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

# Trattamento della Conoscenza Estensionale nel sistema MOMIS

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Francesco Venuta

Correlatore
Ing. Domenico Beneventano

Controrelatore
Chiar.mo Prof. Michele Colajanni

Anno Accademico 1999 - 2000

Parole chiave:

Base Extension
Gerarchia Estensionale
Query Processing
Database eterogenei
Assiomi estensionali

# RINGRAZIAMENTI

*Un sentito ringraziamento va alla Professoressa Sonia Bergamaschi ed all'Ing. Domenico Beneventano per l'aiuto fornitomi durante la realizzazione della presente tesi e per la costante disponibilità dimostrata.*
*Vorrei inoltre ringraziare tutti i componenti del team MOMIS, in particolare l'Ing. Alberto Corni, per i consigli ed i chiarimenti di ordine pratico ed implementativo.*
*Un ringraziamento speciale va poi ai miei genitori, che hanno reso possibile tutto ció e alla mia fidanzata, Claudia, che mi ha sempre sostenuto e sopportato.*

# Indice

# Elenco delle figure

# Introduzione

Lo sviluppo delle tecnologie telematiche, sia nell'ambito dei sistemi di elaborazione, che in quello delle reti di calcolatori, ha portato ad una sempre maggiore presenza di sorgenti informative determinando un vera e propria esplosione nella quantità e nella varietà di dati accessibili.
Parallelamente alla crescita delle sorgenti di informazione si é potuto osservare un degrado per quello che concerne la qualità dell'informazione stessa. Il dato, infatti, spesso si trova collocato in documenti non strutturati, quindi di difficile lettura da parte degli strumenti tradizionali di analisi, inoltre, a causa del fenomeno dell'*information overload*, risulta difficile agli utenti discernere tra le varie sorgenti presenti quelle che contengano dati significativi. Il problema che sta alla base é quindi quello della presenza di una grande varietà di sorgenti, disomogenee le une rispetto alle altre, ma anche a volte di tipo "$semistrutturato$", e quindi *disomogenee* anche nella propria costruzione. Tale eterogeneità dei sistemi può infatti presentarsi in diversi modi sia considerando le differenti piattaforme hardware e software su cui una sorgente è implementata (ad esempio diversi DBMS e linguaggi di interrogazione), sia considerando i modelli di dati utilizzati (relazionale, object-oriented, ecc...), sia facendo riferimento agli schemi usati per la rappresentazione della struttura logica dei dati memorizzati.

In un contesto di questo tipo, nel quale appunto convive una forte eterogeneità sia delle tipologie di sorgenti di informazione, sia delle modellazioni dei dati contenuti nelle singole sorgenti, risulta evidente che, per potere reperire le informazioni desiderate, sarebbe necessario avere una conoscenza specifica sia del contenuto, sia delle strutture, sia dei linguaggi di interrogazione propri delle singole sorgenti. L'utente dovrebbe quindi essere in grado di scomporre la propria interrogazione in una sequenza di sotto-interrogazioni rivolte alle singole sorgenti di informazioni provvedendo poi a compiere una operazione di "$trasformazione$" dei risultati parziali, in modo da ottenere una risposta unificata. Tutto ciò dovrebbe essere fatto tenendo presente le possibili trasformazioni che possono subire i dati, le relazioni che li legano, le proprietà che possono avere in comune e le discrepanze sussistenti tra le diverse rappresentazioni.

Come si é detto, essendo poi il numero delle sorgenti di informazione in continua crescita ed essendo di conseguenza anche l'eterogeneità dei dati costantemente crescente, risulta indispensabile progettare dei meccanismi per automatizzare e rendere pertanto piú agevole per l'utente il processo di interrogazione e di reperimento delle informazioni dalle singole sorgenti.

Questa tesi si inserisce all'interno di un progetto più ampio denominato **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [18, 16, 20, 19, 21, 25, 26, 47], sviluppato con lo scopo di realizzare un processo semi-automatico di integrazione di sorgenti eterogenee e distribuite.
**MOMIS** adotta un'architettura a tre livelli con un *Mediatore* che ne occupa la parte centrale ed avente lo scopo di fornire una visione integrata degli schemi locali. Tale visione integrata degli schemi costitutivi le singole sorgenti permette di effettuare delle interrogazioni prescindendo dalla conoscenza della specifica sorgente, ma utilizzando le informazioni che derivano dall'unico schema, sintesi delle diverse eterogeneità. Il *Mediatore* rappresenta pertanto il cuore del sistema ed ha il compito di realizzare l'integrazione degli schemi e di provvedere alla gestione dei risultati delle operazioni di query.

Elementi indubbiamente innovativi di questo progetto sono rappresentati dall'impiego di un approccio *semantico* e dall'uso di logiche descrittive per la rappresentazione degli schemi delle sorgenti. Questi elementi introducono, infatti comportamenti intelligenti che permettono di sfruttare al meglio le conoscenze intensionali, semantiche ed estensionali sia inter-schema sia intra-schema, per generare una vista globale il più possibile espressiva e sulla quale potere effettuare delle interrogazioni significative.
Di fondamentale importanza è quindi l'impiego di ODB-Tools , un ambiente software sviluppato presso l'Università di Modena, in grado di realizzare la validazione di schemi ad oggetti e l'espansione semantica delle interrogazioni.

Piú precisamente l'obiettivo del seguente lavoro é stato quello di progettare e implementare le strutture dati e le procedure per la gestione degli assiomi logici che legano la conoscenza estensionale[1] delle sorgenti da integrare. Gli sforzi principali sono stati indirizzati allo studio e realizzazione di una metodologia adeguata per il calcolo delle informazioni necessarie al modulo Query Manager del *Mediatore* per la determinazione di una risposta al contempo corretta e, quanto piú possibile, completa.

Nel **Capitolo 1** viene dapprima introdotta l'architettura di riferimento $I^3$ per i sistemi di Integrazione di Informazioni, per poi illustrare le scelte implementative fatte in MOMIS. Viene anche presentato il componente ODB-Tools usato al fine di introdurre, nel sistema, comportamenti intelligenti.

Il **Capitolo 2** é interamente dedicato alla Conoscenza Estensionale: viene det-

[1] Il concetto di assiomi logici e Conoscenza Estensionale verrá chiarito nel Capitolo 2

tagliatamente spiegato cosa é, a cosa serve, come viene ricavata. Nel **Capitolo 3** viene presentato l'algoritmo adottato per il calcolo della Conoscenza Estensionale, i procedimenti adottati e le varie scelte di progetto. Infine il **Capitolo 5** si occuperá di illustrare lo studio e la realizzazione del software per l'implementazione degli algoritmi illustrati, motivando eventuali scelte progettuali ed indicando eventuali sviluppi futuri.

Sono inoltre presenti quattro appendici. In particolare in Appendice A viene riportato un glossario dei termini usati in ambito $I^3$ , in Appendice B viene mostrato un esempio completo in $ODL_{I^3}$ che sará utilizzato come riferimento, in Appendice C é mostrato il codice di una classe significativa(Cnf.java) e la relativa documentazione, infine in Appendice D viene introdotta una piccola presentazione dell'architettura CORBA, utilizzata all'interno del sistema MOMIS.

# Capitolo 1

# L'Integrazione delle Informazioni

I principali aspetti teorici da considerare nell'integrazione delle informazioni sono legati alla eterogeneità dei dati, che si manifesta sia nella natura dei dati (es. testi, immagini, suoni, record, ...) sia nei diversi tipi di sorgenti nei quali tali informazioni sono contenute(es. pagine HTML, DBMS relazionali o ad oggetti, file system, etc . . . ). Gli standard esistenti (TCP/IP, ODBC, OLE, CORBA, SQL, etc...), pur risolvendo anche se solo parzialmente, i problemi legati alle diversità hardware e software dei protocolli di rete e i problemi relativi alle comunicazioni fra i moduli, non trattano le problematiche specifiche connesse con la modellazione delle informazioni. Infatti, i modelli di dati e gli schemi si differenziano gli uni dagli altri in modo da dare una struttura logica ai numerosi generi di dati da memorizzare: in questo modo si crea una eterogeneità *semantica* non risolvibile dagli standard. Inoltre, l'abbondanza di informazioni, dovuta ad un numero sempre maggiore di fonti, contribuisce a far sorgere ulteriori problematiche: tra queste é possibile evidenziare l'*information overload* (sovraccarico delle informazioni), che consiste nella difficoltà da parte dell'utente di discerne e di isolare i dati significativi. Altri problemi per i quali si devono trovare soluzioni sono la riduzione dei tempi di accesso, la salvaguardia della sicurezza e della consistenza, gli elevati costi di mantenimento da affrontare per modificare, eliminare o introdurre una nuova sorgente.

I problemi elencati sono a testimonianza della complessità e della molteplicità degli aspetti che le architetture dedicate all'integrazione di sorgenti di dati eterogenei devono tenere in considerazione. Per facilitare il processo di progettazione e di realizzazione di tali moduli, che devono necessariamente essere affidabili, flessibili e tali da far fronte efficacemente ad un panorama in continua evoluzione, occorre sviluppare un'architettura di moduli che assicuri il riuso e la capacità di interagire con altri sistemi esistenti.

Gli approcci all'integrazione, descritti in letteratura o effettivamente realizzati, presentano diverse metodologie: la reingegnerizzazione delle sorgenti mediante

standardizzazione degli schemi e la creazione di un database distribuito; il *repository independence*, un approccio che prevede di isolare al di sotto di una vista integrata, le applicazioni ed i dati integrati dalle sorgenti, consentendo la massima autonomia e nascondendo al contempo le differenze esistenti; i *datawarehouse*, che realizzano presso l'utente finale delle viste, ovvero delle porzioni di sorgenti, replicando fisicamente i dati ed affidandosi a complicati algoritmi di 'allineamento' per assicurare la loro consistenza a fronte di modifiche nelle sorgenti originali. Nelle pagine seguenti verrà descritta la proposta dell'ARPA (Advanced Research Projects Agency) [27] per un'architettura che favorisca, da una parte l'autonomia delle sorgenti, ma che dall'altra assicuri flessibilità e riusabilità. Si presenterà inoltre l'approccio seguito nel progetto MOMIS.

## 1.1 L'Integrazione Intelligente delle Informazioni

L'integrazione delle Informazioni ($I^2$) si distingue da quella dei dati e dei database in quanto non cerca di collegare semplicemente alcune sorgenti, quanto piuttosto risultati opportunamente selezionati da esse [1]. Per ottenere risultati selezionati è richiesta *conoscenza* ed *intelligenza* finalizzate alla scelta delle sorgenti e dei dati, alla loro fusione e alla conseguente sintesi. L'integrazione comporta perciò grandi difficoltà, sia a livello teorico sia a livello pratico. La dimensione e la quantità delle problematiche che insorgono qualora si desideri fondere informazioni provenienti da sorgenti autonome, fanno sì che si senta l'esigenza di ideare una metodologia che sia riusabile, per diminuire i costi di sviluppo di tali applicazioni e che sia flessibile, per far fronte intelligentemente alle evoluzioni fisiche e logiche delle sorgenti interessate.

### 1.1.1 Il Programma $I^3$

Un'ambiziosa ricerca, finalizzata ad indicare un'architettura di riferimento che realizzi l'integrazione di sorgenti eterogenee in maniera automatica, è quella del programma $I^3$, sviluppato dall'ARPA, l'agenzia che fa capo al Dipartimento di Difesa americano [27].

In quel contesto si é potuto osservare che l'integrazione aumenta il valore dell'informazione ma richiede una forte adattabilità realizzativa: si devono infatti riuscire a gestire i casi di aggiornamento e sostituzione delle sorgenti, dei loro ambienti e/o piattaforme, della loro ontologia e della semantica. Le tecniche sviluppate dall'*Intelligenza Artificiale*, potendo efficacemente dedurre informazioni utili dagli schemi delle sorgenti, diventano pertanto uno strumento prezioso per la costruzione automatica di soluzioni integrate flessibili e riusabili.

Progettare la costruzione di supersistemi ad hoc che interessino una grande quantità di sorgenti non correlate semanticamente, è faticoso ed il risultato è un sistema scarsamente manutenibile o adattabile, strettamente finalizzato alla risoluzione dei problemi per cui è stato implementato.

Secondo il programma $I^3$, una soluzione a questi problemi può essere trovata mediante l'introduzione di architetture modulari sviluppabili secondo i principi proposti da uno standard. Tale standard deve porre le basi dei servizi che devono essere realizzati attraverso l'integrazione e deve cercare di abbassare i costi di sviluppo e di mantenimento.

Pertanto, costruire nuovi sistemi risulta realizzabile con minor difficoltà e minor tempo (e quindi costi), se si riesce a supportare lo sviluppo delle applicazioni riutilizzando la tecnologia già sviluppata. Per la riusabilità è fondamentale l'esistenza di interfacce ed architetture standard. Il paradigma suggerito per la suddivisione dei servizi e delle risorse nei diversi moduli, si articola su due dimensioni:

- *orizzontalmente* in tre livelli: livello utente, livello intermedio che fa uso di tecniche di IA, livello delle sorgenti di dati;
- *verticalmente*: diversi domini in cui raggruppare le sorgenti.

In generale i diversi domini non sono strettamente connessi all'interno di un certo livello ma si scambiano informazioni e dati; la combinazione delle informazioni avviene a livello di utilizzatore riducendo la complessità totale del sistema e consentendo lo sviluppo di numerose applicazioni finalizzate a scopi anche diversi fra loro. Ad esempio, si supponga di dover integrare informazioni sui trasporti mercantili, ferroviari e stradali: ciò permette all'utente di avere un'idea completa su quale mezzo di trasporto sia maggiormente vantaggioso per le proprie necessità. Aggiungendo a questo altri domini, quali le situazioni metereologiche ed i costi di immagazzinamento e trasporto, si possono facilitare le scelte di un dirigente che deve decidere come e quando consegnare delle merci.

Il livello dell'architettura su cui é necessario focalizzare maggiormente l'attenzione per quello che concerne l'uso di tecniche di intelligenza artificiale è quello intermedio: esso costituisce il punto nodale fra le applicazioni sviluppate per gli utenti ed i dati posti nelle sorgenti. Questo livello deve offrire servizi dinamici quali la selezione delle sorgenti, le gestione degli accessi e delle interrogazioni, il ricevimento e la combinazione dei dati, l'analisi e la sintesi degli stessi.

## 1.2 Architettura di riferimento per sistemi $I^3$

L'obiettivo del programma $I^3$ é di ridurre il tempo necessario alla realizzazione di un integratore di informazioni, fornendo una raccolta e una formalizzazione delle

soluzioni prevalenti nel campo della ricerca. In particolare due sono le ipotesi che rappresentano la base del progetto $I^3$ . La prima é connessa con la difficoltà che si ha nel ricercare delle informazione all'interno della molteplicità delle sorgenti di informazione che in questo momento é possibile individuare. Il secondo aspetto é legato al fatto che le fonti di informazione e i sistemi informativi, pur essendo spesso semanticamente correlati tra di loro, non lo sono in una forma semplice né preordinata. Di conseguenza il processo di integrazione delle informazioni puó risultare molto complesso.

Pertanto é quindi necessario proporre una architettura di riferimento che rappresenti alcuni dei servizi che un integratore di informazioni deve contenere e le possibili interconnessioni fra di essi. Tale descrizione non vuole imporre né delle soluzioni implementative, né é da ritenersi esaustiva delle funzionalità che un sistema di integrazione deve includere.

### 1.2.1 A cosa serve la tecnologia $I^3$ e quali problemi deve risolvere

Le applicazioni che possono essere interessate da sviluppi della tecnologia $I^3$ coprono un ampio spettro di campi e possono essere, puramente a titolo esemplificativo, suddivise in questo modo:

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

Naturalmente, avendo l'architettura proposta la presunzione di esprimere delle caratteristiche generali, ed essendo la casistica dei campi applicativi vasta, é sicuramente necessario individuare, al di fuori di un insieme di servizi di base, funzionalità specifiche di ogni ambiente di sviluppo. Ad esempio, un integratore il cui scopo sia interagire con sistemi di basi di dati "classici", come ad esempio i sistemi basati sui file, i data base relazionali, i data base ad oggetti, avrà la necessità di un pacchetto base di servizi molto differenti da un sistema cosiddetto "multimediale", il cui obiettivo sia integrare suoni, immagini . . .

Così come possono essere individuati differenti obiettivi per un sistema $I^3$ , allo stesso modo possono essere individuati diversi problemi ai quali é necessario dare risposta. Tra di questi, si propongono i seguenti:

- *la grande differenza tra le fonti di informazione*:
  - le fonti informative sono semanticamente differenti, e se ne possono individuare i differenti livelli [31];
  - le informazioni possono essere memorizzate utilizzando diversi formati;
  - possono essere differenti gli schemi, i vocabolari usati, le ontologie su cui questi si basano, anche quando le fonti condividono significative relazioni semantiche;
  - può variare la natura dell'informazione, che può essere rappresentata attraverso testi, immagini, audio, media digitali;
  - può variare il modo in cui si accede alle singole sorgenti. Quindi possono essere presenti molteplici interfacce utente, linguaggi di interrogazione, protocolli e meccanismi di transazione;
- *la semantica complessa ed a volte nascosta delle fonti*: i programmi applicativi possono rappresentare, ad esempio per vecchi sistemi, la chiave per l'uso di informazioni. Senza di essi può essere molto difficile dedurre la semantica contenuta nel database, specialmente nel caso in cui occorra interagire con sistemi molto vasti;
- *l'esigenza di creare applicazioni in grado di interfacciarsi con porzioni diverse delle fonti di informazione*: non è sempre possibile avere a disposizione l'intera sorgente di informazione, ma talvolta si dispone unicamente di una sua parte selezionata che può variare nel tempo;
- *il grande numero di fonti da integrare*: con il moltiplicarsi delle informazioni, lo stesso numero di fonti da integrare per ogni singola applicazione può aumentare in maniera considerevole e può succedere che sia necessario accedere in modo coordinato a diverse fonti;
- *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 progettare 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 di un sistema $I^3$ , riportata in Figura 1.1.

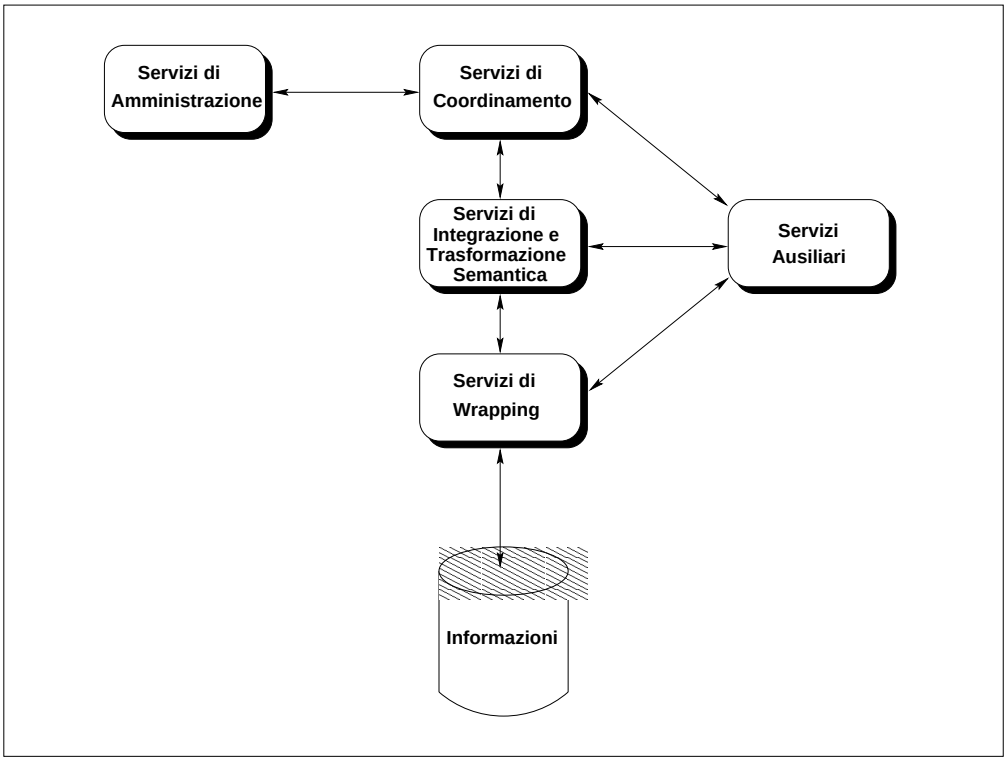


Figura 1.1: Diagramma dei servizi $I^3$

L'architettura di riferimento dà grande rilevanza ai Servizi di Coordinamento, che giocano due ruoli: possono localizzare altri servizi $I^3$ e fonti di informazioni che possono essere utilizzate per costruire il sistema stesso; sono responsabili di individuare ed invocare a run-time gli altri servizi necessari a dare risposta ad una specifica richiesta di dati.
Le famiglie di servizi rappresentate in questa architettura sono cinque. I servizi individuati, così come rappresentati nell'architettura proposta, possono essere soggetti a due differenti chiavi di lettura: quella che seguendo l'asse orizzontale e quella che segue l'asse verticale mettono in evidenza diversi aspetti e compiti del sistema.

Se si percorre l'asse verticale, si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i servizi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti. Tali informazioni vengono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passate 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.

Analizziamo ora in dettaglio le funzionalità specifiche di ogni servizio e le problematiche che devono essere affrontate.

### 1.2.2 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, possono essere rappresentati da meccanismi che includono dalla selezione dinamica delle sorgenti (o brokering, per Integratori Intelligenti) fino 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 possono essere uno alla volta (nel qual caso si parla di Brokering), oppure più di uno (e in questo 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 le risposte vengono presentate 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, ma viene fornita un'unica rappresentazione delle sorgenti e in questo modo un sistema omogeneo che gestisce direttamente la sua richiesta. Pertanto l'utente non é obbligato a conoscere i domini con i quali i vari moduli $I^3$ si trovano ad interagire. Risulta evidente la considerabile diminuzione di complessità di interazione col sistema che deriva da una architettura di questo tipo.

2. **Matchmaking**: il sistema viene configurato manualmente da un operatore in fase di inizializzazione. Successivamente a quella fase, tutte le richieste

verranno 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.

