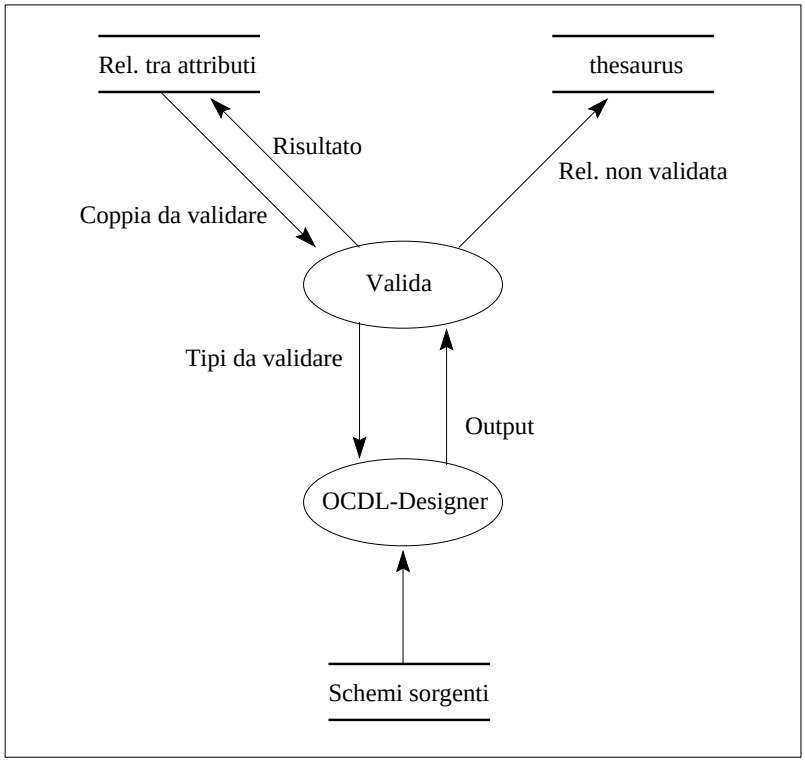


Figura 5.5: DFD della procedura di Validazione

lizzate dal modulo **SIM_1B**, dunque sono memorizzate direttamente nel Thesaurus definitivo di output.

5.3.1 Validazione delle relazioni tra attributi

In Figura 5.5 è riportato il Data Flow Diagram della procedura di validazione, così come è realizzata all'interno del modulo **SIM_1B**. L'input di questa procedura è costituito dalla lista delle relazioni tra attributi inserite dal progettista, che quindi devono superare una fase di validazione. In particolare, seguendo il processo esposto nello step 3 di Sezione 4.5, il modulo realizza i seguenti passi:

- ogni coppia di attributi, legati da una relazione (sia essa di SYN, BT, o NT) viene esaminata singolarmente, estraendola dalla apposita lista;
- sono eseguiti i controlli sui tipi dei domini di questi attributi: se i tipi coincidono, la relazione è automaticamente marcata come valida; se i tipi non coincidono, la relazione dovrà essere validata facendo uso di OCDL-Designer: viene dunque inserita in una opportuna lista di relazioni ancora da verificare;