### 1.2.3 Servizi di Amministrazione

Si tratta di 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 all'uso dei Template, possono essere utilizzate le **Yellow Pages**: si tratta di servizi di directory che memorizzano le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata). Consultando queste Yellow Pages, il mediatore è 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: si 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 all'utente. Inoltre tale servizio potrebbe contenere dei servizi Trasformazione del Vocabolario e dell'Ontologia, come pure di Integrazione Semantica.
Da citare sono pure i servizi di Iterative Query Formulation: si tratta di sistemi che aiutano l'utente a rilassare o a meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

### 1.2.4 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 é rappresentato da una o più sorgenti di dati, e l'output é 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. Sono spesso indicati come servizi di Mediazione, essendo tipici dei moduli mediatori.
In particolare é possibile osservare:

1. Servizi di **integrazione degli schemi**. Supportano la trasformazione e l'in-

tegrazione degli schemi e delle conoscenze derivanti da fonti di dati eterogenee.Ne fanno parte 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 condivide un'unica ontologia. Fondamentale, per la fase di creazione dell'insieme dei 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, e nel momento in cui i risultati provenienti dalle singole subquery 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).

### 1.2.5 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 e viceversa quando si interroga la sorgente di dati. 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 sono 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 da altre fonti.

### 1.2.6 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.

## 1.3 Il mediatore

Secondo la definizione proposta da Wiederhold in [28] "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:

- assicurare un servizio stabile, anche nel caso di cambiamento delle 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.

Il progetto MOMIS, di cui questa tesi fa parte, ha tra i suoi obiettivi la realizzazione di un Mediatore. In corso di progettazione e realizzazione del sistema, é stato necessario introdurre un'ipotesi che restringesse il campo applicativo del sistema stesso (e di conseguenza che restringesse il campo dei problemi a cui dare risposta). Tale ipotesi consiste nel fatto che il mediatore, per il momento, si occupi esclusivamente di sorgenti di dati di testo strutturati e semistrutturati, quali possono essere basi di dati relazionali, ad oggetti, file di testo, pagine html e pagine XML. L'approccio architetturale scelto è quello *classico*, che si sviluppa principalmente su 3 livelli:

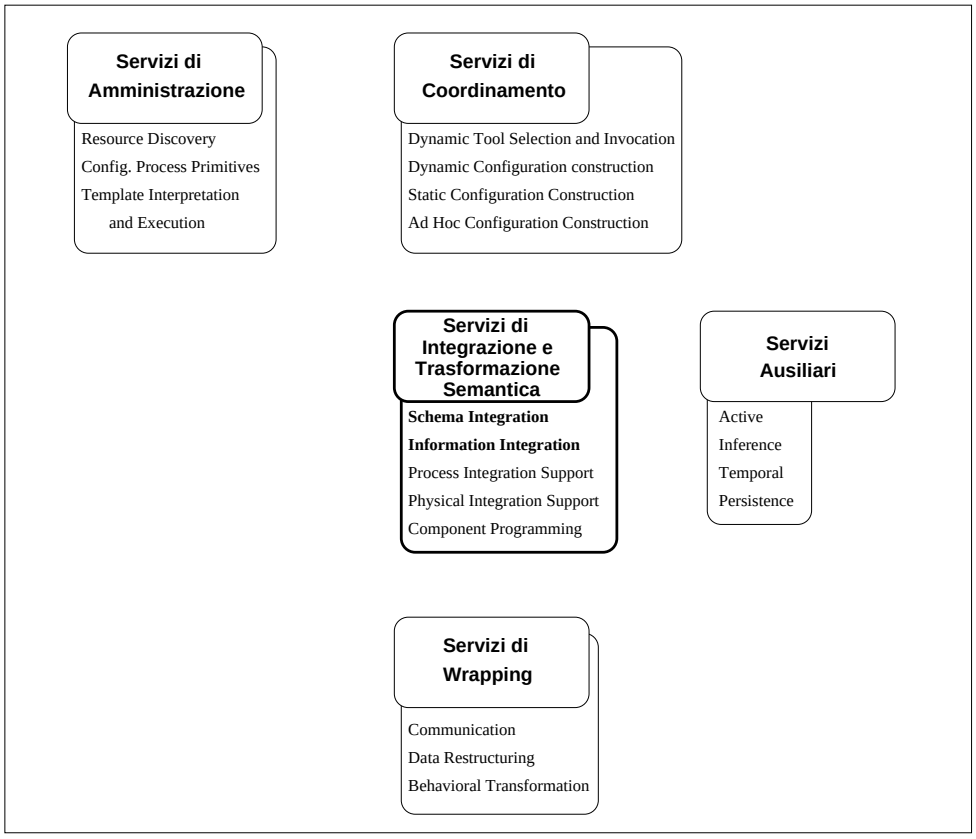


Figura 1.2: Servizi $I^3$ presenti nel mediatore

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 ed ha una duplice funzione: da un lato converte le richieste del mediatore in una forma comprensibile dalla sorgente, dall'altro traduce informazioni estratte dalla sorgente 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, é stato progettato un software che realizzi i servizi di Wrapping per quello che concerne le sorgenti dati di tipo XML, in particolare, come si é osservato in precedenza, il wrapper deve essere un'interfaccia fra il nucleo del mediatore, che fornisce le operazioni di integrazione e trasformazione semantica e la singola sorgente. Nella fattispecie il wrapper realizza due funzionalità specifiche:

1. restituisce la descrizione in $\text{ODL}_{I^3}$ della sorgente alla quale é connesso
2. permette l'esecuzione a livello locale delle query generate dal query manager.

L'impostazione architetturale presentata dimostra che il mediatore in progettazione vuole distaccarsi dall'approccio *strutturale*, cioè sintattico, tuttora dominante tra i sistemi presenti sul mercato. L'approccio strutturale, adottato da sistemi quali TSIMMIS [6, 7, 9, 10], è caratterizzato dall'uso di un self-describing model per rappresentare gli oggetti da integrare, limitando l'uso delle informazioni semantiche alle 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 ogni volta, per ogni suo singolo campo, il significato che ad esso è associato. Questo approccio porta 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 quello proposto, seguono invece un approccio definito *semantico*, che è caratterizzato dai seguenti 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 opportunamente le informazioni semantiche che necessariamente ogni schema sottintende, il mediatore può individuare concetti comuni a più sorgenti e relazioni che li legano.

### 1.3.1 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 e le relazioni che possono legarli, né tantomeno è semplice realizzare una loro coerente integrazione. Trascurando per il momento le differenze dei sistemi fisici (alle quali dovrebbero provvedere i moduli wrapper) i problemi che si sono dovuti 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.

### 1.3.2 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 ci si limita a riportare una semplice classificazione delle ontologie (mutuata da Guarino [29, 30]), per inquadrare l'ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate), sono essenzialmente tre:

1. *top-level ontology*: descrive 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*: descrive, 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*: descrive concetti che dipendono sia da un particolare dominio sia da un particolare obiettivo.

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

### 1.3.3 Problemi semantici

Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare e quindi un insieme di concetti comuni, niente testimonia che i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti e 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), un'altra persona P2 disegna la stessa fonte DB2, le due basi di dati avranno molto probabilmente 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 [31], la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che due persone distinte possono avere.Questa non è l'unica causa. 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 cioè 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.

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.

## 1.4 Il sistema MOMIS

Considerando le problematiche descritte nel paragrafo precedente, nonché alcuni sistemi preesistenti [9, 11, 12, 13, 14], si è giunti alla progettazione di un sistema intelligente di integrazione di informazioni da sorgenti di dati strutturati e semistrutturati denominato **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources).
Il contributo innovativo di questo progetto rispetto ad altri similari, risiede nella fase di analisi ed integrazione degli schemi sorgenti, realizzata in modo semiautomatico [17, 18, 23]. Un lavoro approfondito è stato svolto anche riguardo alla fase di *query processing* ([15, 16, 22]), cioè per il processo che dalla query posta sullo schema unificato provvede a generare automaticamente le sottoquery da inviare alle sorgenti ed ad integrare i risultati. MOMIS nasce all'interno del progetto MURST 40% INTERDATA dalla collaborazione tra i gruppi operativi dell'Università di Modena e Reggio Emilia e di quella di Milano.

### 1.4.1 L'approccio adottato

MOMIS adotta un approccio di integrazione delle sorgenti *semantico* e *virtuale* [15]. Il concetto di semantico è stato illustrato nella sezione 1.3. Con virtuale si intende invece, che la vista integrata delle sorgenti, rappresentata dallo schema globale, non viene *materializzata*, ma il sistema si basa sulla decomposizione delle query e sull'individuazione delle sorgenti da interrogare per generare delle subquery eseguibili localmente; lo schema globale dovrà inoltre disporre di tutte le informazioni atte alla fusione dei risultati ottenuti localmente per poter ottenere una risposta significativa.

Le motivazioni che hanno portato all'adozione di un approccio come quello descritto sono varie:

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

Inoltre, si è deciso di adottare un unico modello dei dati basato sul paradigma ad oggetti, che per la rappresentazione degli schemi sia per la formulazione delle interrogazioni. Il modello comune dei dati utilizzato nel sistema ($\text{ODM}_{I^3}$) è di alto livello e facilita la comunicazione tra il mediatore ed i wrapper. Per definire questo modello si è cercato di seguire le raccomandazioni relative alla proposta di standardizzazione per i linguaggi di mediazione, nata in ambito $I^3$: un mediatore deve poter essere in grado di gestire sorgenti dotate di formalismi complessi (ad es. quello ad oggetti), ed altre decisamente più semplici (come i file di strutture).È quindi preferibile l'adozione di un formalismo il più completo possibile.

Per la descrizione degli schemi si è arrivati a definire il linguaggio $\text{ODL}_{I^3}$, che si presenta come estensione del linguaggio standard ODL proposto dal gruppo di standardizzazione ODMG-93. Ció permette di cogliere le indicazioni emerse in ambito $I^3$ ed al contempo di discostarsi il meno possibile dalle proposte del gruppo appena citato. Un'applicazione di $\text{ODL}_{I^3}$ viene proposta in Appendice B nella stesura dell'esempio di riferimento. $\text{ODL}_{I^3}$ sarà spesso utilizzato in questa tesi, ma vista anche la sua natura intuitiva, non sarà ulteriormente descritto in

questa sede; una trattazione esauriente può comunque essere trovata in [16, 17, 18, 23].

Per quanto riguarda il linguaggio di interrogazione, si è adottato $OQL_{I^3}$ che applica la sintassi OQL senza discostarsi dallo standard. Questo linguaggio richiede un maggiore sforzo dal punto di vista dello sviluppo di moduli per l'interpretazione e la gestione delle interrogazioni (implementando le funzionalità tipiche di un ODBMS), ma risulta altresì estremamente versatile ed espressivo fornendo la possibiltà di sfruttare le informazioni rappresentate nello schema globale. Le maggiori difficoltà implementative sono quindi ampliamente giustificate da una maggiore versatilità e da una migliore facilità d'uso per l'utente finale.

Riassumendo: in MOMIS i wrapper si incaricano di tradurre in $ODL_{I^3}$ le descrizioni delle sorgenti, i moduli del mediatore di costruire lo schema globale, mentre tutte le interrogazioni sono poste dall'utente su questo schema utilizzando il linguaggio OQL, o meglio $OQL_{I^3}$, disinteressandosi dell'effettiva natura ed organizzazione delle sorgenti; sarà il sistema stesso che si incaricherà di tradurre le interrogazioni in un linguaggio comprensibile dalle singole sorgenti e di riorganizzare le risposte ottenute in una unica corretta e completa da fornire all'utilizzatore.

### 1.4.2 L'architettura generale di MOMIS

In Figura 1.3 è illustrata dettagliatamente l'architettura generale di MOMIS. Lo schema evidenzia l'organizzazione a tre livelli utilizzata.

**Livello Mediatore.** Il nucleo centrale del sistema è costituito dal **Mediatore** (o *Mediator*) che presiede all'esecuzione di diverse operazioni. Per meglio comprendere i suoi compiti, è opportuno a questo punto illustrare le due fasi ben distinte in cui si articola la sua attività.

La prima funzionalià del Mediatore è quella di generazione dello Schema Globale. In questa fase il modulo del Mediatore denominato **Global Schema Builder** riceve in input le descrizioni degli schemi locali delle sorgenti espressi in $ODL_{I^3}$ e forniti ognuno dal relativo wrapper. A questo punto, utilizzando strumenti di ausilio quali ODB-Tools Engine, WordNet, ARTEMIS, il Global Schema Builder è in grado di costruire la vista virtuale integrata (**Global Schema**), utilizzando tecniche di clustering e di Intelligenza Artificiale. In questa fase è prevista anche l'interazione con il progettista, il quale oltre ad inserire le regole di mapping, interviene nei processi che non possono essere svolti automaticamente dal sistema (come ad es. l'assegnamento dei nomi alle classi globali, la modifica di relazioni

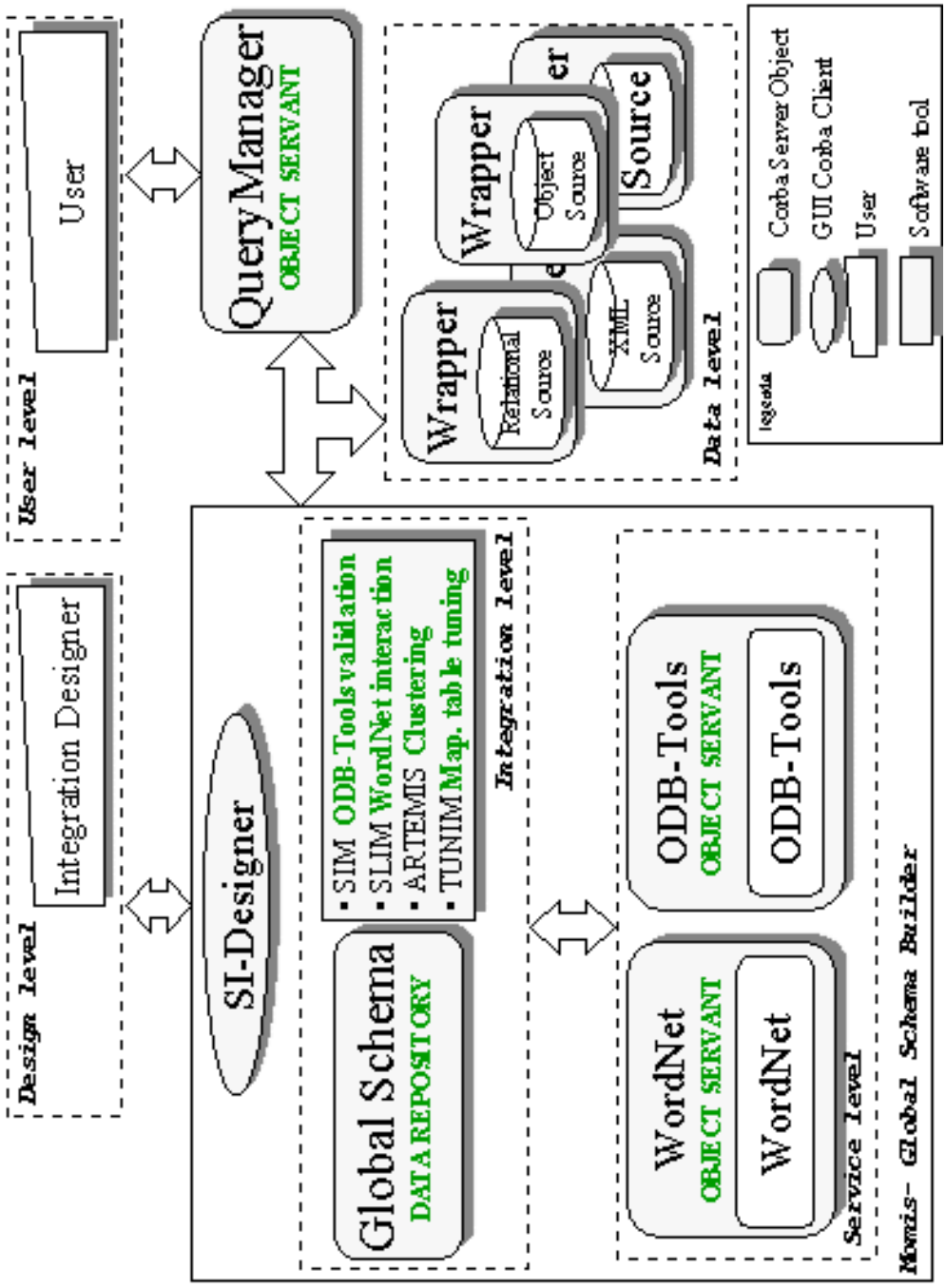


Figura 1.3: Architettura generale di MOMIS

lessicali, . . . ). Oltre allo Schema Globale, altri "prodotti" di questa fase sono le **Mapping Table**, tabelle che descrivono il modo in cui gli attributi globali presenti nello schema generato hanno corrispondenza (*mappano*), nei vari attributi locali presenti negli schemi delle sorgenti.

Un secondo importante modulo che compone la struttura del mediatore, è il **Query Manager** che presiede alla fase di query processing. In questa fase la singola query posta in $OQL_{I^3}$ dall'utente sullo Schema Globale (che chiameremo *Global Query*), sarà rielaborata in più *Local Query* (anche'esse espresse in $OQL_{I^3}$ ), da inviare alle varie sorgenti o meglio, ai wrapper predisposti alla loro traduzione. Questa traduzione avviene in maniera automatica da parte del Query Manager utilizzando tecniche di logica descrittiva.

L'ultimo modulo del Mediatore è rappresentato dall'**Extensional Hierarchy Builder** il quale si occupa della generazione della Conoscenza Estensionale (Gerarchie Estensionali e Base Extension) necessaria per ottimizzare le interrogazioni.La presente tesi tratta la progettazione e l'implementazione di questo modulo.

**Livello Wrapper.** I **Wrapper** costituiscono l'interfaccia tra il mediatore e le sorgenti; ad ogni sorgente corrisponde un determinato wrapper ed ogni wrapper deve essere disegnato esclusivamente per la sorgente (o la tipologia di sorgenti), che sovrintenderà. Ogni wrapper ha due compiti ben precisi:

- in fase di integrazione deve fornire al Global Schema Builder la descrizione della sorgente da integrare in formato $ODL_{I^3}$;
- in fase di query processing deve tradurre la local query (rivolta alla "sua" sorgente), che gli è stata indirizzata dal Query Manager (e che è espressa in $OQL_{I^3}$), nel linguaggio di interrogazione specifico della sorgente per la quale é stato progettato.

Ai wrapper sono quindi collegate le **Sorgenti**, per questo a volte si parla anche di quattro livelli. Esse sono le fonti da integrare, possono essere DataBase , ad oggetti o relazionali, o parti di essi, file system ed anche sorgenti semistrutturate.

**Livello Utente** L'utilizzatore del sistema dovrà potere interrogare lo schema globale. L'accesso ai dati si propone del tutto simile a come avvengono le usuali interrogazioni di un DataBase tradizionale: le sorgenti ed il modo in cui i dati vengono recuperati risultano all'utente del tutto trasparenti, in

quanto é il sistema ad occuparsi di ogni operazione necessaria per reperire le informazioni e combinare le risposte in un'unica risposta corretta, completa e non ridondante.

In precedenza si sono citati alcuni tool di ausilio per il mediatore, vediamo una loro rapida descrizione:

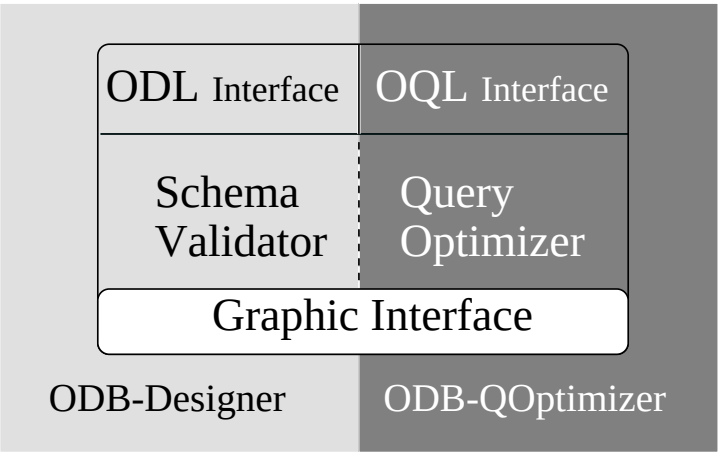


Figura 1.4: Architettura ODB-Tools

- *ODB-Tools* è uno strumento software sviluppato presso il dipartimento di Ingegneria dell'Università di Modena e Reggio Emilia [33, 34]. Esso si occupa della validazione di schemi e dell'ottimizzazione semantica di interrogazioni rivolte a Basi di Dati orientate agli Oggetti (OODB). Facciamo riferimento alla figura 1.4 per una breve spiegazione. ODB-Designer si occupa della validazione di schemi: si può inserire la descrizione di uno schema di DataBase (in ODL) ed il sistema realizzerà automaticamente la sua validazione e la sua riclassificazione: si occuperà cioé di verificare che non esistano classi incoerenti (classi che non possano essere popolate da nessun oggetto) e di determinare eventuali relazioni di specializzazione non esplicitate dallo schema stesso. ODB-Qoptimizer si occupa invece dell'ottimizzazione semantica delle interrogazioni: se si inserisce una query (in OQL) posta su di un determinato schema, questa viene automaticamente riformulata in una equivalente, ma più efficiente, sfruttando l'espansione semantica ed i vincoli di integrità.

- *WordNet* [37] è un DataBase lessicale on-line in lingua inglese. Esso è capace di individuare relazioni semantiche fra termini; cioè, dato un insieme di termini, WordNet è in grado di identificare l'insieme di relazioni lessicali che li legano.

- *ARTEMIS* [38] riceve in ingresso il *thesaurus*, cioè l'insieme delle relazioni terminologiche lessicali e strutturali, precedentemente generate, e sulla base di queste assegna ad ogni classe coinvolta nelle relazioni un coefficiente numerico indicante il suo grado di affinità. Questi coefficienti verranno utilizzati per raggruppare le classi locali in modo tale che ogni gruppo (*cluster*), comprenda solo classi con coefficienti di affinità simili.