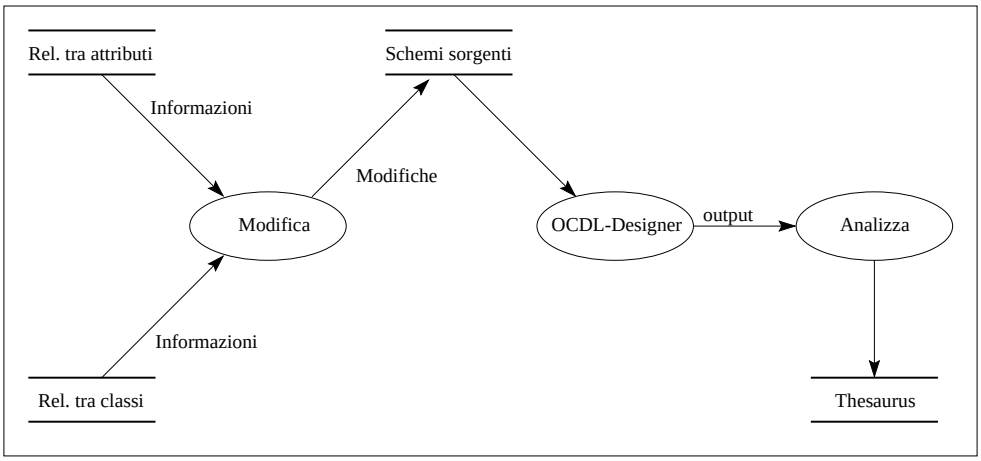


Figura 5.6: DFD della procedura di modifica degli schemi sorgenti

- se, dopo aver esaminato tutte le relazioni, la lista delle relazioni da esaminare usando OCDL-Designer non è vuota, viene invocato il componente esterno, fornendogli in input i tipi da controllare e gli schemi originali delle sorgenti (naturalmente tutto tradotto in **OCDL**, logica descrittiva utilizzata dagli ODB-Tools);

- se è stato lanciato OCDL-Designer, ne viene esaminato l'output per vedere, per ogni coppia di attributi ancora da validare, se i rispettivi domini sono legati da una relazione di sussunzione compatibile con la relazione terminologica tra loro definita: in caso positivo, la relazione è marcata come *valida*;

- le relazioni tra attributi che non hanno superato la validazione sono inserite nel Thesaurus di uscita, in quanto non potranno essere utilizzate all'interno del modulo; le relazioni valide andranno invece ulteriormente processate nella fase successiva.

5.3.2 Modifica degli schemi sorgenti

Le relazioni tra attributi che hanno superato il processo di validazione, come pure le relazioni tra classi (ivi comprese sia quelle dedotte dal modulo $\mathbf{SIM}_1\mathbf{A}$, sia quelle aggiunte dal progettista) costituiscono la base per la modifica degli schemi sorgenti, volta ad ottenere un nuovo schema da cui inferire nuove relazioni terminologiche (vedi Figura 5.6).

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.
Dal punto di vista realizzativo, ogni nodo appartenente a questi alberi gerarchici è memorizzato in una struttura di questo tipo:

```
struct node_el
  {
   char nname[64]; /*nome dell'elemento*/
   char nsource[64]; /* nome della sorgente */
   char fg_capo[5]; /* flag che mi dice  se questo elemento */
                    /* e' a capo di un albero */
   char nsost[64]; /*  nome con cui va sostituito */
   struct lista_el *listasynd; /* lista dei sinonimi diretta*/
   struct lista_el *listasyni; /* lista dei sinonimi inversa*/
   struct lista_el *listant;  /* lista dei narrower terms */
   struct node_el *listanodi; /* lista di tutti i i nodi */
                           /* allocati: facilita la ricerca */
   char fg_used[2]; /* permette la ricorsione */
   char fg_val[2]; /*e' 1 se ' validato, 0 altrimenti */
  };
```

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 16 Con riferimento all'esempio universitario, gli alberi gerarchici creati da **SIM$_1$B** sono mostrati in Figura 5.7. Si può notare che, per quanto riguarda gli attributi `first_name` e `last_name` (entrambi appartenenti alla classe `School_Member`, fra le altre), dovranno entrambi essere sotituiti dall'attributo `name`, a capo del loro albero. Di conseguenza, nella fase successiva, all'interno di