# Capitolo 2

# La Conoscenza Estensionale in MOMIS

Uno dei principali obiettivi che ci si è preposti con MOMIS, è la realizzazione di un sistema di mediazione versatile ed efficiente, capace di assistere l'utente nel reperimento di informazione su di un insieme estremamente diversificato di sorgenti. Per riuscire in questo intento si è agito in due direzioni. Da un lato sono stati progettati strumenti in grado di assistere il progettista nella complessa fase di integrazione degli schemi, dall'altro si è realizzato un *Query Manager* che, posta un'interrogazione sulla vista globale, automatizza il processo di reperimento ed integrazione delle informazioni.
Per garantire che la risposta fornita sia al contempo corretta e quanto più possibile, completa e minima, entrambe queste fasi sfruttano un insieme di conoscenze relative alle sovrapposizioni delle estensioni delle classi, alle relazioni tra le intensioni e alla semantica degli schemi. Tale conoscenza è in parte fornita dal progettista ed in parte ricavata dalle informazioni implicitamente rappresentate negli schemi locali.

In questo capitolo si intende studiare l'integrazione tra gli schemi, ponendo particolare attenzione alle fasi nelle quali viene trattata la *conoscenza estensionale*.

## 2.1 Integrazione intensionale ed estensionale

Gli schemi locali vengono integrati in MOMIS secondo criteri sia *intensionali* che *estensionali*. I conflitti intensionali rappresentano quelle incompatibilitá derivanti dall'avere porzioni di schemi sovrapposte, ossia gli stessi aspetti del dominio applicativo vengono rappresentati usando strutture in parte differenti [8]. Ció

che occorre fare é quindi fornire una rappresentazione unificata ed omogenea dei medesimi concetti descritti in sorgenti differenti.

L'integrazione degli schemi non é peró l'unico aspetto che occorre gestire per ottenere un'effettiva integrazione di sorgenti eterogenee: é necessario risolvere anche i conflitti derivanti dalle sovrapposizioni delle estensioni, cioé dalla presenza, in sorgenti diverse, di informazione relativa alla stessa entitá del "mondo reale".

Il processo di integrazione viene effettuato in due momenti distinti, che sono rispettivamente a livello intensionale ed a livello estensionale:

1. **Unificazione degli schemi** [25]: in questa fase l'uso della logica descrittiva OLCD e di tecniche di *clustering* [40] permette di realizzare un processo semi-automatico di integrazione di schemi, fino a pervenire alla definizione dello Schema Globale, direttamente interrogabile dall'utente, che rappresenta l'unione di tutti gli schemi locali e dal quale vengono rimosse incongruenze e ridondanze;

2. **Fusione delle istanze**: vengono introdotti *assiomi estensionali* al fine di generare gli oggetti *virtuali* che rappresentano la fusione degli oggetti *reali* provenienti dalle sorgenti. In questa fase il progettista può introdurre conoscenza sulle relazioni fra le estensioni di classi simili in sorgenti diverse, questa sarà utilizzata successivamente (in fase di esecuzione delle interrogazioni) dal sistema stesso. La buona realizzazione di questa fase si manifesterà a run time in quanto influirà sulla correttezza e completezza delle risposte ottenute, ma anche sulla velocità con cui queste saranno generate, come verrá descritto nel paragrafo 2.3.

Insieme a questi due momenti, strettamente di integrazione, un sistema $I^3$ deve essere dotato di un componente **Query Manager** [15], che partendo da una query posta dall'utente sullo schema globale, determini l'insieme ottimo di interrogazioni da inviare alle singole sorgenti che contengono dati ed utili alla presentazione di un'unica risposta. In realtà la fase di *query processing* ha una frequenza di esecuzione diversa rispetto alle altre due: essa sarà avviata ogni volta che un utente pone un'interrogazione. Questo va di pari passo con il diverso *utilizzo temporale* dei moduli di MOMIS preposti all'esecuzione di ognuna di esse: il Global Schema Builder e l'Extensional Hierarchy Builder difatti vengono utilizzati una volta per tutte per creare la vista virtuale integrata, o al limite richiamati quando si decide di aggiungere o modificare alcune sorgenti o parti di esse, mentre il Query Manager, al contrario, sarà utilizzato *giornalmente* dagli utenti del sistema.

Per quanto riguarda invece strettamente il processo di integrazione, si è adottato un approccio semantico che si articola nei punti seguenti:

1. *Estrazione di relazioni terminologiche*. Durante questa fase, anche attraverso l'iterazione col progettista, viene generato un *Thesaurus Comune* di *relazioni terminologiche*; con questo termine si intendono quelle relazioni che esprimono la conoscenza inter-schema su sorgenti diverse, possono essere relazioni di sinonimia (SYN), di corrispondenza (RT), di maggiore (BT) o minore (NT) generalità del significato tra termini (nomi di classi, di attributi, ... ).
   Partendo dalle descrizioni delle sorgenti in $\text{ODL}_{I^3}$ queste relazioni vengono derivate in maniera semi-automatica attraverso l'analisi strutturale e di contesto delle classi coinvolte, grazie anche all'ausilio di ODB-Tools. Questo passo è realizzato dal modulo $\textbf{SIM}_1$ [18] e dal modulo **SLIM** [19] di MOMIS.

2. *Analisi delle affinità intensionali fra classi*. Questa fase è realizzata considerando le relazioni terminologiche memorizzate nel Thesaurus ed i coefficienti di affinità;

3. *Creazione dei cluster*. I *cluster* sono raggruppamenti di classi affini: si tratta di classi intensionalmente affini per le quali si presume esista anche una qualche sovrapposizione tra le estensioni. I passi 2 e 3 sono realizzati mediante ARTEMIS [38];

4. *Generazione dello schema globale del mediatore*. Da ogni cluster si definisce una *Classe Globale* che ha un'estensione, formata dall'unione delle estensioni delle classi sorgenti che costituiscono il cluster, ed un'intensione, ricavata dall'unione ragionata degli attributi delle stesse. Questa fase ha come risultato la definizione in $\text{ODL}_{I^3}$ delle Classi Globali con le relative regole di mapping fra i loro attributi *globali* e quelli *locali* delle classi delle sorgenti. Questo passo viene realizzato dal modulo TUNIM [21] di MOMIS.

Il processo fin qui descritto è quello presentato in [18, 25, 26], successivamente in [16] si sono studiate altre fasi relative alla gestione della conoscenza estensionale, che saranno trattate nel prosieguo di questo capitolo, e che estendono il precedente punto 4 attraverso:

- *Arricchimento dei cluster*. Il progettista arricchisce la descrizione dei vari cluster inserendo assiomi estensionali intra ed inter classi sorgente: può succedere di dovere modificare alcuni cluster qualora siano riconosciuti particolari legami estensionali fra classi in cluster diversi. Questo passo viene ripetuto finché il progettista non ottiene un insieme di cluster soddisfacente dal quale generare lo schema globale;

**Sorgente University** $(S_1)$

```
University_Worker(first_name,last_name,dept_code,pay)
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 2.1: Esempio di riferimento

- *Fusione della Gerarchia Virtuale*. Una volta fissati i cluster ed individuate di conseguenza le classi globali, ODB-Tools ne verifica la congruenza considerando gli assiomi estensionali e computa attraverso il suo sistema di inferenza gli assiomi impliciti. Ad ogni classe globale si applica un algoritmo di *individuazione delle Base Extension* e di *fusione delle gerarchie* [41]. Viene costruita una gerarchia composta da nuove classi, l'intensione delle quali è un sottoinsieme delle proprietà del cluster e la cui estensione è costituita da tutti gli oggetti reali e/o fusi provenienti dalle sorgenti che posseggono almeno le proprietà specificate nell'intensione della classe.

## 2.2 Esempio di riferimento

Il seguente esempio che verrà utilizzato per illustrare le fasi di integrazione fa riferimento alle definizioni degli schemi delle sorgenti espresse in $\text{ODL}_{I^3}$ e riportate in Appendice B. In Figura 2.1 viene invece presentato in modo schematico per maggiore semplicità.
Esso 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 univer-

sità. È composta da sei tabelle: `University_Worker`, `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 inoltre viene memorizzata l'aula (`Room`) dove questo si svolge, mentre del dipartimento sono descritti, oltre al nome (`dept_name`) ed al codice (`dept_code`), il budget (`budget`) che ha a disposizione e l'area (`dept_area`) a cui appartiene, sia essa Scientifica, Economica, ... Per gli studenti presenti nella tabella `School_Member` sono invece mantenuti il nome (nella coppia `first_name` e `last_name`), la facoltà di appartenenza (`faculty`) e l'anno di corso (`year`).

La sorgente `Computer_Science` ($S_2$) contiene invece informazioni sulle persone afferenti a questa facoltà, 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 infine una terza sorgente, `Tax_Position` ($S_3$), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (`tax_fee`). In quest'ultimo caso non si tratta di un DataBase ma di un file system, che contiene quindi semplici tracciati record.

## 2.3 Utilizzo della Conoscenza Estensionale

Per comprendere meglio le problematiche relative alla conoscenza estensionale occorre chiarire bene la distinzione tra *Oggetti* di un database ed *Entitá* del mondo reale. Studiando un particolare contesto applicativo possono essere individuate entitá caratterizzate da un determinato insieme di proprietá o attributi, esse esprimono concetti ben definiti del *"mondo reale"* ai quali peró possono essere associate molte rappresentazioni alternative. In particolare Database indipendenti forniranno modellazioni differenti, sia descrivendo con strutture diverse gli stessi attributi e soprattutto andranno a cogliere, in funzione delle loro finalitá ed obiettivi, aspetti differenti delle stesse entitá. In definitiva un entitá rappresenta un concetto astratto che prescinde da qualsiasi rappresentazione, mentre un ogget-

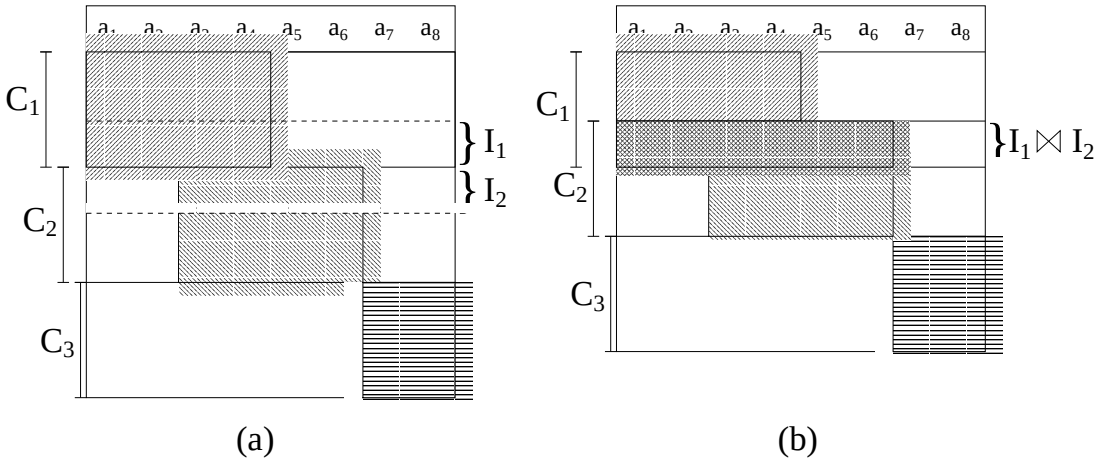


Figura 2.2: Estensione della classe di entità

to é uno strumento di modellazione che, utilizzando una particolare struttura, ne cattura determinati aspetti.

A questo punto risulta evidente che sorgenti autonome possono contenere oggetti corrispondenti alla stessa entitá ed ognuno di questi, in parte, replicherá proprietá descritte da altri oggetti ma potrá anche fornire un proprio contributo descrivendone di nuove. Pertanto l'obiettivo dell'integrazione deve essere non solo il reperimento dei singoli oggetti ma piuttosto la ricomposizione (*fusione*) dell'entitá a cui sono associati.

Supponiamo di voler gestire un dominio applicativo in cui è presente il concetto di "$persona$" (P) ed immaginiamo di avere tre classi locali ($C_1$, $C_2$ e $C_3$) contenenti informazioni relative a persone. Immaginiamo poi che la classe di entità sia caratterizzata dalle proprietà $a_1, \ldots, a_8$ e che gli schemi[1] delle classi locali siano parzialmente sovrapposti.

int(P) = $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$

int($C_1$) = $\{a_1, a_2, a_3, a_4\}$

int($C_2$) = $\{a_3, a_4, a_5, a_6\}$

int($C_3$) = $\{a_7, a_8\}$

[1]Con int(P) denotiamo l'intensione di P

Per arrivare ad un'integrazione corretta può non essere sufficiente fare una semplice unione delle estensioni delle classi, in quanto dati relativi alla stessa entità potrebbero essere presenti in più classi. L'unione, in questo caso, comporterebbe un errore nella rappresentazione, infatti, ogni oggetto nelle classi locali verrebbe considerato come un'istanza distinta della classe globale, Figura 2.2 (a), determinando una duplicazione di informazione (la stessa persona sarebbe presente più volte), e l'incompletezza della risposta ( le persone duplicate hanno un insieme incompleto di proprietà).
Questi inconvenienti possono essere risolti soltanto gestendo in modo corretto le relazioni tra le estensioni. Se, ad esempio, le estensioni di $C_1$ e $C_2$ risultano essere sovrapposte, un sottoinsieme delle loro istanze, pur avendo attributi diversi ed appartenendo a sorgenti differenti, individua le stesse entità del mondo reale. In Figura 2.2 (b) viene appunto mostrato come gli oggetti presenti nei sottoinsiemi $I_1$ e $I_2$ ed appartenenti alla sovrapposizione tra $C_1$ e $C_2$, debbano essere "$fusi$" per fornire un'informazione corretta e completa.

Il processo che porta alla effettiva integrazione estensionale puó essere descritto dai seguenti passi:

- Definizione degli *assiomi estensionali*
- Calcolo dell'insieme delle *Base Extension*
- Generazione della *Gerarchia Estensionale*.

### 2.3.1 Definizione degli assiomi estensionali

Gli assiomi estensionali descrivono le relazioni insiemistiche esistenti tra le estensioni delle sorgenti. Per chiarire meglio il concetto diamo alcune definizioni.

**Definizione 2.3.1 (Stato di una classe)** *Lo stato di una classe* **C***(dove C puó essere una espressione logica che coinvolge piú classi) all'istante t, scritto* $Stato^t_C$*, é costituito dall'insieme degli oggetti che popolano la classe* **C** *all'istante t. Lo stato di una classe viene spesso indicato come* l'estensione della classe.

**Definizione 2.3.2 (Assiomi estensionali)** *Date due classi* **A** *e* **B** *é possibile definire quattro tipi di assiomi estensionali, la cui validitá sussiste per ogni stato del database:*

- *Equivalenza:*$A \equiv B$: $\Leftrightarrow \forall t$: $Stato^t_A = Stato^t_B$
- *Inclusione:*$A \subset B$: $\Leftrightarrow \forall t$: $Stato^t_A \subseteq Stato^t_B$
- *Disgiunzione:*$A \oslash B$: $\Leftrightarrow \forall t$: $Stato^t_A \cap Stato^t_B = \oslash$

- *Sovrapposizione:*$A \cap B: \Leftrightarrow \forall t: \neg(A \equiv B) \wedge \neg(A \subset B) \wedge \neg(B \subset A) \wedge \neg(A \oslash B)$

L'analisi estensionale effettuata in MOMIS si basa su due presupposti:

1. Per classi appartenenti ad uno stesso cluster, per le quali non é stata specificata nessuna relazione, si assume che le loro estensioni siano sovrapposte, mentre, tra classi appartenenti a cluster diversi deve sussistere una relazione di disgiunzione. Questa ipotesi é stata introdotta perché ogni classe globale deve raccogliere tutte le informazioni relative ad un medesimo concetto del dominio applicativo, pertanto si presuppone che tutte le classi locali che compongono un cluster descrivano i medesimi concetti mentre non possono esistere entitá partizionate su cluster distinti. Se uno o piú assiomi violassero queste condizioni si renderebbe necessario rimuoverli, perché non corretti, o modificare i cluster.

2. Ogni assioma definito *vincola* le classi coinvolte ad avere anche un legame intensionale.
   Estendendo la notazione usata per le relazioni intensionali **NT, BT, SYN**, gli assiomi estensionali possono essere definiti in $ODL_{I^3}$ come:

   - **C1** $SYN_{Ext}$ **C2**: le istanze della classe C1 sono le stesse della classe C2. Questo implica una relazione intensionale di tipo SYN tra le due classi e due relazioni ISA.
   - **C1** $NT_{Ext}$ **C2**: le istanze della classe C1 sono un sottoinsieme di quelle della classe C2. Questo assioma viene scomposto in una relazione intensionale di tipo NT e una relazione ISA.
   - **C1** $BT_{Ext}$ **C2**: le istanze della classe C1 sono un sovrainsieme di quelle della classe C2. Viene generata una relazione intensionale di tipo BT ed una ISA tra le classi.
   - **C1** $DISJ_{Ext}$[2] **C2**: le istanze della classe C1 sono diverse da quelle della classe C2. Questo assioma non implica nessun tipo di relazione intensionale, esiste solamente un legame di tipo BOTTOM[3] tra le due classi coinvolte.

L'assioma che definisce la *sovrapposizione estensionale* non necessita di notazione in quanto, come giá specificato, viene considerato di default per le classi appartenenti allo stesso cluster. La definizione degli assiomi estensionali avviene,

[2]Questa notazione non era presente nella rappesentazione delle relazioni intensionali ma é stata introdotta per poter rappresentare il concetto di *disgiunzione estensionale*.

[3]Nella logica descrittiva un tipo o classe *bottom* rappresenta un concetto incongruente, cioé che non puó essere in nessun caso popolato da dati o istanze.

in buona parte, ad opera del progettista. Parte degli assiomi possono essere ricavati direttamente dalle definizioni degli schemi (es. relazioni di specializzazione tra classi), ma per le relazioni inter-schema l'intervento del progettista rimane fondamentale.

In $ODL_{I^3}$ una relazione "ISA" puó inoltre essere rappresentata dal seguente vincolo di integritá:

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

mentre una relazione di tipo "BOTTOM" viene espressa attraverso il vincolo:

```
rule Rule2 forall X in (C1 AND C2) then X in BOTTOM
```

Per come sono state definiti gli assiomi estensionali di tipo $SYN_{Ext}$, $NT_{Ext}$ e $BT_{Ext}$ si intuisce che il legame che generano tra le classi coinvolte é molto forte poiché implica sia un legame tra gli schemi che un legame tra le istanze. Per questo motivo, le classi legate da relazioni di questo tipo vengono forzate ad appartenere allo stesso cluster. In particolare, le relazioni intensionali logicamente implicate dagli assiomi estensionali $SYN_{Ext}, NT_{Ext}, BT_{Ext}$, vengono registrate nel *Common Thesaurus* e gli viene assegnato un peso di affinitá uguale a uno in modo da forzare le classi nel medesimo cluster.

La conoscenza estensionale viene utilizzata quindi per la definizione dei cluster e, come vedremo in seguito, anche per la costruzione dell'insieme delle *Base Extension* e della *Gerarchia Estensionale*.

### 2.3.2 Calcolo dell'insieme delle Base Extension

Il prerequisito fondamentale per il calcolo delle *base extension* e della gerarchia estensionale é che tra le sorgenti coinvolte siano risolti tutti i conflitti intensionali relativi ai nomi ed ai tipi degli attributi. In pratica, il procedimento di calcolo delle *base extension* puó essere applicato singolarmente ad ogni classe globale, cioé quando la composizione dei cluster e le *mapping table* sono ben delineate. Inoltre, per quanto detto in precedenza, la definizione di assiomi tra classi appartenenti a classi globali differenti potrebbe portare alla ridefinizione di alcuni cluster.

Riprendiamo l'esempio introdotto nella Sezione 2.2 e supponiamo che il processo di integrazione porti alla formazione delle classi globali illustrate in Figura 2.3[4](in Tabella 2.4 viene riportata la *mapping table* della classe globale **Person**[5]).

Cerchiamo di dare una definizione intuitiva di *base extension* per rendere il concetto il piú chiaro possibile.

[4]Per distinguere eventuali classi locali omonime é stata utilizzata la notazione *nome sorgente.nome classe*

[5]Per ragioni di spazio non sono mostrati tutti gli attributi

**Classe Globale Person** $(GC_1)$

```
University.University_Worker
University.Research_Staff
University.School_Member
Computer_Science.CS_Person
Computer_Science.Professor
Computer_Science.Student
Tax_Position.University_Student
```

**Classe Globale Workplace** $(GC_2)$

```
University.Department
Computer_Science.Division
```

**Classe Globale Course** $(GC_3)$

```
University.Section
Computer_Science.Course
```

**Classe Globale Location** $(GC_4)$

```
Computer_Science.Location
```

**Classe Globale Room** $(GC_5)$

```
University.Room
```

Figura 2.3: Classi globali generate

| GC1 | name | rank | faculty | year | relation | e_mail | tax_fee | ... |
|---|---|---|---|---|---|---|---|---|
| **Research_Staff** | first_name AND last_name | "Professor" | NULL | NULL | relation | e_mail | NULL | ... |
| **School_Member** | first_name AND last_name | "Student" | faculty | year | NULL | NULL | NULL | ... |
| **CS_Person** | name | NULL | "CS" | NULL | NULL | NULL | NULL | ... |
| **University_Student** | name | "Student" | faculty_name | NULL | NULL | NULL | tax_fee | ... |
| **Student** | name | rank | "CS" | year | NULL | NULL | NULL | ... |
| **Professor** | name | rank | "CS" | NULL | NULL | NULL | NULL | ... |
| **University_Worker** | first_name AND last_name | NULL | NULL | NULL | NULL | NULL | NULL | ... |

Figura 2.4: Mapping table della classe globale GC1

Supponiamo che per la classe globale $GC_1$ siano stati definiti gli assiomi estensionali :

1. `U.School_Member` $SYN_{Ext}$ `TP.University_Student`
2. `CS.Student` $NT_{Ext}$ `U.School_Member`
3. `U.Research_Staff` $NT_{Ext}$ `U.University_Worker`
4. `CS.Professor` $NT_{Ext}$ `U.Research_Staff`
5. `CS.Professor` $DISJ_{Ext}$ `U.School_Member`
6. `U.Research_Staff` $DISJ_{Extm}$ `TP.University_Student`
7. `U.Research_Staff` $DISJ_{Extm}$ `CS.Student`

   Analizzando la sorgente *Computer_Science*, dalle relazioni di ereditarietá, ricaviamo gli assiomi:

8. `CS.Student` $NT_{Ext}$ `CS.CS_Person`
9. `CS.Professor` $NT_{Ext}$ `CS.CS_Person`

In Figura 2.5[6] é schematizzata tutta la conoscenza estensionale appena espressa attraverso le rule. Le *base extension* sono indicate dai numeri 1, . . . ,12.

Una *base extension* rappresenta un sottoinsieme di entitá appartenenti ad uno stesso concetto del dominio applicativo. Presa quindi una classe di entitá, un insieme di *base extension* ne rappresenta il partizionamento in modo che ogni istanza appartenga ad una ed una sola di esse e che tutte le istanze di una stessa *base extension* abbiano lo stesso insieme di proprietá. Le *base extension* sono individuate dalle relazioni estensionali presenti tra le classi locali e sono caratterizzate

[6]Da notare che tra le classi per le quali non é stata definita nessuna rule si sottintende la sovrapposizione estensionale di default

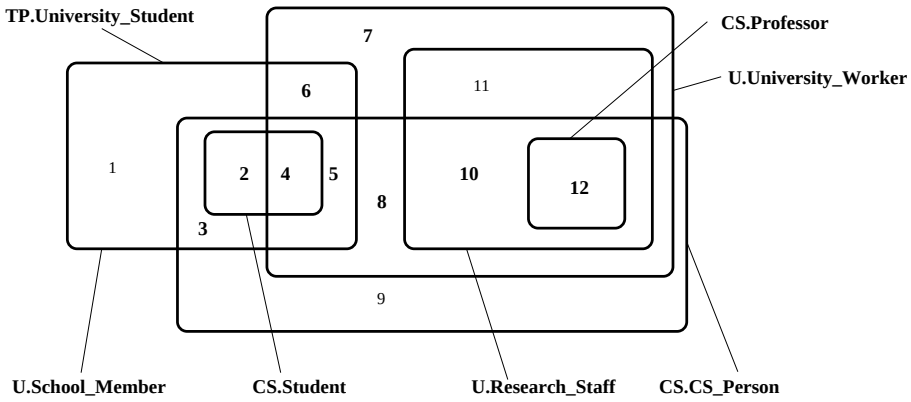


Figura 2.5: Relazioni estensionali per la classe $GC_1$

dall'avere l'estensione contenuta nell'insieme di entitá formate dall'intersezione delle classi che la compongono e come intensione l'unione dei loro schemi.

Uno degli scopi principali dell'introduzione del concetto di *base extension* e dovuto al fatto di poter considerare una singola istanza componente di una ed una sola *base extension* mentre se consideriamo le singole classi locali una generica istanza puó appartenere a piú di una classe(es. Le istanze della classe CS.Professor appartengono sia alla classe CS.Professor che alla classe U.Research_Staff e sono rappresentate dalla singola *base extension* 4).

L'insieme delle *base extension* puó avere anche una rappresentazione di tipo tabellare(Tabella 2.1) dove vengono messe in risalto le singole classi componenti. Le righe indicano le classi sorgenti e le colonne le base extension. Leggendo la matrice per colonne si ha rispettivamente un segno '1' o '0' se la classe, della riga corrispondente, comprende o meno nella propria estensione la base extension.
Una base extension rappresenta l'insieme intersezione delle estensioni di tutte le classi che, nella relativa colonna, sono contraddistinte da un segno '1' meno l'insieme unione di tutte le estensioni delle classi contraddistinte con un segno '0'. Leggendo per righe si deduce invece che l' estensione di una sorgente è data dall' unione delle base extension corrispondenti alle colonne per cui esiste una casella marcata dal segno '1'.

Occorre naturalmente prevedere un algoritmo che sia in grado di controllare la correttezza degli assiomi specificati dal progettista e di individuare eventuali incongruenze. Ad esempio, riconsiderando le *rule estensionali* rappresentate dalla Figura 2.5, l'inserimento dell'assioma :

$$\text{Computer\_Science.Professor } NT_{Extm} \text{ Computer\_Science.Student,}$$

Porterebbe ad un evidente incongruenza con gli assiomi specificati in precedenza. In questo caso il sistema, non solo deve essere in grado di rilevare l'inconsistenza,

| Base Ext. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| U.University_Worker | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| TP.University_Student | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| U.School_Member | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| CS.CS_Person | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| CS.Student | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CS.Professor | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| U.Research_Staff | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

Tabella 2.1: Tabella delle Base Extension

ma deve anche segnalare quali sono le *rule* coinvolte in modo da facilitare il lavoro del progettista, soprattutto in quei casi in cui il numero degli assiomi inseriti é elevato. La metodologia adottata per il controllo della congruenza verrá illustrata nel capitolo 3.

Una volta individuato un insieme corretto di *base extension*, occorre ricostruirne l'intensione e l'estensione.

- L'estensione, di ogni *base extension*, é contenuta nell'intersezione delle estensioni delle classi locali che la compongono (si ricava dalla tabella 2.1).

- L'intensione di ogni *base extension* é formata dall'unione degli attributi **globali** che vengono mappati[7] dalle classi locali che formano la *base extension*. Questo perché gli attributi di query delle interrogazioni inviate dall'utente saranno di tipo globale e attraverso le *base extension* si riuscirá a individuare le classi locali target.

Data la *mapping table* della classe globale si ricava una tabella (vedi Tabella 2.2) dove vengono individuati gli attributi globali che hanno un mapping non *NULL* sulle classi componenti. In seguito sovrapponendo questa tabella a quella delle *base extension* (Tabella 2.1) si ottiene un ulteriore tabella (Tabella 2.3) che specifica l'intensione di ogni *base extension*.

Tutte le informazioni relative all'intensione vengono impiegate per la generazione della *gerarchia estensionale*, mentre quelle che riguardano la composizione dell'estensione vanno ad arricchire la rappresentazione globale fornita dal Mediatore. Deve essere presente infatti, una struttura dati permanente che affiancata alla *mapping table*, indichi "quali" sono le classi locali e "come" devono essere combinate per ricomporre le entitá descritte.

[7] Che hanno un mapping non NULL

| Attributi | name | rank | faculty | year | relation | e_mail | tax_fee | ... |
|---|---|---|---|---|---|---|---|---|
| **University_Worker** | X | | | | | | | ... |
| **Research_Staff** | X | X | | | X | X | | ... |
| **School_Member** | X | X | X | X | | | | ... |
| **CS_Person** | X | | X | | | | | ... |
| **Univ_Student** | X | X | X | | | | X | ... |
| **Student** | X | X | X | X | | | | ... |
| **Professor** | X | X | X | | | | | ... |

Tabella 2.2: Proprietá intensionali della classi locali di GC1(Intensional Overlapping)

| Base Extension | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **name** | X | X | X | X | X | X | X | X | X | X | X | X |
| **rank** | X | X | X | X | X | X | | | | X | X | X |
| **faculty** | X | X | X | X | X | X | | X | X | X | | X |
| **year** | X | X | X | X | X | X | | | | | | |
| **relation** | | | | | | | | | | X | X | X |
| **e_mail** | | | | | | | | | | X | X | X |
| **tax_fee** | X | X | X | X | X | X | | | | | | |
| **works** | | | | X | X | X | X | X | | X | X | X |
| **title** | | | | | | | | | | | | X |
| **takes** | | X | | X | | | | | | X | X | X |
| **student_code** | X | X | X | X | X | X | | | | | | |
| **pay** | | | | X | X | X | X | X | | X | X | X |

Tabella 2.3: Intensione delle base extension(Extension-Attribute Relation)

É necessario prevedere una struttura che memorizzi quali operazioni di join devono essere eseguite, in quale ordine e quali attributi di collegamento occorre utilizzare.

### 2.3.3 Generazione della Gerarchia Estensionale

Ad ogni classe globale viene associata una gerarchia estensionale. Queste gerarchie sono costruite a partire dalle informazioni relative alle *base extension*.

Utilizzando un algoritmo che verrá descritto nel Capitolo 3, viene generata una gerarchia formata da classi virtuali totalmente nuove. Lo scopo principale é quello di individuare le relazioni di specializzazione che legano tra loro le intensioni delle varie *base extension* di una classe globale. In questo modo, partendo da una query posta dall'utente e quindi da una serie di attributi globali, attraverso la gerarchia estensionale, si riesce a ricavare l'insieme ottimo di *base extension* alle quali inviare la query. Si ricava in questo modo un insieme di istanze che devono essere combinate attraverso determinate chiavi di join per arrivare a ricostruire le entitá descritte dalle varie sorgenti.

La gerarchia estensionale permette il passaggio da una informazione di tipo intensionale ad una di tipo estensionale. Le classi virtuali vengono organizzate in una gerarchia di ereditarietá in modo da ottimizzare al massimo la procedura di ricerca della classe virtuale target.

Tutte le classi virtuali che compongono la gerarchia sono caratterizzate dall'avere :

- Intensioni corrispondenti alla intersezione degli schemi delle *base extension* presenti nella classe virtuale. In questo modo si individua un insieme di intensioni sovrapposte, che rappresentano le descrizioni di tutti i tipi di entitá[8] che sono descritte all'interno della classe globale. Per ogni intensione esisterá almeno un tipo di entitá, appartenente alla classe globale, caratterizzata da tutti e soli gli attributi dell'intensione stessa e non sono previsti tipi di entitá che non siano associati a nessuna intensione.

- Estensione data dall'unione di tutte le *base extension* che hanno almeno tutti gli attributi presenti nell'intensione della classe virtuale.

In Figura 2.6 é presentata la gerarchia estensionale associata alla classe globale Person (GC1). La classe virtuale C1 ha come estensione tutte le *base extension*, contiene quindi tutte le istanze descritte dalla classe globale e l'attributo **name**

[8]Si parla di tipi di entitá e non di entitá, perché tutto quello che riguarda la descrizione estensionale esprime condizioni di potenziale esistenza, quindi individuare un insieme di entitá non significa che sia necessariamente popolato.

(intensione della classe virtuale) rappresenta il solo attributo comune a tutte la *base extension*. Mentre le classi virtuali pi'ĭn alto della gerarchia sono in genere formate da l'unione di piú classi, scendendo le classi virtuali sono ottenute intersecando le classi locali, per questo motivo cala il numero delle *base extension* riducendosi il numero di istanze appartenenti a tutte le classi sovrapposte.
Esistono classi virtuali che non aggiungono ulteriori attributi di specializzazione, cioé la loro intensione é formata da tutti e soli gli attributi delle relative superclassi ma sono popolate da un minor numero di istanze, questo perché aumentando il numero di classi locali intersecate (che non introducono ulteriori nuovi attributi globali) decresce il numero di istanze comune a tutte la classi.

Facendo riferimento allo schema delle relazioni estensionali in Figura 2.5 si riescono a individuare le classi locali che vengono ricostruite in base alla estensione di ogni classe virtuale, precisamente :

- $C1 = TOP$[9]
- $C2 = \{U.School\_Member \cup U.Research\_Staff\}$
- $C3 = \{U.School\_Member \cup CS.CS\_Person\}$
- $C4 = \{U.School\_Member\}$
- $C5 = \{U.University\_Worker\}$
- $C6 = \{CS.Student \cup U.Research\_Staff\}$
- $C7 = \{CS.CS\_Person \cap U.University\_Worker\}$
- $C8 = \{U.University\_Worker \cap U.School\_Member\}$
- $C9 = \{U.Research\_Staff \cap CS.Person\}$
- $C10 = \{U.Research\_Staff\}$
- $C11 = \{CS.Student\}$
- $C12 = \{CS.Student \cap U.University\_Worker\}$
- $C13 = \{CS.Professor\}$

[9]Contiene tutte le istanze descritte nella classe globale

Alcune di queste classi (come C1), hanno in realtà un'estensione non propria, ma determinata dall'unione insiemistica delle estensioni delle sottoclassi (perchè non esistono oggetti che abbiano solo gli attributi che caratterizzano l' intensione di quella particolare classe); altre (come C7, C8, C9, C11, C12), non hanno attributi propri ed ereditano solo quelli delle superclassi dirette, però hanno un' estensione propria caratterizzata dagli oggetti che rappresentano l'intersezione delle superclassi. Queste due tipologie di classi possono essere eliminate o mantenute dallo schema in base a considerazioni sulla comprensibilità dello schema e sulla capacità del sistema di gestire oggetti in più di una classe (gli oggetti memorizzati in C8, se questa fosse eliminata, sarebbero sia in C6 che in C7: conserviamo C8 se il sistema non gestisce il caso di avere uno stesso oggetto in più classi).

Se in fase di *Query processing* l'utente dovesse inviare una query sull'attributo globale **title**, grazie alla gerarchia estensionale é possibile individuare la classe virtuale target, che in questo caso é la classe C13. La query peró, non deve essere inviata solamente alla classe locale "Professor" poiché si otterrebbe come risultato la ricostruzione delle istanze di "Professor" non dell'entit'a, ma anche a tutte le classi che compongono l'estensione delle classi virtuali dalle quali la classe C10 ereditá (nel nostro caso devo inviare la query anche alla classi "University_Worker","School_Member","Research_Staff"), in modo da recuperare tutte le informazioni possibili che ricombinate, utilizzando opportune chiavi di join, andranno a ricostruire l'entitá cercata.

Queste gerarchie di classi virtuali devono essere memorizzate ed aggiunte, assieme alle descrizioni delle *base extension*, alle informazioni sullo schema globale fornito dal Mediatore.

Una volta costruita tutta la conoscenza necessaria é possibile determinare le classi locali coinvolte in una interrogazione. Ottenute le istanze in risposta alla query occorre disporre delle informazioni per "fondere" le istanze che si riferiscono alla stessa entitá del dominio applicativo. Senza tali informazioni non saremmo in grado di ricostruire una risposta corretta e completa.

Questa ulteriore conoscenza non puó essere generata in modo automatico, pertanto diventa fondamentale l'intervento del progettista durante tutta il processo di integrazione. In particolare si dovrá studiare un algoritmo che preveda, a partire dalle classi locali che compongono una generica *base extension* permette di:

- trovare per ogni *base extension*, una o piú chiavi semantiche, cioé insiemi di attributi che individuano in modo univoco le entitá;
- determinare quali operazioni di join possono essere eseguite tra le classi locali che compongono la *base extension*. Individuare, se possibile, per ogni coppia di classi una chiave comune che ne permetta il join;

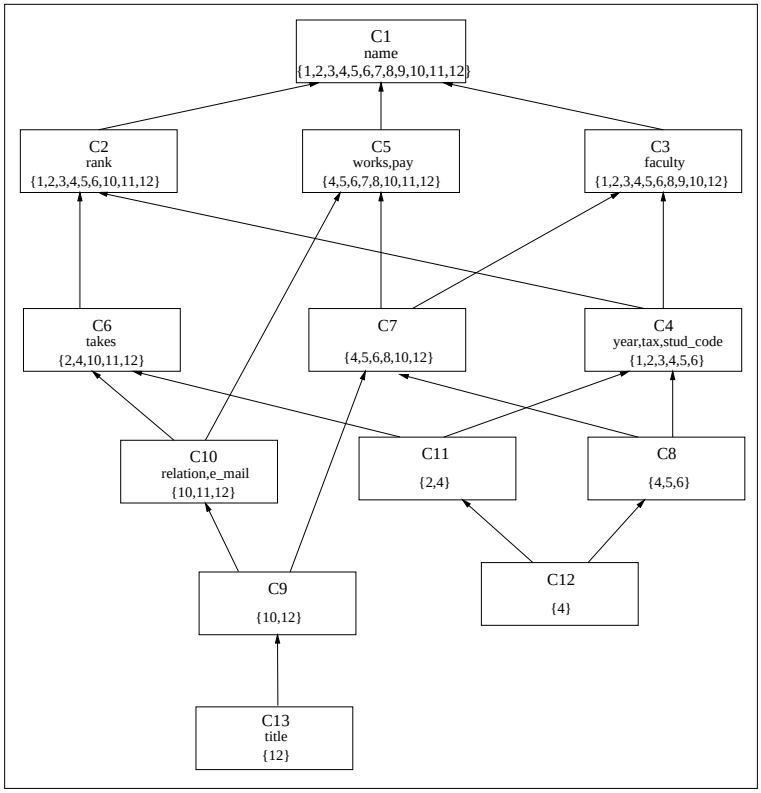


Figura 2.6: Rappresentazione della gerarchia estensionale

- individuare una sequenza di join che consenta la fusione di tutte le classi di una *base extension*(se non é possibile effettuare il join direttamente tra due classi occorre utilizzare delle classi di collegamento);

## 2.4 Definizione di un modello di rappresentazione dello Schema Globale

In questa sezione verrá definita una formalizzazione della conoscenza generata all'interno della fase di integrazione degli schemi. I concetti e le strutture definite vengono descritti attraverso una rappresentazione matematica.

### 2.4.1 Schema Globale

Sia **L** un insieme di *nomi di classi locali* (denotati da $L_1, L_2, \ldots$) e **AL** un insieme di *nomi di attributi locali* (denotati da $al_1, al_2, \ldots$). Gli attributi locali di un classe

locale sono determinati tramite la funzione $A_L : \mathbf{L} \rightarrow 2^{\mathbf{AL}}$.

Sia **G** un insieme di *nomi di classi globali* (denotati da $G_1, G_2, \ldots$) e **AG** un insieme di *nomi di attributi globali* (denotati da $ag_1, ag_2, \ldots$). Gli attributi globali di un classe globale sono determinati tramite la funzione $A_G : \mathbf{G} \rightarrow 2^{\mathbf{AG}}$.

**Definizione 2.4.1 (Schema Globale)** *Data un insieme **G** ed un insieme **L**, uno* schema globale $SG$ *di **L**, è una funzione* $SG$

$$SG : \boldsymbol{G} \rightarrow 2^{\boldsymbol{L}}$$

*tale che* $SG(G_1), SG(G_2), \ldots, SG(G_k)$ *sia un partizionamento di **L**.*