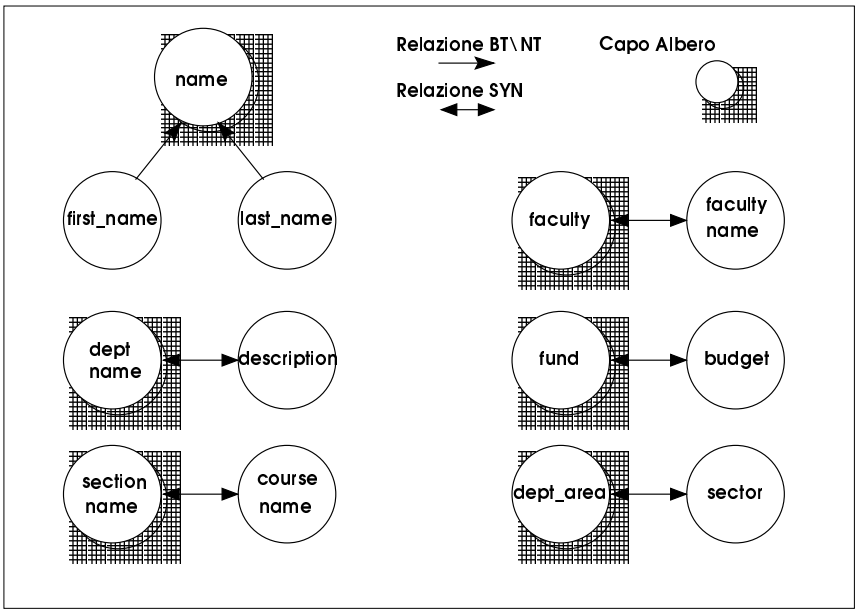


Figura 5.7: Alberi gerarchici

`School_Member` andrà rimosso un attributo tra i due designati con `name`, onde evitare inconsistenze nelle descrizioni degli schemi.
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: ⟨`Computer_Science.name` SYN `Tax_Position.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.

Utilizzo delle relazioni tra classi Così come è stato fatto per le relazioni tra attributi, **SIM$_1$B** 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);

- 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 (che a questo livello ha una direzione definita, ma che nel Thesaurus potrà essere percorsa in entrambe le direzioni, come descritto in Sezione 4.6);

- relazioni SYN: non essendo comprensibile, per gli ODB-Tools, una relazione ciclica del tipo (`ClasseA` *isa* `ClasseB`) e (`ClasseB` *isa* `ClasseA`), al fine di rendere le due classi effettivamente equivalenti, ne sono duplicate tutte le proprietà, ottenendo in questo modo due classi con la medesima struttura (di fatto equivalenti).

Inferenza di nuove relazioni terminologiche Sulla base dei nuovi schemi ottenuti, come descritto nel passo 4 di Sezione 4.5, si determina un nuovo insieme di relazioni terminologiche da essi inferite. In questa fase, utilizzando il modulo OCDL-Designer (a cui sono passate le descrizioni degli schemi, precedentemente tradotte in linguaggio **OCDL**), tutti gli schemi sorgenti, opportunamente modificati, sono visti come un unico schema, al fine di inferire pure le relazioni tra classi appartenenti a sorgenti diverse.

Lo scopo rimane comunque il dedurre il maggior numero di relazioni (siano esse implicate dalle informazioni precedentemente inserite attraverso le modifiche agli schemi originali, siano implicate da legami dedotti tra le strutture stesse delle classi con l'uso dell'algoritmo di calcolo della sussunzione): maggiore sarà l'insieme delle relazioni inferite, maggiore il legame che si creerà tra le classi in fase di calcolo dei coefficienti di affinità (descritta in Sezione 4.6 e realizzata dal modulo Artemis), e minore il compito di inserimento manuale di nuove relazioni da parte del progettista.

5.4 Istruzioni per l'utente

Con riferimento all'architettura del software di Figura 5.2, il modulo è stato suddiviso in due eseguibili separati (**SIM$_1$A** e **SIM$_1$B**), in quanto saranno separati i tempi in cui questi andranno utilizzati: non è infatti pensabile che, in tempo reale, il progettista sia in grado di inserire le relazioni che il primo modulo non ha derivato dagli schemi, essendo la fase di inserimento manuale un compito non banale e contemporaneamente fondamentale per l'intero processo di integrazione.

I moduli sono eseguibili rispettivamente tramite i comandi:

`sim1a nomefile` oppure `sim1b nomefile`.

Per quanto riguarda invece i file di input/output, essi dovranno avere (o acquisteranno) le seguenti estensioni:

- `.odl`: contiene le descrizioni originali degli schemi in ODL_{I^3};
- `.th`: contiene le relazioni appartenenti al Thesaurus;
- `.sc`: contiene le descrizioni degli schemi tradotti in **OCDL**;
- `.inf`: contiene le relazioni inferite da OCDL-Designer;
- `.vf`: contiene le informazioni necessarie per la rappresentazione degli schemi (originali e integrati) sul sito web.