La *Mapping Table* è la struttura dati che contiene tutte le informazioni riguardanti il passaggio dalla rappresentazione globale agli schemi locali, definisce quindi la conoscenza intensionale di una classe globale $G$. É rappresentata da una matrice, le cui colonne sono gli attributi globali di $G$, $ag$, e le righe sono le classi locali di $G$, $L$; i suoi elementi rappresentano la corrispondenza fra la classe locale $L$ e l'attributo globale $ag$.

**Definizione 2.4.2 (Mapping Table)** *Data una classe globale $G$, una* Mapping Table *di $G$, $MT$ è una matrice* $||SG(G)|| \times ||A_G(G)||$

$$MT = \begin{pmatrix} m_{11} & m_{12} & \ldots \\ m_{21} & m_{22} & \ldots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

*i cui elementi* $m_{kh} = MT[L_k][ag_h]$ *possono assumere i seguenti valori:*

- $al_i \in A_L(L_k)$

  mapping semplice: $\exists!\ al_i \in A_L(L_k) : al_i \leftrightarrow ag_h$

- `null`

  null mapping: *not* $\exists\ al_i \in A_L(L_k) : al_i \leftrightarrow ag_h$

- `<costante>`

  default mapping: (*not* $\exists^{10}\ a_i \in A(c_k) : a_i \leftrightarrow ga_h) \vee (\exists\ a_i \in A(c_k) : a_i = ga_h =$ `<costante>`)

[10] In questo caso il valore di default é inserito dal progettista.

- $al_1$ `and` ... `and` $al_M$
  and mapping: $\exists\{al_1, \cdots, al_M\} \subseteq A_L(L_k) : \{al_1, \cdots, al_M\} \leftrightarrow ag_h$
- $al_1, \ldots, al_M$
  complex mapping: $\exists\{al_1, \cdots, al_M\} \subseteq A_L(L_k) : \{al_1, \cdots, al_M\} \leftrightarrow ag_h$
- **case** *al* **of** <costante>$_1$ : $al_1$, ... <costante>$_M$ : $al_M$
  union mapping: $\exists\{al, al_1, \cdots, al_M\} \subseteq A_L(L_k)$ : *se* $al =$ <costante>$_i$,
  *con* $i = 1, \ldots, M \Rightarrow al_i \leftrightarrow ag_h$

### 2.4.2 Base Extension

Informalmente, per *Base Extension* si intende un partizionamento dell'insieme complessivo di tutti gli oggetti rappresentati dalle sorgenti: sono sottoinsiemi disgiunti delle estensioni delle classi e sono ottenute dall'intersezione delle stesse. Più formalmente, un *insieme di Base Extension*, scritto $BES_{A1,A2,...,An}$ delle classi $A1, A2, ..., An$ viene definito da una formula booleana in DCF (Forma Canonica Disgiuntiva)[11] sulle variabili $A1, A2, ...., An$. Ogni *mintermine* di detta formula rappresenta una singola Base Extension.

Per applicare tale definizione nel contesto di una classe globale costituito da un insieme di classi locali, si parte dalla definizione di *Istanza di una classe globale*.

**Definizione 2.4.3 (Istanza di una classe globale)** *Data una classe globale $G$ ed un dominio **D**, un'istanza di $G$ sul dominio **D** è una funzione $\mathcal{I}$*

$$\mathcal{I} : SG(G) \rightarrow 2^{\boldsymbol{D}}$$

Un'istanza della classe globale $G$ verrà detta *legale* se soddisfa gli assiomi estensionali definiti sulle classi locali $SG(G)$.

**Definizione 2.4.4 (Base Extension)** *Data una classe globale G ed una sua istanza legale $\mathcal{I}$, un* set di base extension *di $G$ rispetto ad $\mathcal{I}$ è una coppia $(\boldsymbol{B}, F)$, dove **B** è un insieme di* nomi di base extension *(denotati da $B_1, B_2, \ldots,$) e $F$ è una funzione*

$$F : \boldsymbol{B} \rightarrow 2^{SG(G)}$$

*tale che*

1. $\bigcup_{B \in \boldsymbol{B}} F(B) = SG(G)$

[11] Vedi Definizione 3.1.1

*2.*

$$\bigcup_{B \in \boldsymbol{B}} \Big( \bigcap_{L \in F(B)} \mathcal{I}(L) - \bigcup_{L \in (SG(G) - F(B))} \mathcal{I}(L) \Big)$$

*sia un partizionamento di* $\mathcal{I}(G)$.

Un set di base extension di una classe globale $G$ viene rappresentato tramite una tabella le cui righe riportano le classi locali della classe globale, cioè $SG(G)$, le colonne riportano le base extension, cioè gli elementi di **B**: una X in corrispondenza dell'elemento della tabella $(L, B)$ indicherà che $L \in F(B)$; un esempio è riportato in 2.1.

La parte intensionale di una base extension viene determinata a partire dall'insieme di classi locali che la compongono, considerando come attributi della base extension l'unione degli attributi globali che hanno un mapping non nullo in tali classi locali. Formalmente:

**Definizione 2.4.5 (Attributi di una base extension)** *Data una classe globale* $G$*, un suo set di base extension* $(\boldsymbol{B}, F)$ *ed una Mapping Table di* $G$*,* $MT$*, si definiscono gli attributi di* $B \in \boldsymbol{B}$ *come segue:*

$$A_{BE}(B) = \{ag \in A_G(G) \mid \exists L \in F(B), MT[L][ag] \textit{è un mapping non nullo}\}.$$

### 2.4.3 Schema Virtuale e Gerarchia Estensionale

Nella sezione precedente una classe globale è stata caratterizzata tramite il suo set di base extension e ad ogni base extension sono stati associati un insieme di attributi globali. Ora si considera una query sulla classe globale ed il problema che si vuole affrontare è il seguente: quali solo le base extension che occorre considerare per rispondere all'interrogazione, cioè quali sono le base extension che hanno almeno tutti gli attributi specificati nell'interrogazione. Il problema può essere risolto semplicemente controllando per tutte le base extension il rispettivo insieme di attributi. Un'altra possibilità che velocizza la ricerca è quella di creare un *indice* che dato l'insieme di attributi dell'interrogazione restituisca direttamente l'insieme delle base extension interessate. A tale scopo, le base extension di una classe globale vengono raggruppate in *classi virtuali*; una classe virtuale è descritta da un insieme di attributi globali (*intensione*) e da un insieme di base extension (*estensione*) in modo tale che ogni attributo è comune a tutte le base extension e, viceversa, ogni base extension ha tutti gli attributi. Formalmente:

**Definizione 2.4.6 (Schema Virtuale)** *Data una classe globale* $G$ *ed un suo* set di base extension $BE = (\boldsymbol{B}, F)$*, lo schema virtuale di* $G$ *rispetto a* $BE$ *è una tripla* $(\boldsymbol{V}, INT, EST)$*, dove*

- ***V** è un insieme di* nomi di classi virtuali *(denotati da $V_1, V_2, \ldots, V_k$)*
- $INT : \boldsymbol{V} \rightarrow 2^{A_G(G)}$, intensione *dello schema virtuale*
- $EST : \boldsymbol{V} \rightarrow 2^{\boldsymbol{B}}$, estensione *dello schema virtuale*

*tale che*

1. $\forall V \in \boldsymbol{V}, \forall ag \in INT(V), \forall B \in EST(V)$ *si ha che* $ag \in A_{BE}(B)$
2. $\forall ag \in A_G(G), \exists V \in \boldsymbol{V} : ag \in INT(V)$
3. $\forall B \in \boldsymbol{B}, \exists V \in \boldsymbol{V} : B \in EST(V)$

Un esempio di schema virtuale per la classe globale `University_Person` è riportato in figura 3.12, dove con C1, ... ,C13 sono indicate le classi virtuali.

Le classi virtuali di una classe globale vengono organizzate in una gerarchia, detta *gerarchia estensionale*, sulla base della relazione di inclusione tra i rispettivi insiemi di attributi:

**Definizione 2.4.7 (Gerarchia Estensionale)** *Dato uno schema virtuale $(\boldsymbol{V}, INT, EST)$ di una classe globale $G$, la* Gerarchia Estensionale *è costruita sulla base della relazione $ISA_{EXT} \subseteq \boldsymbol{V} \times \boldsymbol{V}$ tale che, $\forall V, V' \in \boldsymbol{V}$:*

$$V \; ISA_{EXT} \; V' \;\; \texttt{iff} \;\; INT(V) \supseteq INT(V')$$

Dalle definizioni date si può verificare che:

$$V \; ISA_{EXT} \; V' \;\; \texttt{iff} \;\; EST(V) \subseteq EST(V')$$

In Figura 2.6 è illustrata la gerarchia estensionale relativa alla classe globale `Person`.

A questo punto possiamo illustrare in che modo lo schema virtuale e la relativa gerarchia estensionale di una classe globale sono utili per l'elaborazione di una query posta dall'utente sulla classe globale. Intuitivamente, data una query rispetto ad una classe globale, si individuano le classi più in basso nella gerarchia che contengono almeno tutti gli attributi globali specificati nella query. e si considerano quindi le base extension associate a tali classi virtuali.

Questa struttura consente quindi il passaggio da un'informazione di tipo intensionale ad una di tipo estensionale, in modo tale da minimizzare i tempi impiegati dal **Query Manager** per la procedura di ricerca della *Classe Virtuale Target*.

# Capitolo 3

# Algoritmo di calcolo della Conoscenza Estensionale

Nel capitolo precedente sono stati introdotti i concetti di *base extension* e *gerarchia estensionale* e spiegato il loro utilizzo. Occorre ora presentare l'algoritmo utilizzato per il calcolo di tali strutture dati.

In questa fase diventa fondamentale la figura del progettista poiché deve essere la principale fonte per la determinazione degli assiomi estensionali, si presuppone quindi che abbia una profonda conoscenza delle sorgenti che devono essere integrate sia a livello intensionale che estensionale. Mentre alcune relazioni possono essere ricavate dallo schema delle sorgenti (es. relazioni di inclusione tra super e sottoclasse) non é possibile ricavare nessun tipo di relazione estensionale tra classi locali appartenenti a sorgenti differenti. Per contro é alquanto improbabile che il progettista sia a conoscenza, fin dal principio, di tutte le relazioni estensionali che legano le classi da integrare, é molto probabile che sia nota solo un sottoinsieme di tale conoscenza. Per questo si é cercato di studiare un algoritmo [35] che proceda attraverso raffinamenti successivi. Partendo cioé da un insieme di assiomi é possibile calcolare l'insieme delle *base extension* e la relativa gerarchia estensionale, che saranno poi utilizzati per la deduzione di nuovi assiomi che andranno ad aggiungersi a quelli giá presenti. É possibile iterare questo procedimento fino al raggiungimento di un obiettivo considerato accettabile (vedi Figura 3).

Analizzando la figura si vede come in base agli assiomi estensionali, definiti dal progettista, vengono ricavati i *requisiti di esistenza*[1] e quindi la gerarchia estensionale. Una volta determinata tale gerarchia é possibile, in base al risultato ottenuto, modificare l'insieme iniziale degli assiomi fino al raggiungimento del risultato voluto.

[1] Il concetto di *requisito di esistenza* verrá chiarito in seguito

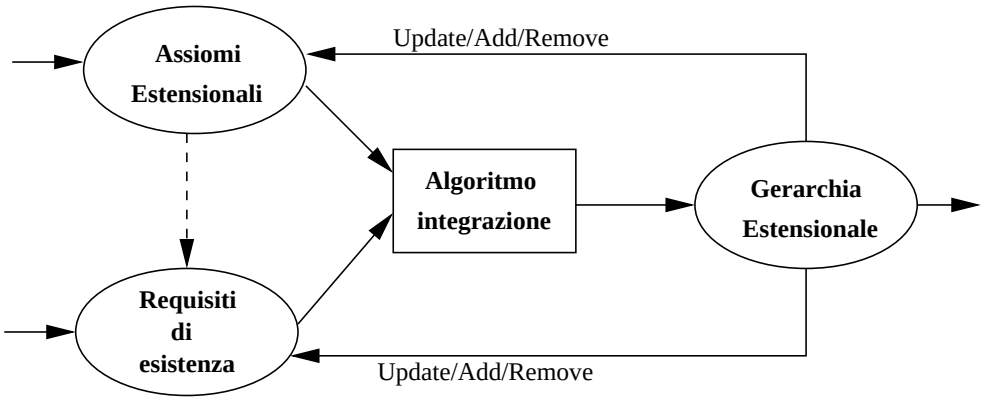


Figura 3.1: Procedimento di integrazione

## 3.1 Concetti base

Prima di illustrare i passi dell'algoritmo occorre dare alcune definizioni che saranno fondamentali per la comprensione.

**Definizione 3.1.1 (Forme Disgiuntive)** *Date n variabili di ingresso* $A_1, \ldots, A_n$*, una formula booleana viene detta in* forma normale disgiuntiva *o DNF quando é formata da una sommatoria di termini, ciascuno dei quali costituito da una produttoria di letterali costituiti da un sottoinsieme delle n variabili di ingresso oppure da una loro negazione. Se ogni termine é composto da una produttoria di letterali costituita da tutte le n variabili di ingresso allora la forma si dice* forma canonica disgiuntiva[2] *o DCF. La formula viene detta* minimale *quando, applicando le proprietá algebriche di equivalenza, non é possibile ottenere una formula disgiuntiva equivalente contenente un numero inferiore di letterali.*

**Definizione 3.1.2 (Forme Congiuntive)** *Date n variabili di ingresso* $A_1, \ldots, A_n$*, una formula booleana viene detta in* forma normale congiuntiva *o CNF quando é formata da una produttoria di termini, ciascuno dei quali costituito da una sommatoria di letterali costituiti da un sottoinsieme delle n variabili di ingresso oppure da una loro negazione. Se ogni termine é composto da una sommatoria di letterali costituita da tutte le n variabili di ingresso allora la forma si dice* forma canonica congiuntiva[3] *o CCF. La formula viene detta* minimale *quando, applicando le proprietá algebriche di equivalenza, non é*

[2]La DCF puó essere definita anche come un sottoinsieme di tutti i possibili mintermini delle variabili di ingresso $A_1, \ldots, A_n$

[3]La CCF puó essere definita anche come un sottoinsieme di tutti i possibili maxtermini delle variabili di ingresso $A_1, \ldots, A_n$

| | | |
|---|---|---|
| Idempotenza: | A + A | = A |
| | AA | = A |
| Complemento: | A + $\overline{A}$ | = true |
| | A$\overline{A}$ | = false |
| Assorbimento: | A + (AB) | = A |
| | A(A + B) | = A |
| Identitá: | A + true | = true |
| | A + false | = false |
| | A true | = true |
| | A false | = false |

Tabella 3.1: Regole di semplificazione

*possibile ottenere una formula congiuntiva equivalente contenente un numero inferiore di letterali.*

É possibile fare un esempio per chiarire ulteriormente. Date le variabili di ingresso $A_1, A_2, A_3$ posso scrivere una generica espressione in DNF[4]:

$$DNF = A_2 + A_1A_2\overline{A_3} + \overline{A_1}A_2\overline{A_3} + A_2\overline{A_3} + A_1\overline{A_3}$$

Mentre una loro espressione in DCF é la seguente:

$$DCF = A_1A_2A_3 + A_1A_2\overline{A_3} + \overline{A_1}A_2\overline{A_3}$$

Utilizzando invece la notazione CNF:

$$CNF = (A_1 + A_2)(A_2 + \overline{A_3})(\overline{A_1} + A_2 + \overline{A_3})$$

É sempre possibile passare da una notazione di tipo CNF ad una di tipo DNF applicando la proprietá distributiva, utilizzando poi le regole di semplificazione definite in Tabella 3.1 si ottiene la forma minimale.

Ricordando la definizione delle rule estensionali (Definizione 2.3.2) é possibile rappresentarle utilizzando i due tipi di notazione appena introdotti. Il risultato é schematizzato in Tabella 3.2. Mentre in Figura 3.2 sono rappresentate tutte le possibili relazioni estensionali tra due generiche classi, si nota che il numero massimo di *base extension* tra due classi é di quattro.

Riprendendo quanto detto nel Paragrafo 2.4.2 é possibile dare una definizione alternativa di insieme di *base extension* in accordo con la definizione 2.4.4.

[4]Le operazioni booleane di AND e di OR sono rappresentate rispettivamente dagli operatori · e +. Il simbolo · viene sempre omesso.

| | DNF | CNF |
|---|---|---|
| A ≡ B | A B + $\overline{A}$ $\overline{B}$ | ( A + $\overline{B}$)($\overline{A}$ + B) |
| A ⊂ B | A B + $\overline{A}$ B + $\overline{A}$ $\overline{B}$ | $\overline{A}$ + B |
| B ⊂ A | A B + $\overline{B}$ A + $\overline{B}$ $\overline{A}$ | $\overline{B}$ + A |
| A ⊘ B | A $\overline{B}$ + $\overline{A}$ B + $\overline{A}$ $\overline{B}$ | $\overline{A}$ + $\overline{B}$ |
| A ∩ B | A B + A $\overline{B}$ + $\overline{A}$ B + $\overline{A}$ $\overline{B}$ | true |

Tabella 3.2: Formule DNF e CNF per le rule estensionali

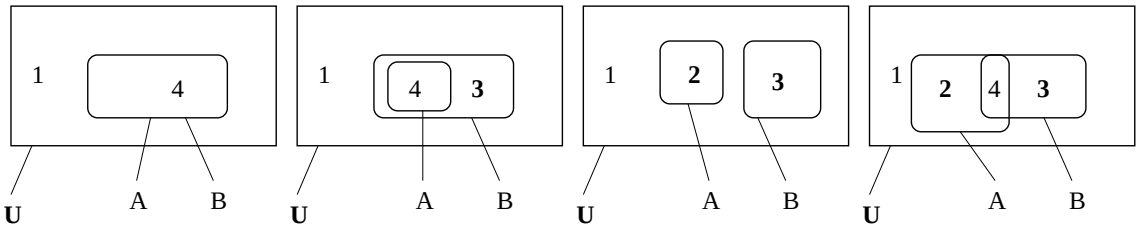


Figura 3.2: Relazioni estensionali definibili tra coppie di classi

**Definizione 3.1.3 (Insieme di Base Extension o BES)** *Un insieme di base extension, scritto $BES_{A1,...,An}$, delle classi $A_1$,...,$A_n$ puó essere rappresentato da una formula booleana in Forma Canonica Disgiuntiva (DCF), dove i termini sono rappresentati dalle classi $A_1$,...,$A_n$. Ogni mintermine di questa formula rappresenta una singola base extension ed ogni possibile oggetto dell'insieme di classi $A_1$,...,$A_n$ puó essere assegnato ad una ed una sola base extension.*

In base a questa definizione si ricava immediatamente che il numero massimo di base extension individuabili tra n classi é di $2^n$. Ponendo, per esempio, n=2 si ricavano quattro base extension, rappresentate dalla formula booleana:

$$A_1\ A_2 + A_1\ \overline{A_2} + \overline{A_1}\ A_2 + \overline{A_1}\ \overline{A_2}$$

## 3.2 Algoritmo per il calcolo delle base extension

Presentiamo ora una metodologia che permette di calcolare un insieme corretto[5] e completo[6] di base extension date n classi legate da relazioni.

Alcune rule estensionali possono essere derivate da relazioni interschema, ad esempio una relazione di specializzazione tra due classi implica una rule estensionale di inclusione,ma queste relazioni rappresentano una minoranza rispetto a

[5]Cioé soddisfa tutte le rule estensionali definite sull'insieme delle classi.

[6]Descrive tutta la conosenza estensionale che il progettista voleva rappresentare

quelle inserite dal progettista, le quali forniscono quelle informazioni aggiuntive che permettono di sfruttare al massimo tutte le potenzialitá ed i vantaggi offerti dall'uso della conoscenza estensionale. Un prerequisito fondamentale per una corretta integrazione é rappresentato dalla corretta e completa conoscenza da parte del progettista delle relazioni estensionali che legano le diverse classi coinvolte.

Raramente il progettista é in grado di specificare,sin dal principio, tutte le rule estensionali esistenti, per questo spesso si preferisce generare una soluzione imcompleta ma di immediata definizione piuttosto che investire grosse risorse per determinare tutta la conoscenza estensionale. Questa metodologia introduce un approccio diverso, di tipo incrementale. Partendo da un insieme non completo di rule é possibile calcolare il relativo insieme di base extension e la gerarchia estensionale che possono essere utilizzati per la deduzione di nuova conoscenza da integrare, il processo viene ripetuto iterativamente fino al raggiungimento di una soluzione soddisfacente. Inoltre il procedimento é in grado di controllare la consistenza delle rule e rilevare e segnalare eventuali relazioni in conflitto tra di loro.

Supponiamo che l'insieme $\mathcal{A}$ contenga tutte le possibili rule estensionali definibili tra *n* classi e l'insieme $\mathcal{M}$ contiene i $2^n$ mintermini corrispondenti. Calcolare l'insieme delle base extension BES relativo ad un dato insieme di rule estensionali $\mathcal{R}$ significa implementare la funzione *generate* che puó essere definita :

$$generate\colon 2^{\mathcal{A}} \rightarrow 2^{\mathcal{M}}$$

Dove BES $\in 2^{\mathcal{M}}$ e $\mathcal{R} \in 2^{\mathcal{A}}$

Supponiamo di applicare tale metodologia all'esempio giá introdotto (vedi Sezione 2.2), in particolare alla classe globale[7] **Person (GC1)**(Tabella 2.3. Per rendere la comprensione dell'algoritmo piú chiara ed evitare calcoli troppo onerosi si suppone la classe globale formata da quattro classi locali (vedi Tabella 3.3).

Dallo schema delle sorgenti vengono ricavate le rule :

1. `CSS` $NT_{Ext}$ `CSP`;

2. `CSPr` $NT_{Ext}$ `CSP`;

Mentre il progettista inizialmente specifica l'assioma:

[7] Si ricorda che il calcolo della gerarchia estensionale puó essere attuato solo dopo avere risolto tutti i conflitti intensionali, cioé dopo aver definito la *mapping table*, quindi é applicabile alle singole classi globali

| **Classe locale** | **Alias** |
|---|---|
| University.Research_Staff | URS |
| Computer_Science.CS_Person | CSP |
| Computer_Science.Professor | CSPr |
| Computer_Science.Student | CSS |

Tabella 3.3: Classi locali che compongono la classe globale

| Rule Ext. | CNF |
|---|---|
| 1 | $\overline{CSS} + CSP$ |
| 2 | $\overline{CSPr} + CSP$ |

Tabella 3.4: Rule ricavata dallo schema delle sorgenti

3. `CSPr` $NT_{Ext}$ `URS`;

Nelle Tabelle 3.4 e 3.5 sono schematizzate le forme CNF degli assiomi.

Per il calcolo dell'insieme delle base extension si segue il seguente procedimento :

Dato un insieme di rule $\mathcal{R} = r^1, r^2, \ldots, r^m$

1. Trasformare ogni rule estensionale $r^i$ nella corrispondente forma congiuntiva normale CNF $r^i_{CNF}$;
2. Combinare insieme tutte le forme trovate al passo precedente in un unica forma normale congiuntiva $r^1_{CNF} r^2_{CNF} \ldots r^m_{CNF}$;
3. Trasformare la forma normale congiuntiva in forma normale disgiuntiva DNF e applicare le regole di semplificazione(vedi Tabella 3.1) per rendere la forma il piú semplice possibile;

Si ottiene una espressione booleana in forma normale disgiuntiva DNF che ancora non rappresenta l'insieme delle base extension cercato poiché in ogni termine dell'espressione non necessariamente compaiono tutte le classi coinvolte.

| Rule Ext. | CNF |
|---|---|
| 3 | $\overline{CSPr} + URS$ |

Tabella 3.5: Rule definite dal progettista

Per ottenere un insieme di base extension corretto occorre trasformare la forma DNF appena ricavata nella corrispondente forma canonica disgiuntiva DCF.

In base alla rule ricavata dagli schemi ed a quella inserita dal progettista ricaviamo la forma CNF :

$$\text{CNF} = \quad (\overline{CSS} + CSP)(\overline{CSPr} + CSP)(\overline{CSPr} + URS)$$

Si ricava la corrispondente forma DNF minimale :

$$\text{DNF} = \quad \text{CSP } \overline{CSPr} + \text{CSP URS} + \overline{CSPr}\ \overline{CSS}$$

Come accennato nel Capitolo pecedente, all'interno delle classi globali si suppone per default un assioma di sovrapposizione estensionale tra quelle coppie di classi per le quali non é stata definita nessuna relazione. Poiché la rule di sovrapposizione in notazione CNF ha traduzione uguale a *true*, questa non va a modificare in alcun modo la formula booleana. Un primo vantaggio dell'utilizzo di questo algoritmo é dato proprio dal fatto che l'ipotesi di default non comporta nessun calcolo ulteriore per la determinazione della forma CNF.

Visto che la forma DNF appena calcolata é stata ricavata direttamente dalle rule estensionali, rappresenta quindi le condizioni minime che ogni *base extension* deve verificare per poter essere corretta. Per ottenere un insieme di base extension[8] coerente con le rule date sarebbe sufficiente passare alla relativa forma canonica ottenendo $(3*\frac{2^4}{4}) - (2^1 + 2^1 + 2^0) = 7$ base extension. Il rischio é quello di avere un esplosione nel numero delle base extension poiché, per esempio, nel caso di dieci classi con una sola rule di inclusione definita si ottengono $(2 * \frac{2^{10}}{2}) - 2^8 = 768$ base extension che rappresentano il numero massimo di base extension ottenibili con le rule date. In alternativa si puó provare ad effettuare alcune considerazioni per cercare di individuare una metodologia che permetta di ridurre il numero delle base extension.

Le relazioni estensionali oltre a imporre delle condizioni che dovranno essere verificate dall'insieme delle base extension assicurano anche l'esistenza di alcuni insiemi sicuramente popolati da istanze. Una rule di intersezione A ∩ B implica l'esistenza degli insiemi:

$$\text{AB, } \overline{A}\text{B,A}\overline{B}$$

[8]Date n rule su m classi l'insieme di base extension che soddisfa le n rule non é in genere unico. Lo diventa solamente se vengono specificate tutte le $\frac{m*(m-1)}{2}$ possibili rule tra le m classi

| Rule Est. | Req. esistenza |
|---|---|
| A $\equiv$ B | AB |
| A $\subset$ B | AB, $\overline{A}$B |
| B $\subset$ A | AB, A$\overline{B}$ |
| A $\oslash$ B | A$\overline{B}$, $\overline{A}$B |
| A $\cap$ B | AB, A$\overline{B}$, $\overline{A}$B |

Tabella 3.6: Requisiti di esistenza

In pratica se le classi A e B si intersecano devono esistere oggetti che appartengono all'intersezione e oggetti che sono membri di una sola delle due classi. Analogo ragionamento é possibile fare gli altri tipi di rule. Queste espressioni logiche derivate dalle rule sono chiamate *requisiti di esistenza*.

L'insieme dei requisiti di esistenza definibili per ogni rule estensionale sono riportati in Tabella 3.6.

**Definizione 3.2.1 (Requisito di esistenza)** *Un* requisito di esistenza *é rappresentato da una espessione logica che deve essere soddisfatta da almeno una* base extension *dell'insieme calcolato in base alla rule estensionali definite. Deve essere verificata in ogni momento la:*

$$State^{t}_{ER} \neq \emptyset$$

*dove ER é un requisito di esistenza.*

Nota la forma DNF compiendo la trasformazione alla forma DCF si ottiene un insieme di *base extension* corretto ma ridondante, cioé contiene in numero massimo di *base extension* ottenibili con le rule date, mentre utilizzando i *requisiti di esistenza* si determina l'insieme minimo, sicuramente popolato, di *base extension*. Si puó osservare che gli assiomi di default (sovrapposizione estensionale) generano dei *requisiti di esistenza* che occorre tenere in considerazione per il calcolo delle *base extension*. Una volta nota la forma DNF non occorre piú effettuare la conversione alla forma DCF ma attraverso l'utilizzo dei requisiti di esistenza si riesce ugualmente ad individuare un insieme corretto di base extension.

Abbiamo ora a disposizione la forma DNF ricavata dalle rule e l'insieme dei requisiti di esistenza, entrambi questi due elementi devono essere *soddisfatti* dall'insieme di base extension finale. Di seguito diamo una definizione piú rigorosa del concetto.

**Definizione 3.2.2 (Soddisfacimento da parte di un insieme di base extension)** *Date* n *classi e posto :*

*A =* $\{a^1_{CNF}, \ldots, a^k_{CNF}\}$ *come l'insieme delle rule estensionali in CNF.*
*E =* $\{e^1_{DNF}, \ldots, e^l_{DNF}\}$ *come l'insieme dei requisiti di esistenza in DNF, dove ogni requisito* $e^i_{DNF}$ *é formato da un insieme di termini* $et^{i_1}_{DNF}, \ldots, et^{i_p}_{DNF}$.
*Un insieme di base extension BES =* $\{m^1_{DNF}, \ldots, m^m_{DNF}\}$ *dove ogni mintermine* $m^j_{DNF} = v^{j_1}, \ldots, v^{j_n}$ *e le variabili* $v^{j_r}$ *rappresentano le classi e possono comparire all'interno del mintermine in forma diretta (*$v^{j_r}_+$*) o in forma negata (*$v^{j_r}_-$*).*
*L'insieme BES* soddisfa *gli insiemi A ed E* $\Leftrightarrow$

- *Tutti gli elementi di A sono soddisfatti dall'insieme BES :*

$$\forall a_{CNF} \in A\colon BES \Rightarrow a_{CNF}$$

- *Tutti gli elementi di E sono soddisfatti dall'insieme BES :*

$$\forall e_{DNF} \in E\colon \exists et_{DNF} \in e_{DNF}\colon \exists m \in BES\colon m \Rightarrow et_{DNF}$$

Quindi per il calcolo delle base extension occorre oltre all'insieme delle rule in CNF anche l'insieme dei requisiti di esistenza. Possiamo quindi espandere la funzione *generate* vista all'inizio:

$$generate\colon (2^{\mathcal{A}} \times 2^{\mathcal{E}}) \rightarrow 2^{\mathcal{M}}$$

dove $\mathcal{E}$ rappresenta l'insieme di tutti i possibili requisiti di esistenza tra *n* classi.

Illustriamo ora in dettaglio i passi dell'algoritmo per il calcolo dell'insieme di base extension utilizzando i requisiti di esistenza:

1. Utilizzando l'insieme delle rule in forma DNF si costruisce una tabella, detta *Source Table* che verrá utilizzata come base di partenza per il calcolo. Ad ogni riga della *Source Table* corrisponde una classe mentre per ogni termine della forma DNF delle rule si aggiunge una colonna e si inserisce un '1' in corrispondenza di quelle classi che sono presenti in forma diretta mentre si inserisce uno '0' per le classi presenti in forma negata, le caselle relative alle classi non presenti nel termine vengono lasciate vuote.
   Ritornando all'esempio si era ricavato:

   DNF = CSP $\overline{CSPr}$+CSP URS+$\overline{CSPr}$ $\overline{CSS}$

   Si ricava la relativa *Source Table*:

| | | | |
|---|---|---|---|
| CSP | 1 | 1 | |
| CSS | | | 0 |
| CSPr | 0 | | 0 |
| URS | | 1 | |

Procedendo in questo modo si ha la certezza che le base extension che verranno trovate *soddisfano*[9] le rule estensionali.

2. Creazione di una tabella, detta *Target Table*, vuota con lo stesso numero di righe della *Source Table* e senza nessuna colonna.

| |
|---|
| CSP |
| CSS |
| CSPr |
| URS |

Tabella 3.7: Target Table relativa all'esempio

3. Dato l'insieme E dei requisiti di esistenza $e^i$ si eseguono i seguenti passi per ogni $e^i \in$ E:
   (a) Se la *Target Table* soddisfa $e^i$ allora si passa al requisito successivo;
   (b) Se la *Target Table* é in grado di soddisfare $e^i$ aggiungendo valori '0' od '1' ad una colonna,si riempiono solamente tali campi con i valori adeguati nella prima colonna disponibile. In seguito si passa al requisito successivo;
   (c) Si prende la prima colonna della *Source Table* in grado di soddisfare $e^i$ come al punto (b) e la si aggiunge alla *Target Table*. Si passa poi al requisito successivo;

   Procedendo in questo modo si ha la certezza che l'insieme di base extension trovato *soddisfa* tutti i requisiti di esistenza.

4. Si riempiono i rimanenti campi vuoti con il valore"0"[10]

La complessitá di questo algoritmo é di tipo polinomiale e dipende direttamente dal numero e dal tipo delle rule estensionali formulate. Il procedimento descritto al punto 3 é illustrato in Figura 3.3. L'insieme E dei *requisiti di esistenza* che vengono ricavati dalle rule definite e da quelle di default é il seguente :

[9] vedi Definizione 3.2.2

[10] Con questa operazione si specifica il fatto che nei casi in cui non sia stata specificata nessuna regola si suppone che la relativa classe non sia presente nella base extension

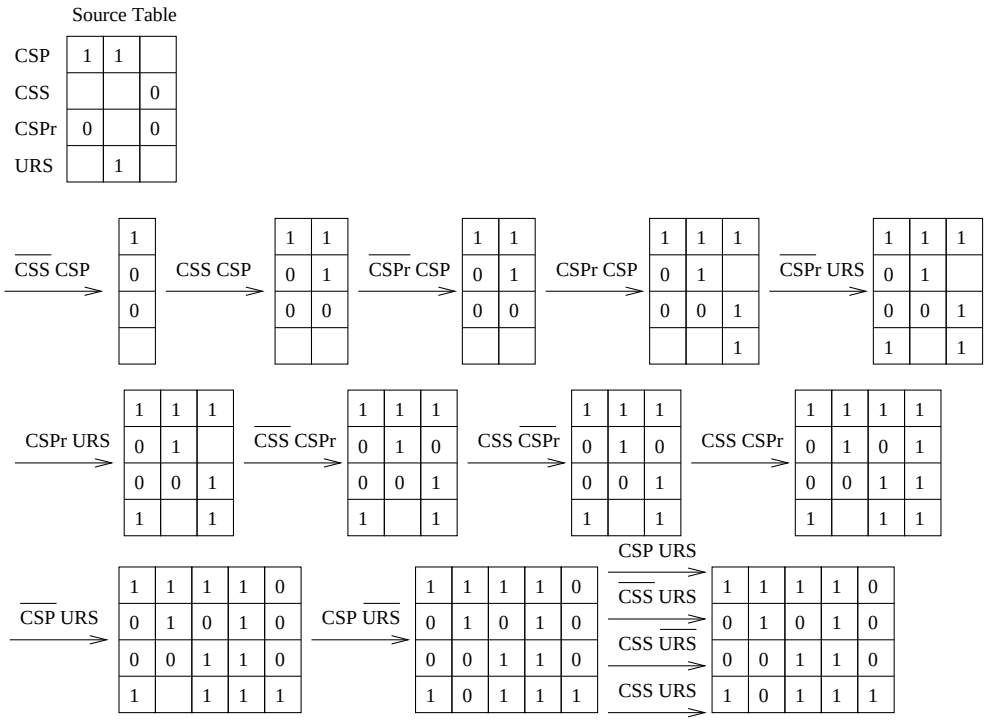


Figura 3.3: Algoritmo per il calcolo delle base extension

E = {$\overline{CSS}$ CSP,CSS CSP,$\overline{CSPr}$ CSP,CSPr CSP, $\overline{CSPr}$ URS,CSPr URS, $\overline{CSS}$ CSPr,CSS $\overline{CSPr}$,CSS CSPr,$\overline{CSP}$ URS,CSP $\overline{URS}$,CSP URS, $\overline{CSS}$ URS,CSS $\overline{URS}$,CSS URS}

L'insieme delle base extension ottenuto é illustrato nella Tabella 3.8. Per maggiore chiarezza é possibile rappresentare graficamente la tabella delle *base extension*, come mostrato in Figura 3.4

Utilizzando l'algoritmo che verrá presentato nella Sezione 3.3 si puó calcolare la relativa gerarchia estensionale, mostrata in Figura 3.5.

A questo punto il progettista ha sufficenti informazioni per analizzare il risultato ottenuto ed eventualmente modificarlo agendo sugli assiomi estensionali. Si supponga che analizzando tutte le informazioni a disposizione venga dedotto un ulteriore assioma:

- `CS.Professor` $DISJ_{Ext}$ `CS.Student`;

Reiterando il procedimento appena illustrato si arriva alla definizione di un

| Base Ext. | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| CSP | 1 | 1 | 1 | 1 | 0 |
| CSS | 0 | 1 | 0 | 1 | 0 |
| CSPr | 0 | 0 | 1 | 1 | 0 |
| URS | 1 | 0 | 1 | 1 | 1 |

Tabella 3.8: Insieme delle *base extension* BES

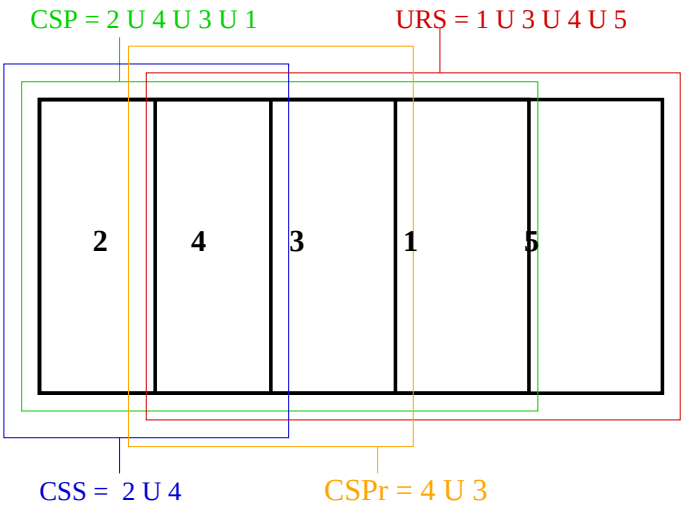


Figura 3.4: Insieme delle *base extension*

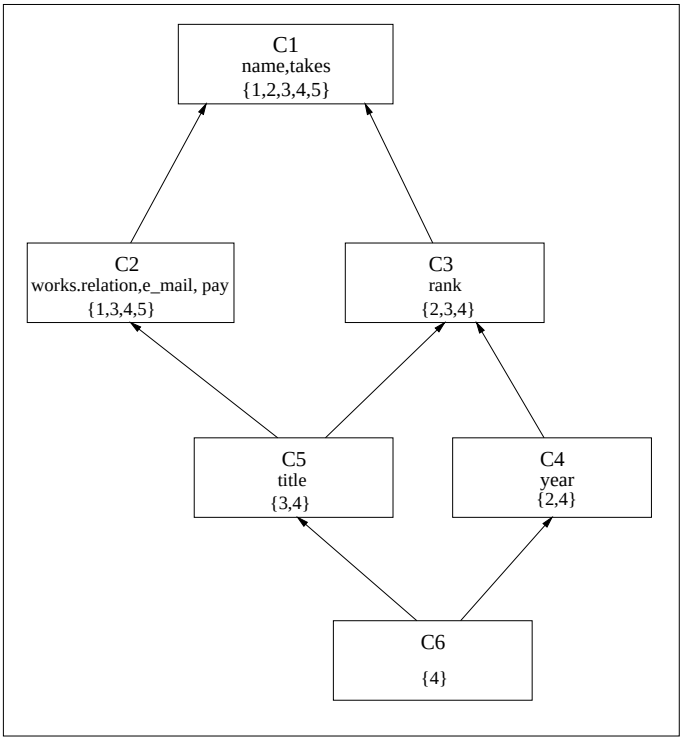


Figura 3.5: Gerarchia estensionale

nuovo insieme BES(Tabella 3.9) e di una nuova gerarchia estensionale (vedi Figura 3.6) che rispettano le rule date.

| | | | | | |
|---|---|---|---|---|---|
| CSP | 1 | 1 | 1 | 0 | 1 |
| CSS | 1 | 0 | 0 | 0 | 1 |
| CSPr | 0 | 0 | 1 | 0 | 0 |
| URS | 1 | 0 | 1 | 1 | 0 |

Tabella 3.9: Insieme delle *base extension* BES

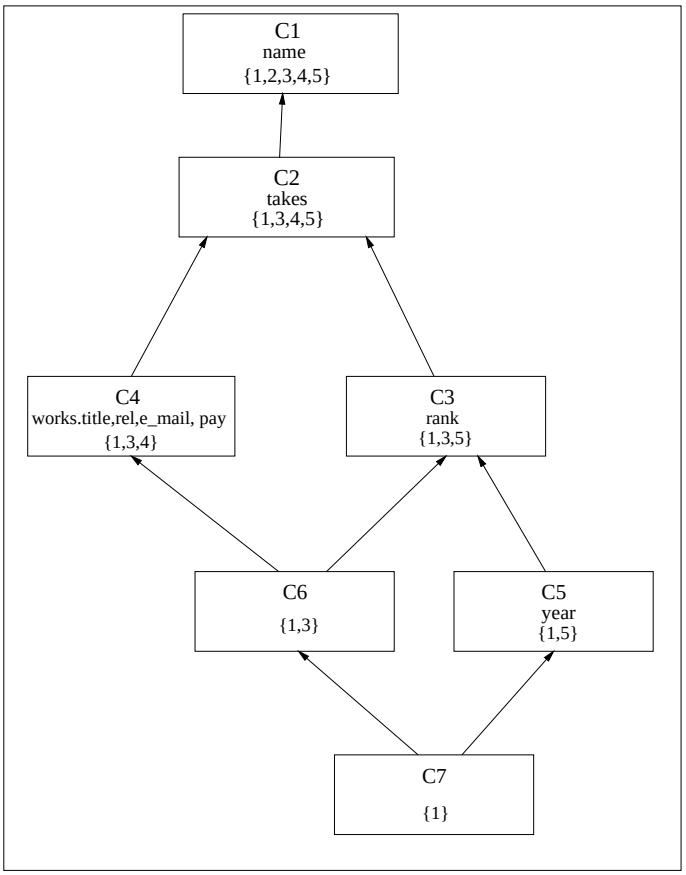


Figura 3.6: Gerarchia estensionale

Occorre naturalmente prevedere la situazione nella quale il progettista inserisca inavvertitamente un insieme di assiomi che non sono consistenti tra loro e che quindi porterebbero alla generazione di un insieme BES scorretto. Nella

prossima sezione verrá presentata la metodologia adottata per affrontare questo problema.

### 3.2.1 Rilevazione degli assiomi inconsistenti

L'algoritmo presentato ha il grosso vantaggio di essere in grado di rilevare eventuali assiomi inconsistenti.

Seguendo il procedimento illustrato dal punto 1 al punt 4 é possibile identificare tre possibili differenti stati(vedi Figura 3.7):

1. **BES Incompleto.** Gli assiomi specificati non presentano inconsistenze ma non rappresentano la conoscenza estensionale esprimibile per le classi date. In questo caso viene calcolata la gerarchia estensionale perché risulta di fondamentale supporto al progettista per la deduzione di nuovi assiomi o la modifica di quelli giá esistenti.

2. **BES Corretto.** L'insieme BES trovato non presenta inconsistenze ed esprime tutta la conoscenza estensionale che il progettista é in grado di fornire. In questo caso viene in seguito calcolata la relativa gerarchia estensionale ed entrambe queste strutture dati vengono registrate per essere utilizzate dal *Query manager*

3. **BES Errato.** Gli assiomi specificati sono inconsistenti.
   Supponiamo che dopo aver definito gli assiomi:

   - CSS $NT_{Ext}$ CSP;
   - CSPr $NT_{Ext}$ CSP;
   - CSPr $NT_{Ext}$ URS;

   e calcolato il relativo insieme BES (vedi tabella 3.8) Il progettista aggiunge l'assioma:

   - CSP $DISJ_{Ext}$ URS;

   essendo la classe CSPr contenuta sia nella classe CSP che nella classe URS, le due classi CSP e URS non possono essere disgiunte. Un importante considerazione da fare é che in questo caso non si parla di assioma inconsistente

ma di un insieme di assiomi inconsistenti tra loro. Per rendere consistente questo insieme di assiomi é possibile, sia eliminare l'ultima rule definita sia modificare o cancellare la relazione di inclusione tra CSPr e URS.

Calcolando la *Source Table* per l'insieme inconsistente di assiomi risulta:

| | | | | |
|---|---|---|---|---|
| CSP | 0 | | 0 | 1 |
| CSS | 0 | 0 | 0 | |
| CSPr | 0 | 0 | 0 | 0 |
| URS | | 0 | 1 | 0 |

L'assioma di inclusione tra CSPr e URS genera il *requisito di esistenza* $CSPr\ URS$ che non verifica la *Source Table*, in questo modo viene rilevata l'inconsistenza, inoltre, forzando questo controllo ogni volta che viene inserito un nuovo assioma é possibile risalire all'insieme di rule che risultano inconsistenti rispetto all'ultima inserita. Diviene cosí possibile segnalare al progettista quali sono le rule da correggere.

Gli assiomi di sovrapposizione estensionale, di default per la classe globale, possono,in alcuni casi, rendere inconsistente l'insieme degli assiomi. Poiché tali rule non sono espressamente inserite dal progettista é stato deciso di far reagire in modo differente il sistema a seconda del tipo di rule che ha generato l'inconsistenza, e precisamente:

- **assioma inserito dal progettista.** Viene segnalata l'inconsistenza e l'algoritmo si blocca finché non vengono modificati gli assiomi;
- **assioma di default.** La rule che causa inconsistenza viene scartata, non vi é alcuna segnalazione per il progettista;

Una volta determinato l'insieme delle *base extension* occorre ricavare la gerarchia estensionale, per farlo viene applicata la teoria del *formal context analisys* [41] che verrá spiegata in dettaglio nel Paragrafo 3.3.

## 3.3 Algoritmo per la generazione della gerarchia estensionale

Questa tecnica è illustrata in [41] ed è stata utilizzata per tradurre la conoscenza estensionale necessaria per le fasi di integrazione e query processing. Facciamo un breve esempio chiarificatore e descriviamo la metodologia ideata, che viene impiegata per completare l' integrazione delle informazioni in MOMIS.

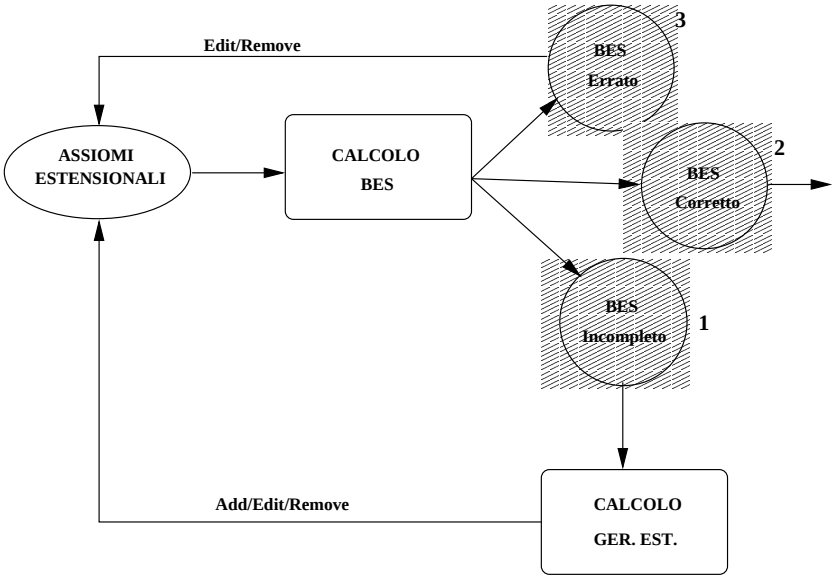


Figura 3.7: Stati finali dell'algoritmo