Conclusioni

In questa tesi è stato progettato un sistema mediatore, MOMIS, il cui compito è fornire accesso integrato ad una molteplicità di sorgenti eterogenee di informazioni. In particolare, MOMIS va a collocarsi all'interno di una vasta e recente area di ricerca, l'Integrazione Intelligente di Informazioni, che si pone come obiettivo l'utilizzo di tecniche di Intelligenza Artificiale nell'ambito dell'integrazione dei dati. In che misura è stato raggiunto l'obiettivo? In altri termini, quanto è *intelligente* lo strumento sviluppato?

La risposta non è sicuramente univoca, e il sistema va comunque messo a confronto con le soluzioni già presenti in letteratura. Si è senza dubbio raggiunto un risultato innovativo: i sistemi precedentemente sviluppati realizzano una buona (a volte eccellente) fase di "Query Processing" (ovvero di gestione delle interrogazioni), ma sono per la maggior parte dei casi rinunciatari nella fase di integrazione, lasciata alla responsabilità del progettista del sistema. Sotto questo punto di vista, si è allora sicuramente raggiunto lo scopo che ci si era prefissi: con l'aiuto dell'operatore, attraverso un processo semi-automatico, MOMIS giunge non solo alla unificazione degli schemi delle sorgenti, ma pure alla creazione di uno schema globale vero e proprio, direttamente interrogabile dall'utente. Per arrivare a questo risultato, é stato sviluppato un sistema che, sfruttando i suggerimenti forniti dal progettista attraverso un linguaggio dichiarativo (ODL_{I^3}), giunge alla creazione di un *mediatore di informazioni* in grado di gestire la maggior parte delle problematiche che si possono incontrare in questo campo. Importante è anche, a nostro avviso, la componente *intelligente* dell'intero processo: sebbene la presenza del progettista rimanga essenziale, le tecniche di Intelligenza Artificiale che erano a disposizione sono risultate essere una componente fondamentale per l'intero sistema, usate ampliamente e nelle diverse fasi dell'integrazione. Se l'obiettivo fosse stato posto in termini di realizzazione di un Integratore Intelligente di Informazioni che potesse procedere in modo completamente autonomo ed indipendente, non solo MOMIS avrebbe fallito, ma sarebbe stato eccessivamente velleitario l'obiettivo stesso. Si noti che la natura stessa del problema dell'inte-

grazione di fonti di informazione (caratterizzata da un alto livello di ambiguità) non può prescindere dall'interazione con il progettista della visione integrata delle sorgenti. È allora da ritenersi un successo l'aver realizzato un sistema che *guidi* l'operatore attraverso il processo di unificazione, sfruttandone al meglio i suggerimenti per dedurre nuove relazioni, e coadiuvandolo nella amministrazione di un problema che, soprattutto con l'aumentare del numero delle sorgenti da integrare, diverrebbe comunque ingestibile anche da parte di più operatori.

Da non mettere in secondo piano inoltre è l'utilizzo delle tecniche di Inferenza proposto nella fase di "Query Reformulation", dove attraverso un algoritmo di ottimizzazione semantica si giunge ad un effettivo risparmio sui costi di realizzazione delle interrogazioni.

Per quanto riguarda invece i prossimi sviluppi del progetto, si può ritenere completamente conclusa la fase di Integrazione degli Schemi, mentre rimangono problemi aperti nella fase di Query Reformulation (dove comunque sono già abbastanza chiare le future linee guida) e soprattutto nella fase di riunificazione delle risposte ottenute dalle sorgenti. In particolare, dovranno essere approfonditi gli effetti dell'utilizzo di assiomi estensionali, definiti dal progettista stesso sulla base dell'analisi dei dati memorizzati nelle sorgenti, che possano fornire criteri sicuri per l'identificazione di tutte le informazioni che si riferiscono alla stessa entità del mondo reale.

Parallelamente a questo, dovranno inoltre essere realizzati i componenti wrapper, per rendere effettivamente utilizzabile l'intero sistema.