Supponiamo di applicare tale algoritmo alla classe globale Person(GC1). Dato un insieme di assiomi estensionali siamo in grado, come illustrato nelle sezioni precedenti, di determinare l'insieme BES delle *base extension*. In questa sezione verrá presentato l'algoritmo che, partendo dalle *base extension*, permette il calcolo della gerarchia estensionale.

Una volta determinata tale gerarchia il progettista, analizzando tutte le informazioni ricavate, avrá a disposizione un supporto per la deduzione di nuovi assiomi o la modifica di altri giá esistenti forzando in questo modo il ricalcolo dell'insieme delle *base extension* e della gerarchia estensionale.

Supponiamo di avere a disposizione l'insieme BES giá presentato in Tabella 2.1.

### 3.3.1 Metodologia per la fusione delle gerarchie

La tecnica per ottenere questo risultato applica la teoria detta *formal context analysis* [41]. Date le gerarchie iniziali, in cui si immaginano già risolti i conflitti intensionali, i passi per ottenere uno schema integrato sono i seguenti:

- espressione delle sovrapposizioni estensionali attraverso l' introduzione delle *base extension*

  Le base extension costituiscono una partizione dell' insieme complessivo

di tutti gli oggetti rappresentati dalle sorgenti: sono sottoinsiemi disgiunti delle estensioni delle classi e sono ottenute dall' intersezione delle stesse.

- espressione delle sovrapposizioni intensionali

  Avendo risolto i conflitti intensionali gli attributi sinonimi identificano le medesime proprietà nelle classi che li comprendono. Al fine di creare una nuova gerarchia è indispensabile rappresentare non solo la condivisione degli oggetti delle classi sorgenti ma anche delle proprietà (attributi).

- generazione del *context*

  Il context costituisce l' input per l' algoritmo di creazione della nuova gerarchia. Esso esprime la relazione esistente tra le base extension e gli attributi, ovvero specifica gli attributi di ogni base extension.

- creazione del lattice rappresentante la gerarchia

  L' algoritmo elaborato genera un insieme di classi caratterizzate da una certa intensione (attributi) ed estensione (un insieme di base extension). Dalla analisi degli attributi di ogni classe si può stabilire la gerarchia di specializzazione e generare la gerarchia finale. La complessità dell' algoritmo è polinomiale in n=max(numero degli attributi, numero delle base extension).

Si descrive ogni passo con maggiore dettaglio.

**1.** La prima informazione di cui occorre disporre è la relazione di sovrapposizione fra le estensioni delle classi dei sorgenti rappresentata dalla tabella delle *base extension*(Tabella 2.1), fondamentale alla creazione della gerarchia.

**2.** Analogamente si esprime in una matrice anche il legame fra gli attributi e le classi sorgenti (Intensional overlapping, in Tabella 2.2): per righe si hanno le classi, per colonne gli attributi: le caselle marcate indicano che un certo attributo è una proprietà della classe corrispondente.

**3.** I passi precedenti preparano alla individuazione degli attributi propri degli oggetti nelle base extension (Extension-Attribute Relation, Tabella 2.3). Si crea una matrice che ha per righe gli attributi e per colonne le base extension: le caselle marcate sono ricavate sulla base delle precedenti tabelle: per ogni base extension (colonna) sono marcate le caselle corrispondenti agli attributi che caratterizzano le proprietà degli oggetti che descrive. Per determinare gli attributi da marcare in corrispondenza di una certa colonna (base extension) si considera la Tabella 2.1 e si scelgono le classi corrispondenti a delle caselle marcate, cioè la cui estensione includa la base extension, poi dalla Tabella 2.2 si prende l' unione di tutti gli attributi che caratterizzano le classi scelte.

**Definizione 3.3.1 (Context)** *Un* context *è individuato da una terna (G, M, I) dove G è un insieme di oggetti, M è un insieme di attributi, I è l' insieme di coppie (oggetto g, attributo m) tali che g possiede l' attributo m.*

Nel nostro esempio G è l' insieme delle base extension, M degli attributi, I l'insieme delle coppie (g, m) evidenziate dalla Tabella 2.3.

**4.** L' algoritmo di creazione della gerarchia ha in input il *context* ed in output una gerarchia di classi costituite dall' unione di base extension. Le fasi di lavoro dell' algoritmo si susseguono in quest' ordine:

- definizione degli insiemi *Int* ed *Ext* (in Tabella 3.10)

  $$Int: = \{intent(\{g\}) \mid g \in G\}$$

  *Int* é l' insieme di tutti i gruppi di attributi che caratterizzano ogni base extension. Ogni elemento dell' insieme corrisponde ad una colonna[11] della Tabella 2.3.

  $$Ext: = \{extent(\{m\}) \mid m \in M\}$$

  *Ext* é l' insieme di tutti i gruppi di base extension che sono caratterizzate da ogni attributo. Ogni riga[12] della Tabella 2.3 è tradotta nell' insieme di base extension che posseggono quell' attributo.

- definizione degli insiemi *ConI* e *ConE* (in Tabella 3.11)

  $$ConI: = \{(extent(I), I) \mid I \in Int\}$$

  *ConI* e *ConE* sono insiemi di *concept*[13] cioé di coppie: ogni coppia é costituita da un insieme di attributi e di *base extension.* In particolare ad ogni elemento di *Int* (costituito da un insieme di attributi) si associa l' insieme delle base extension che posseggono almeno tutti gli attributi dell' insieme considerato[14].

  $$ConE: = \{(E, intent(E)) \mid E \in Ext\}$$

  Ad ogni insieme di estensioni di *Ext* si associa l' insieme degli attributi che caratterizzano almeno tutte le base extension dell' insieme considerato[15].

[11] Non vengono replicati gli insiemi di attributi nel caso che due base extension siano caratterizzate dai medesimi

[12] Non vengono replicati gli insiemi di base extension nel caso presentino i medesimi attributi

[13] Un *concept* si definisce in un *context* (G, M, I) ed è una coppia $(A, B) \in \wp(G) \times \wp(M)$ tale che A corrisponda all' insieme di tutti gli oggetti di G che hanno almeno gli attributi di B e B corrisponda all' insieme di tutti gli attributi di M posseduti da almeno tutti gli oggetti in A.

[14] Dato un insieme di attributi B, extent(B) indica un insieme di oggetti g di G che posseggono tutti gli attributi di B

[15] Dato un insieme A di oggetti, intent(A) indica un sottoinsieme degli attributi di M che siano posseduti da tutti gli oggetti di A

$$Int: = \left\{ \begin{array}{l} \{name, rank, faculty, year, tax\_fee, student\_code\}, \\ \{name, rank, faculty, year, tax\_fee, takes, student\_code\}, \\ \{name, rank, faculty, year, tax\_fee, works, takes, student\_code, pay\}, \\ \{name, rank, faculty, year, tax\_fee, works, student\_code, pay\}, \\ \{name, works, pay\}, \\ \{name, faculty, works, pay\}, \\ \{name, faculty\}, \\ \{name, rank, faculty, relation, e\_mail, works, takes, pay\}, \\ \{name, rank, relation, e\_mail, works, takes, pay\}, \\ \{name, rank, faculty, relation, e\_mail, works, title, takes, pay\} \end{array} \right\}$$

$$Ext: = \left\{ \begin{array}{l} \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}, \\ \{1, 2, 3, 4, 5, 6, 10, 11, 12\}, \\ \{1, 2, 3, 4, 5, 6, 8, 9, 10, 12\}, \\ \{1, 2, 3, 4, 5, 6\}, \\ \{10, 11, 12\} \\ \{4, 5, 6, 7, 8, 10, 11, 12\}, \\ \{12\}, \\ \{2, 4, 10, 11, 12\} \end{array} \right\}$$

Tabella 3.10: Insiemi Int ed Ext

$$ConI: = \left\{ \begin{array}{l} (\{name, rank, faculty, year, tax\_fee, student\_code\}, \{1,2,3,4,5,6\}), \\ (\{name, rank, faculty, year, tax\_fee, takes, student\_code\}, \{2,4\}), \\ (\{name, rank, faculty, year, tax\_fee, works, takes, student\_code, pay\}, \{4\}), \\ (\{name, rank, faculty, year, tax\_fee, works, student\_code, pay\}, \{4,5,6\}), \\ (\{name, works, pay\}, \{4,5,6,7,8,10,11,12\}), \\ (\{name, faculty, works, pay\}, \{4,5,6,8,10,12\}), \\ (\{name, faculty\}, \{1,2,3,4,5,6,8,9,10,12\}), \\ (\{name, rank, faculty, relation, e\_mail, works, takes, pay\}, \{10,12\}), \\ (\{name, rank, relation, e\_mail, works, takes, pay\}, \{10,11,12\}), \\ (\{name, rank, faculty, relation, e\_mail, works, title, takes, pay\}, \{12\}) \end{array} \right\}$$

$$ConE: = \left\{ \begin{array}{l} (\{name\}, \{1,2,3,4,5,6,7,8,9,10,11,12\}), \\ (\{name, rank\}, \{1,2,3,4,5,6,10,11,12\}), \\ (\{name, faculty\}, \{1,2,3,4,5,6,8,9,10,12\}), \\ (\{name, rank, faculty, year, tax\_fee, student\_code\}, \{1,2,3,4,5,6\}), \\ (\{name, rank, relation, e\_mail, works, takes, pay\}, \{10,11,12\}) \\ (\{name, works, pay\}, \{4,5,6,7,8,10,11,12\}), \\ (\{name, rank, faculty, relation, e\_mail, works, title, takes, pay\}, \{12\}), \\ (\{name, rank, takes\}, \{2,4,10,11,12\}) \end{array} \right\}$$

Tabella 3.11: Insiemi ConI e ConE

- definizione dell' insieme *Con* e delle nuove classi

$$Con: = ConI \cup ConE$$

  La classe *Con*, in Tabella 3.12, si ottiene dall' unione di *ConI* e *ConE*. Ognuna delle sue coppie rappresenta una nuova classe le cui estensione ed intensione sono rappresentate rispettivamente dall'insieme delle base extension e degli attributi che costituiscono la coppia in esame.

- calcolo delle relazioni di specializzazione fra le classi

  Le relazioni di specializzazione sono memorizzate in una matrice quadrata M avente per righe e colonne le nuove classi. Un '1' indica che la classe della riga specializza quella della colonna (in Tabella 3.13).

- rimozione delle relazioni di specializzazione transitive

  Nella matrice M sono specificate tutte le specializzazioni comprese quelle transitive (se A specializza B e B specializza C allora ci sarà un '1' ad in-

$$Con: = \left\{ \begin{array}{l} C1 = (\{name\}, \{1,2,3,4,5,6,7,8,9,10,11,12\}), \\ C2 = (\{name, rank\}, \{1,2,3,4,5,6,10,11,12\}), \\ C3 = (\{name, faculty\}, \{1,2,3,4,5,6,8,9,10,12\}), \\ C4 = (\{name, rank, faculty, year, tax\_fee, student\_code\}, \{1,2,3,4,5,6\}), \\ C5 = (\{name, works, pay\}, \{4,5,6,7,8,10,11,12\}), \\ C6 = (\{name, rank, takes\}, \{2,4,10,11,12\}) \\ C7 = (\{name, faculty, works, pay\}, \{4,5,6,8,10,12\}), \\ C8 = (\{name, rank, faculty, year, tax\_fee, works, student\_code, pay\}, \{4,5,6\}), \\ C9 = (\{name, rank, faculty, relation, e\_mail, works, takes, pay\}, \{10,12\}), \\ C10 = (\{name, rank, relation, e\_mail, works, takes, pay\}, \{10,11,12\}), \\ C11 = (\{name, rank, faculty, year, tax\_fee, takes, student\_code\}, \{2,4\}), \\ C12 = (\{name, rank, faculty, year, tax\_fee, works, takes, student\_code, pay\}, \{4\}), \\ C13 = (\{name, rank, faculty, relation, e\_mail, works, title, takes, pay\}, \{12\}) \end{array} \right\}$$

Tabella 3.12: Insiemi Con delle Classi Virtuali

dicare anche che A specializza C), ma per ottenere la gerarchia a noi interessano solo quelle fra subclasse e classe specializzata (A specializza B e B specializza C). Per ottenere queste relazioni si calcola la matrice di Tabella 3.14 definita da: N=M-MxM: dove gli '1' indicano relazioni dirette di specializzazione.

**5.** Dalla matrice N si ricavano le relazioni di specializzazione fra le classi e conseguentemente la gerarchia di Figura 2.6. Se necessario, si procede all' eventuale raffinamento della gerarchia ottenuta eliminando le classi con intensione od estensione nulla (C7,C8,C9,C11,C12,C1).

### 3.3.2 Considerazioni

Il metodo illustrato realizza la fusione delle gerarchie utilizzando delle informazioni sulle estensioni delle classi, che devono essere note a priori: tutto il lavoro si appoggia sulla possibilità di conoscere le relazioni insiemistiche fra le estensioni, se queste non sono disponibili la fusione risulta impraticabile.

In generale si usa la conoscenza estensionale per creare una partizione dell' insieme complessivo delle entità[16] rappresentate nelle sorgenti. Ogni sottoinsieme (base extension) costituente la partizione contiene entità distinte non contenute dagli altri sottoinsiemi. Le proprietà delle entità in ogni partizione non sono

[16]La differenza fra *oggetto* ed *entità* è che il primo termine è vincolato alle proprietà che vengono descritte per quell' elemento, mentre il secondo non si interessa di altro se non dell' esistenza dell' oggetto a prescindere dagli attributi che lo descrivono

| $\prec$ | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C3** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C4** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C5** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C6** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C7** | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C8** | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C9** | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| **C10** | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C11** | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C12** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| **C13** | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

Tabella 3.13: Matrice M di Specializzazione

uniche per ogni suo elemento, ma sono definite da un insieme massimo di attributi (Attribute-Extension Relation) ricavati dall' unione di tutti gli attributi delle classi la cui intersezione forma la base extension: questo consente di considerare come oggetti della estensione base sia un oggetto di una delle classi che la costituiscono, sia un oggetto virtuale ottenuto fondendo oggetti provenienti da due o più classi che partecipino alla base extension. Una volta ottenute queste informazioni, per creare le classi della gerarchia, si devono considerare gli insiemi di attributi propri di ogni gruppo di oggetti: si forma una classe per ogni insieme di attributi cui corrisponda una base extension. Nel caso pessimo si crea un numero massimo di classi pari alla somma del numero degli attributi e delle base extension. Questo caso è causato da una forte disomogeneità intensionale ed estensionale fra le sorgenti ed è abbastanza improbabile inquanto l' omogeneità intensionale è assicurata dal metodo complessivo di integrazione di MOMIS, svolto in [25, 26, 18].

**Complessità** La complessità di questo algoritmo vale $O(n^3)$. Illustriamo brevemente come si è arrivati a questa conclusione. Sia dato in input il context: sia M l' insieme degli attributi, G quello delle base extension. Il primo passo comporta la definizione degli insiemi *Int* ed *Ext*: *Int* è l' insieme delle intensioni (insiemi di attributi) di ogni base extension e si ricava esaminando la tabella delle Relazioni Estensioni_Attributi per colonne; viceversa *Ext* è l'insieme delle estensioni corrispondenti ad ogni attributo m di M e si ricava esaminando per righe la stessa tabella. Dovendo eseguire due cicli for innestati, uno sugli attributi, l'altro sulle base extension, la complessità di calcolo è $O(n^2)$ dove $n = max(\|G\|, \|M\|)$. Il calcolo degli insiemi *ConI* e *ConE* ha complessità $O(n^3)$. Consideriamo ad

| $\prec$ | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **C1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C3** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C4** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C5** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C6** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C7** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C8** | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C9** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| **C10** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C11** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **C12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **C13** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Tabella 3.14: Matrice N di Specializzazione non Transitiva

esempio il calcolo di *ConI*: esso contiene al più n elementi (uno per colonna se $n = \|M\|$), ognuno di essi deve essere confrontato contro ogni colonna per vedere se la base extension corrispondente è caratterizzata dalle proprietà in esame: questo confronto implica di esaminare gli attributi di ogni riga per cui l' insieme è computato dopo al massimo $O(n^3)$ confronti. Idem per *ConE*.

# Capitolo 4

# Progetto e realizzazione del software

Oltre allo studio delle metodologie e degli algoritmi usati per il calcolo delle *base extension* e della gerarchia estensionale, in questa tesi é stato progettato e sviluppato il software che implementa sia gli algoritmi descritti nel capitolo precedente che un interfaccia grafica che permette al progettista l'inserimento degli assiomi estensionali e la gestione di tutta la conoscenza estensionale.

Verranno descritte di seguito le classi java implementate, le loro caratteristiche e funzioni. Infine si dará una piccola dimostrazione del funzionamento del software, si giustificheranno le scelte implementative e si indicherá quali sviluppi ci si aspetta in futuro.

## 4.1 Il modello software per la gestione della conoscenza estensionale

In figura 4.1 é illustrato una schema funzionale di alto livello che descrive l'insieme delle attivitá che caratterizzano il sistema MOMIS.
In particolare é stato evidenziato il modulo **EHB**(Extensional Hierarchy Builder) che ha come obiettivo la creazione delle *base extension* e della gerarchia estensionale. Questa fase viene svolta interagendo con il progettista che definisce gli assiomi estensionali. É stata prevista anche una fase di revisione dello schema globale in quanto, come giá specificato, gli assiomi estensionali partecipano alla definizione dei cluster.