Appendice A

Glossario I^3

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

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

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

A.1 Architettura

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

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

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

A.2 Servizi

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

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

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

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

A.3 Risorse

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

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

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

A.4 Ontologia

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

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

Appendice B

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

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

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

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

Appendice C

Esempio in ODL$_{I^3}$

Di seguito é riportata la descrizione, attraverso il linguaggio ODL$_{I^3}$, dell'esempio di riferimento di Sezione 4.3.

```
UNIVERSITY source:

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

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

interface Room
```

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

COMPUTER_SCIENCE source:

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

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

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

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

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

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

Tax_Position source:

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

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

Appendice D

“An Intelligent Approach to Information Integration” accettazione FOIS 98

Si riporta la documentazione di accettazione dell’articolo “An Intelligent Approach to Information Integration”, che sará presentato al Convegno Internazionale Formal Ontology in Information System - FOIS98, e nel quale sono contenuti i risultati della presente tesi. L’articolo stesso non è stato riprodotto in Appendice in quanto la sua pubblicazione in documenti ufficiali è di competenza esclusiva dell’editore.

Appendice E

SIM_1: il codice sorgente

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] S.Bergamaschi, S.Castano, S.De Capitani di Vimercati, S.Montanari, and M.Vincini. Exploiting schema knowledge for the integration of heterogeneous sources. *Submitted for: Int. Conf. on Very Large Databases VLDB98*.

[3] S.Bergamaschi, S.Castano, S.De Capitani di Vimercati, S.Montanari, and M.Vincini. Integration architectures for semistructured and heterogeneous data. *Submitted for: Information Systems. special Issue on Semistructured Data*.

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

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

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

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

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

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

[10] Daniel P.Miranker and Vasilis Samoladas. Alamo: an architecture for integrating heterogenous data sources. In *Proceedings of the 4th KRDB Workshop*, Athens, Greece, August 1997.

[11] Oliver M.Duschka and Micheal R.Genesereth. Infomaster - an information integration toolkit. Technical report, Department of Computer Science. Stanford University, 1996.

[12] V.S. Subrahmanian, Sibel Adali, Anne Brink, J.ames, J. Lu, Adil Rajput, Timothy J. Rogers, Robert Ross, and Charles Ward. Hermes: A heterogeneous reasoning and mediator system. Available at http://www.cs.umd.edu/projects/hermes/overview/paper/index.html.

[13] Alon Levy, Dana Florescu, Jaewoo Kang, Anand Rajaraman, and Joanne J. Ordille. The information manifold project. Available at http://www.research.att.com/ levy/imhome.html.

[14] H. Garcia-Molina et al. The tsimmis approach to mediation: Data models and languages. In *NGITS workshop*, 1995. Available ftp://db.stanford.edu/pub/garcia/1995/tsimmis-models-languages.ps.

[15] Y.Papakonstantinou H.Garcia-Molina and J.Ullman. Medmaker: a mediation system based on declarative specification. Technical report, Stanford University, 1995. ftp://db.stanford.edu/pub/papakonstantinou/1995/medmaker.ps.

[16] Y.Papakonstantinou, H.Garcia-Molina, and J.Widom. Object exchange across heterogeneous information sources. Technical report, Stanford University, 1994.

[17] M.J.Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Object exchange across heterogeneous information sources. Technical report, Stanford University, 1994.

[18] M.T. Roth and P. Scharz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proc. of the 23rd Int. Conf. on Very Large Databases*, Athens, Greece, March 1995.

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

[20] Y. Arens, C.Y. Chee, C. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[21] Y. Arens, C. A. Knoblock, and C. Hsu. Query processing in the sims information mediator. *Advanced Planning Technology*, 1996.

[22] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

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

[24] 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.

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

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

[27] J.P. Ballerini, D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. A semantics-driven query optimizer for oodbs. In *Int. Workshop on Description Logics*, Rome, Italy, June 1995.

[28] 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.

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

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

[31] 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.

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

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

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

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

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

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

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

[39] S. Shenoy and M. Ozsoyoglu. A system for semantic query optimization. *ACM-SIGMOD*, pages 181–195, May 1987.

[40] S. Shenoy and M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. and Data Engineering*, 1(3):344–361, September 1989.

[41] D. Beneventano, S. Bergamaschi, and C. Sartori. Semantic query optimization by subsumption in oodb. In *Proc. of the Int. Workshop on Flexible Query-Answering Systems (FQAS96)*, pages 167–185, Roskilde, Denmark, May 1996.