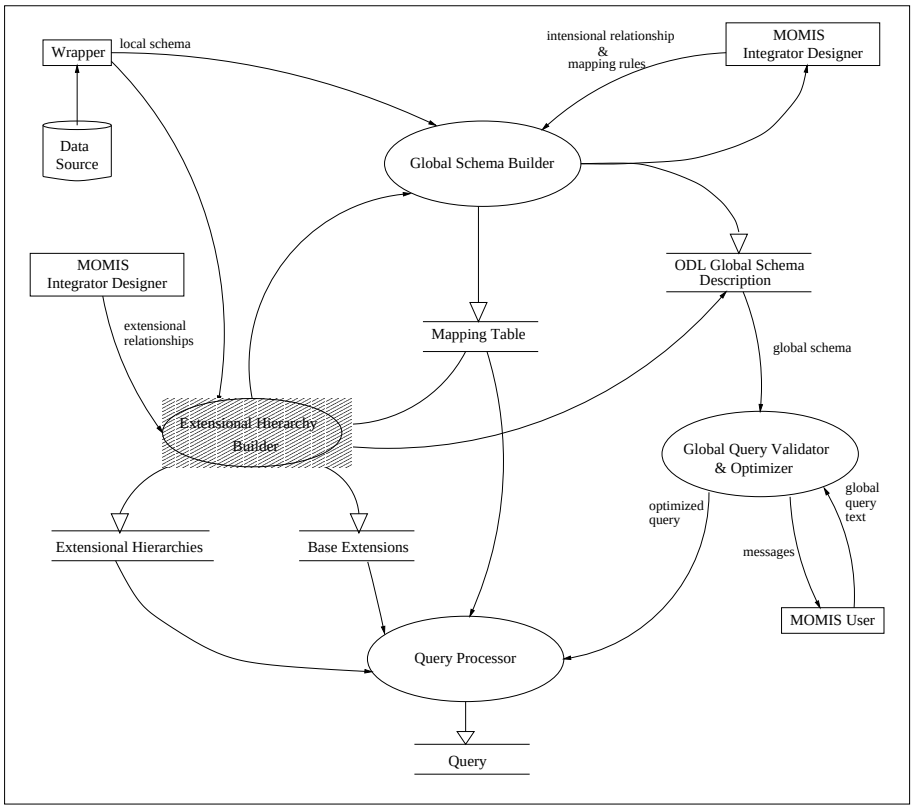


Figura 4.1: Schema funzionale del Mediatore MOMIS

Per poter calcolare le strutture dati necessarie il modulo EHB deve aver accesso ad alcune strutture dati, in particolare deve conoscere:

- la mapping table e la classi globali (contenute nel package *globalschema* [15])

- la descrizione delle sorgenti ricavata dai wrapper (contenuta nel package *odli3*)

Gli algoritmi che realizzano il calcolo delle *base extension* e della gerarchia estensionale sono stati implementati all'interno del package *EXTM*.
Verrá ora illustrato il procedimento adottato, ponendo particolare attenzione all'analisi del package *EXTM* che implementa gli algoritmi descritti nel Capitolo 3.

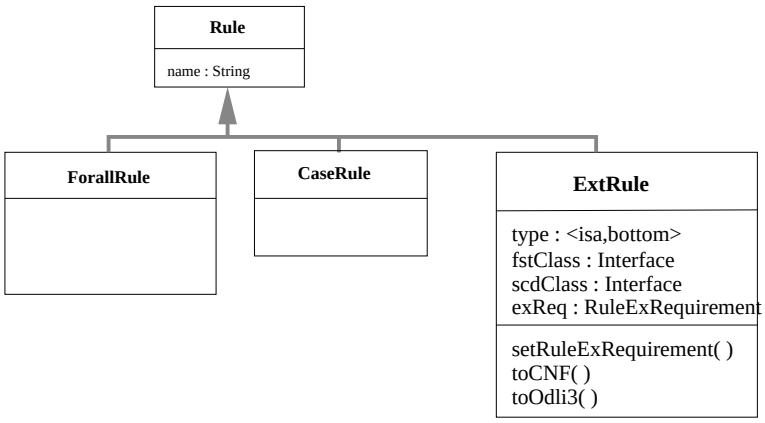


Figura 4.2: Modello ad oggetti della classe ExtRule

## 4.2 Gestione degli assiomi estensionali

Tutti gli assiomi estensionali specificati dal progettista o ricavati dallo schema delle sorgenti vengono memorizzati all'interno della classe *ExtRule*(contenuta nel package *odli3*), mostrata in Figura 4.2.

La classe *ExtRule* é stata definita come sottoclasse della classe *Rule*, in questa gerarchia vengono memorizzati tutti i tipi di regole, sia intensionali che estensionali, definibili dal progettista sugli schemi delle sorgenti. Analizziamo piú in dettaglio questa classe:

- **Proprietá**
  - *type*: specifica il tipo di assioma, puó essere un assioma di disgiunzione (BOTTOM) o un assioma di inclusione (ISA). L'assioma di equivalenza viene espresso attraverso due assiomi di inclusione.
  - *fstClass,scdClass*: specifica le due classi coinvolte dalla rule.
  - *exReq*: specifica i *requisiti di esistenza* che la rule implica.
- **Metodi**
  - *setRuleExRequirement( )*: in base alla rule calcola i *requisiti di esistenza*, che verranno poi memorrizzati in un apposita classe all'interno del package *EXTM*.
  - *toCNF( )*: calcola la forma normale congiuntiva della rule
  - *toOdli3( )*: restituisce la rule scritta secondo il formalismo $odl_{I^3}$

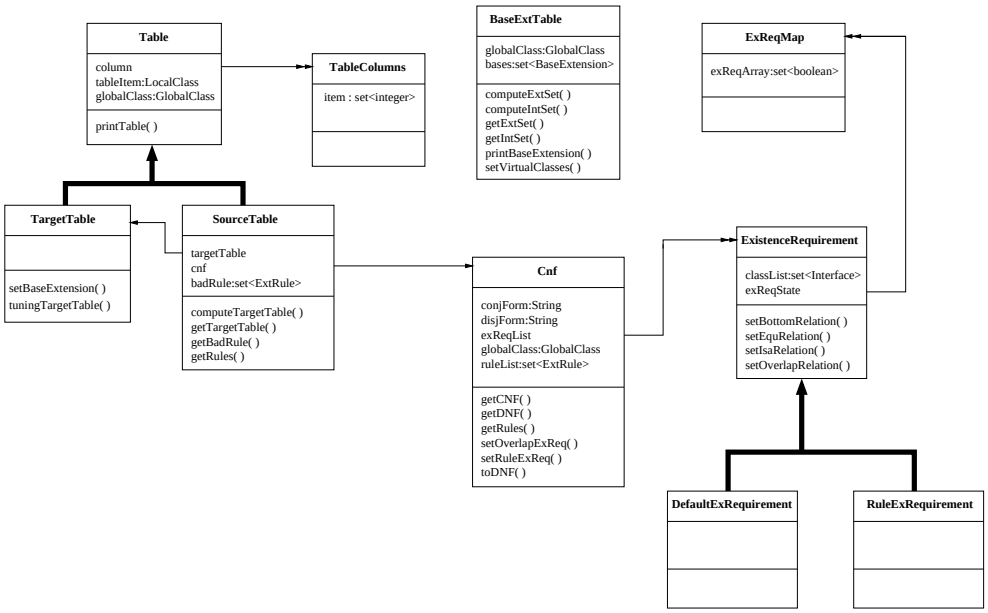


Figura 4.3: Modello ad oggetti del package EXTM

Gli assiomi di inclusione e equivalenza implicano anche un legame intensionale tra le classi coinvolte. Questo legame si esprime attraverso un assioma intensionale che verrá registrato nel *Common Thesaurus*(all'interno del package *odli3*).

Passiamo ora ad analizzare il package *EXTM* che contiene le classi che, prendendo in input gli assiomi registrati, implementano il calcolo delle *base extension* e della gerarchia estensionale.

## 4.3 Il package *EXTM*

In Figura 4.3 viene presentato il modello ad oggetti del package *EXTM*. Il calcolo delle *base extension* e della gerarchia estensionale deve essere ripetuto per ogni classe globale, questo porta a due importanti considerazioni:

- Il calcolo puó essere fatto solo dopo aver definito i cluster e la mapping table;
- Occorre, tutte le volte che si esegue il calcolo, fare riferimento ad una specifica classe globale;

Tutte le informazioni legate alla composizione delle classi globali: classi locali componenti, mapping table, *base extension*, gerarchia estensionale vengono memorizzate nel package *globalschema* [15, 22].

Di seguito vengono, per ogni classe di Figura 4.3, elencate le proprietá ed i metodi principali:

### 4.3.1 Classe “Table”

Implementa le tabelle *Source Table* e *Target Table* usate per il calcolo dell’insieme delle *base extension*. La superclasse *Table* definisce alcuni attributi e metodi comuni:

- **Proprietá**
  - *column*: rappresenta le colonne della tabella, specificate dalla classe *TableColumn* che ha un unico attributo *item* che implementa un vettore di interi.
  - *tableItem*: specifica le righe della tabella, rappresentate da un vettore di classi locali
  - *globalClass*: specifica la classe globale alla quale la tabella si riferisce.
- **Metodi**
  - *printTable( )*: stampa la tabella sullo standard output.

Nel caso della *Source Table* la relativa classe specifica alcune proprietá aggiuntive e diversi metodi:

- **Proprietá**
  - *targetTable*: specifica la *Target Table* che si ottiene applicando l’algoritmo di calcolo
  - *cnf*: si riferisce all’istanza della classe “Cnf” dove sono memorizzate le varie forme normali disgiuntive e congiuntive per il calcolo della *Target Table*
  - *badRule*: lista delle eventuali rule estensionali trovate inconsistenti
- **Metodi**
  - *computeTargetTable( )*: Calcola la relativa *Target Table* in base alla *Source Table* di partenza

  - *getTargetTable( )*: restituisce la *Target Table* calcolata
  - *getBadRule( )*: restituisce la lista delle eventuali rule trovate incoerenti
  - *getRules( )*: restituisce la lista delle rule definite sulla classe globale di riferimento

Mentre la classe “TargetTable” non contiene ulteriori attributi oltre a quelli specificati dalla sua superclasse, vengono specificati solamente alcuni metodi aggiuntivi:

- **Metodi**
  - *setBaseExtension( )*: questo metodo é molto importante, in quanto in base alla *Target Table* costruisce le *base extension* generando le istanze della classe “BaseExtension”(contenuta nel package *globalschema* [15, 22])
  - *tuningTargetTable( )*: gestisce la caselle eventualmente rimaste vuote al termine del calcolo della *Target table*

### 4.3.2 Classe “Cnf”

Questa classe, in base agli assiomi estensionali definiti su una classe globale, gestisce il calcolo delle forme CNF e DNF. In seguito calcola l’insieme dei *requisiti di esistenza* necessari per l’elaborazione della *Source Table*.

- **Proprietá**
  - *conjForm*: stringa che rappresenta la forma CNF delle rule definite sulla classe globale di riferimento
  - *disjForm*: stringa che rappresenta la forma DNF delle rule definite sulla classe globale di riferimento
  - *exReqList*: lista dei *requisiti di esistenza* calcolati in base agli assiomi definiti
  - *globalClass*: classe globale di riferimento
  - *ruleList*: lista degli assiomi estensionali definiti tra classi appartenenti alla classe globale di riferimento. Viene ricavata dalla lista di tutti gli assiomi(nella classe “ExtRule”)

- **Metodi**
  - *getCNF( )*: restituisce la forma CNF degli asssiomi definiti
  - *getDNF( )*: restituisce la forma DNF degli asssiomi definiti

- *getRules( )*: restituisce la lista degli asiomi specificati dal progettista sulla classe globale di riferimento
- *setOverlapExReq( )*: calcola i *requisiti di esistenza* di default, cioé quelli generati dagli assiomi di intersezione
- *setRuleExReq( )*: in base alla lista degli assiomi definiti sulla classe globale calcola i relativi *requisiti di esistenza*
- *toDNF( )*: trasforma la forma CNF nella relativa forma DNF, compiendo le necessarie semplificazioni

### 4.3.3 Classe “ExistenceRequirement”

Le classi di questa gerarchia sono utilizzate per il calcolo dell’insieme dei *requisiti di esistenza* indispensabili per ricavare l’insieme delle *base extension*. Per maggiore chiarezza i *requisiti di esistenza* sono stati partizionati in due classi differenti:

- “RuleExRequirement”: generati dagli assiomi definiti dal progettista;
- “DeafaultExRequirement”: generati dagli assiomi di default di sovrapposizione;

Entrambe queste classi hanno un unica superclasse, la classe “ExistenceRequirement”, che definisce le proprietá ed i metodi usati:

- **Proprietá**
  - *classList*: lista delle classi locali coinvolte dal*requisito di esistenza*. Le classi che compongono un *existence requirement* sono sempre due, ma é stata utilizzata una lista aperta che prevede, in caso di sviluppi futuri, la definizione di *existence requirement* che coinvolgono piú classi
  - *exReqState*: vettore di classi “ExReqMap”, utilizzato per definire quali tipi di *requisiti di esistenza* sono stati definiti tra le classi specificate nella proprietá *classList*. Supponiamo, ad esempio, che sia dato l’assioma:

    $$\text{A } DISJ_{Ext} \text{ B}$$

    devono quindi essere generati i *requisiti di esistenza*:

$$A\overline{B}, \overline{A}B$$

Nella proprietá *classList* vengono memorizzate le classi A e B. Mentre l'attributo *exReqState* contiene due istanze della classe "ExReqMap". Ogni istanza della classe "ExReqMap" rappresenta il singolo *requisito di esistenza* e contiene un vettore di boolean(proprietá *exReqArray*) che indica se le relative classi locali(registrate nell'attributo *classList*) sono presenti dirette o negate. Ritornando all'esempio, i due vettori che vengono registrati sono:

$$[1, 0], [0, 1]$$

La scelta di questo tipo di implementazione é stata obbligata se si tiene in considerazione la possibilitá, accennata in precedenza, di poter gestire *requisiti di esistenza* formati da un numero qualunque di classi.

- **Metodi**
  - *setBottomRelation( )*: definisce il vettore appropriato di "ExReqMap" nel caso sia stata specificata un assioma di disgiunzione
  - *setEquRelation( )*: definisce il vettore appropriato di "ExReqMap" nel caso sia stata specificata un assioma di equivalenza
  - *setIsaRelation( )*: definisce il vettore appropriato di "ExReqMap" nel caso sia stata specificata un assioma di inclusione
  - *setOverlapRelation( )*: definisce il vettore appropriato di "ExReqMap" nel caso sia stata specificata un assioma di sovrapposizione.

### 4.3.4 Classe "BaseExtTable"

Questa classe é utilizzata per il calcolo delle classi virtuali, applicando l'algoritmo descritto in 3.3:

- **Proprietá**
  - *globalClass*: classe globale alla quale ci si riferisce per il calcolo
  - *bases*: insieme delle *base extension* calcolate in base agli assiomi estensionali definiti
- **Metodi**
  - *computeExtSet( )*: calcola l'insieme *Ext* in base alle *base extension*
  - *computeIntSet( )*: calcola l'insieme *Int* in base alle *base extension*

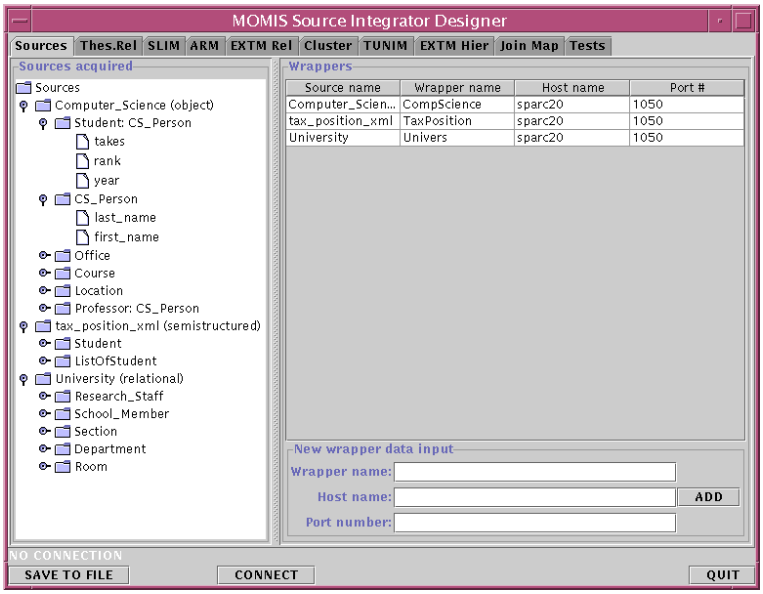


Figura 4.4: SI-Designer

- *getExtSet( )*: ritorna l'insieme *Ext* calcolato
- *getIntSet( )*: ritorna l'insieme *Int* calcolato
- *printBaseExtension( )*: stampa, sullo standard output, l'insieme delle *base extension*
- *setVirtualClasses( )*: calcola l'insieme delle classi virtuali.

## 4.4 L'interfaccia grafica

SI-Designer [21] costituisce l'interfaccia grafica di MOMIS, attraverso la quale il progettista é in grado di interagire col sistema e compiere le operazioni necessarie per realizzare l'integrazione. É costituita da un pannello principale in cui sono inseriti una sequenza di pannelli: uno per ogni fase dell'integrazione. Ogni pannello é selezionabile dal progettista attraverso la selezione di un etichetta che lo identifica(vedi Figura 4.4).

Per implementare l'interazione di SI-Designer con i wrapper è stata adottata l'architettura CORBA: in questo modo i wrapper sono dei *servant-object* (vedi appendice D) che rispondono ai messaggi inviati dal client SI-Designer. La stessa architettura è stata poi adottata anche per utilizzare i tool esterni: WordNet e ODB-Tools sono stati *incapsulati* in oggetti CORBA, rendendo uniforme l'interazione di SI-Designer con il mondo esterno.

Al fine di rendere il sistema più aperto si è pensato di offrire al progettista la possibilità di creare oggetti CORBA contenenti tutte le informazioni relative ad uno schema globale: tali oggetti sono istanza della classe Java **GlobalSchema**. SI-Designer accede agli oggetti **GlobalSchema** attraverso un altro oggetto Java che funge da *proxy*: un oggetto istanza della classe Java **GlobalSchemaProxy** (Figura 4.5). In questo modo, è possibile mettere in rete gli schemi globali di MOMIS e renderli *consultabili* da uno o più Query Manager client oppure, in futuro, da altre applicazioni.
Un oggetto **GlobalSchemaProxy** memorizza comunque tutte le informazioni dello schema globale, rendendo opzionale la creazione di un oggetto CORBA **GlobalSchema**: SI-Designer lascia la scelta al progettista, aumentando così la flessibilità d'utilizzo del tool.

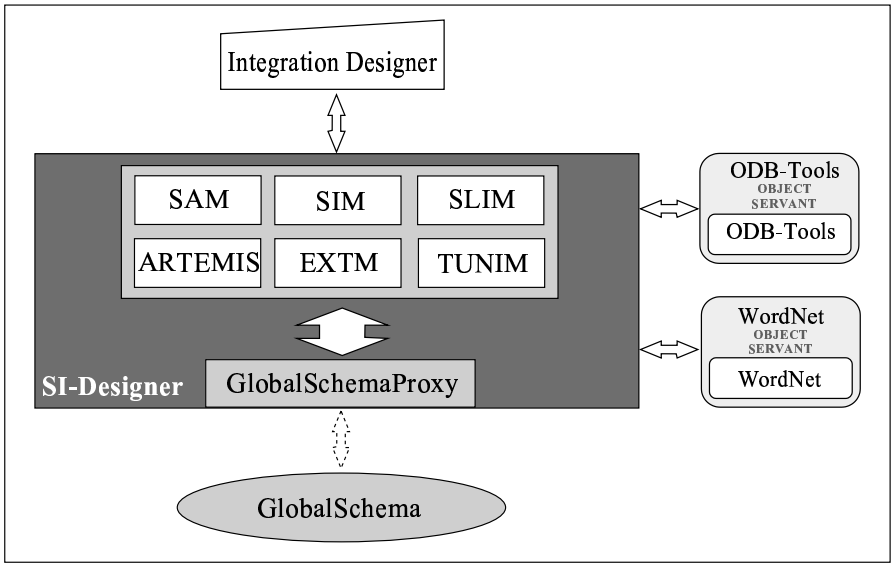


Figura 4.5: L'architettura di SI-Designer

Di seguito verranno presentate le schede che permettono di gestire la conoscenza estensionale e che sono state progettate e implementate nell'ambito di questa tesi.

### 4.4.1 Il pannello "EXTM Rel"

Come é stato giá visto, gli assiomi estensionali di equivalenza e di inclusione implicano un legame intensionale molto forte tra le classi coinvolte. Per questo

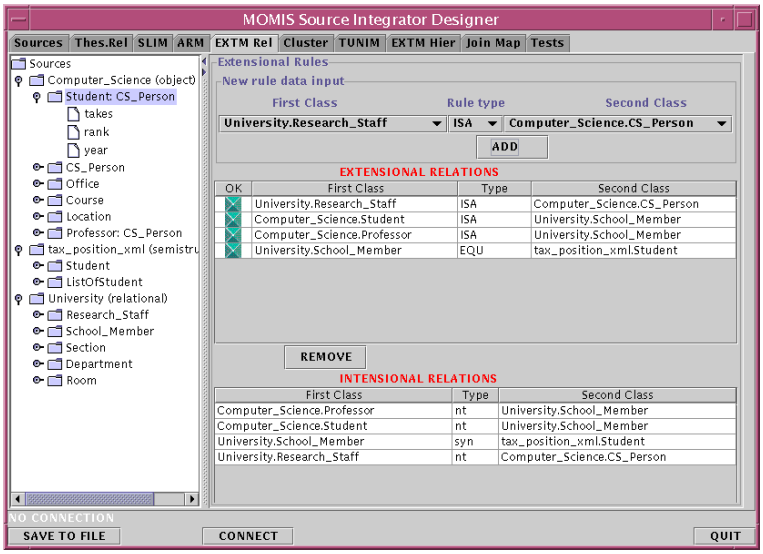


Figura 4.6: SI-Designer: il pannello “EXTM Rel”

motivo due classi tra le quali é stato definito un assioma di questo tipo devono necessariamente essere *forzate* ad appartenere allo stesso cluster.

La scheda “ExtRel”(vedi Figura 4.6) é stata posta prima delle fasi di clustering e di generazione della *mapping table*(TUNIM). In questo pannello il progettista puó specificare degli assiomi estensionali tra tutte le classi delle sorgenti(non sono ancora stati definiti i cluster). Il sistema é in grado di generare automaticamente il relativo assioma intensionale implicato da ogni rule estensionale definita dal progettista.

Le informazioni raccolte da questo pannello saranno utilizzate per la creazione dei cluster, inoltre le rule estensionali saranno memorizzate in modo da poter essere recuperate in seguito per il calcolo delle strutture estensionali.

### 4.4.2 Il pannello “EXTM Hier”

La scheda “EXTM Hier”(Figura 4.7) é stata posizionata dopo la fase di calcolo dei cluster e definizione della *mapping table*(TUNIM). In questo modo é possibile gestire il calcolo delle *base extension* e della gerarchia estensionale considerando una classe globale per volta.

Il pannello a sinistra (Figura 4.7) mostra le classi locali componenti per ogni classe globale. É possibile selezionare la classe globale da esaminare attraverso l’apposita *combo box*, il sistema ricalcola automaticamente le classi componenti, limitando la scelta delle classi locali per la definizione degli assiomi alle sole

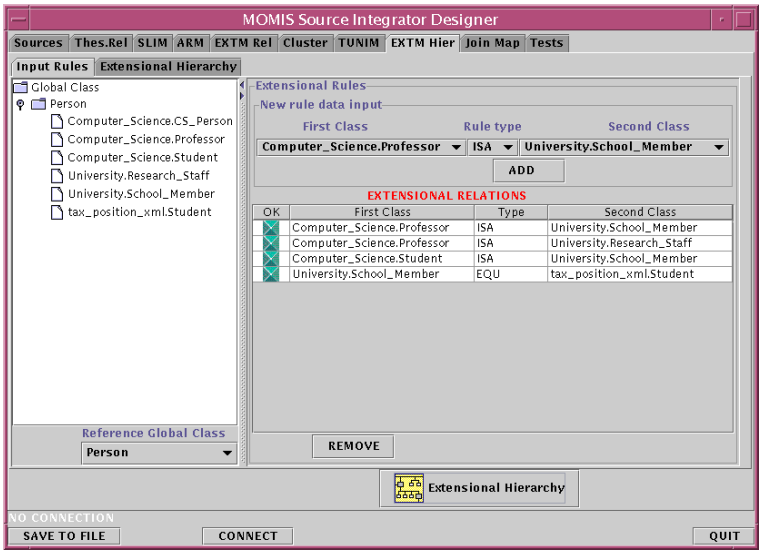


Figura 4.7: SI-Designer: il pannello “EXTM Hier”

classi che appartengono alla classe globale scelta. In questo modo si impedisce al progettista la definizione di assiomi tra classi locali appartenenti a classi globali differenti poiché questo comporterebbe la ridefinizione di tutti i cluster.

Il pannello “EXTM Hier” é stato diviso in due sottopannelli: il primo, denominato “Input Rules”, é utilizzato dal progettista per l’inserimento ed il controllo della consistenza degli assiomi, mentre il secondo, denominato “Extensional Hierarchy”, visualizza tutte le informazioni calcolate.

Una volta specificati gli assiomi, cliccando sul pulsante “Extensional Hierarchy”, il sistema calcola l’insieme delle *base extension* e la gerarchia estensionale che vengono visualizzati (vedi Figura 4.8). Nella parte superiore viene presentata la gerarchia estensionale, cliccando sulle varie classi virtuali (VC1, . . . ,VCn) cambia in modo interattivo il contenuto del pannello inferiore, denominato “Virtual Class Description”(vedi Figure 4.8 e 4.9).

Il pannello inferiore “Virtual Class Description” fornisce le seguenti informazioni:

- Viene rappresentata una serie di pulsanti: uno che identifica la classe virtuale cliccata ed uno per ogni *base extension* che forma l’estensione della classe virtuale selezionata. Cliccando sul pulsante che identifica la classe virtuale od una qualunque delle *base extension* componenti vengono visualizzate le seguenti informazioni:

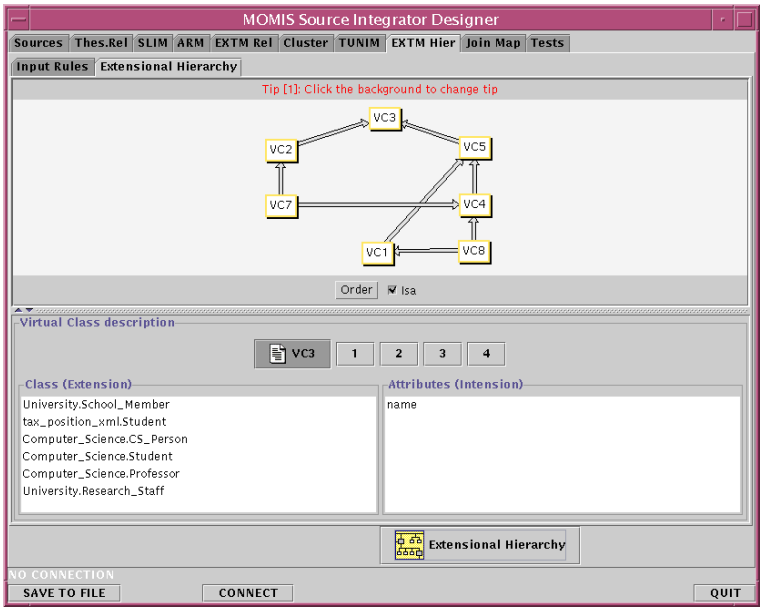


Figura 4.8: Pannello “EXTM Hier”: la gerarchia estensionale

– gli attributi globali della classe virtuale o della *base extension*, cioé gli attributi globali comuni alle classi locali che compongono la classe virtuale o la *base extension* selezionata

– le classi locali che compongono l’estensione della classe virtuale o della *base extension*. Possono essere viste anche come le classi *target* di una ipotetica query posta sulla classe virtuale o sulla *base extension*.

L’algoritmo presentato nel capitolo precedente fornisce anche un supporto per l’individuazione degli assiomi inconsistenti. L’interfaccia grafica é in grado di segnalare al progettista la lista di questo tipo di assiomi. Come mostrato in Figura 4.10 il sistema controlla, ad ogni inserimento di un assioma estensionale, che sia consistente rispetto a quelli giá esistenti, in caso negativo segnala quali sono gli assiomi che sono in contrasto con l’ultimo inserito. In questo caso occorre intervenire modoficando l’insieme degli assiomi per ripristinarne la consistenza.

Occorre osservare che in caso di inconsistenza di alcuni degli assiomi di default(sovrapposizione estensionale), questa non viene segnalata al progettista ma semplicemente non vengono utilizzati gli assiomi che causano tale inconsistenza. Questa scelta é stata fatta per evitare inutile confusione che si creava segnalando al progettista l’inconsistenza di assiomi da lui non definiti.

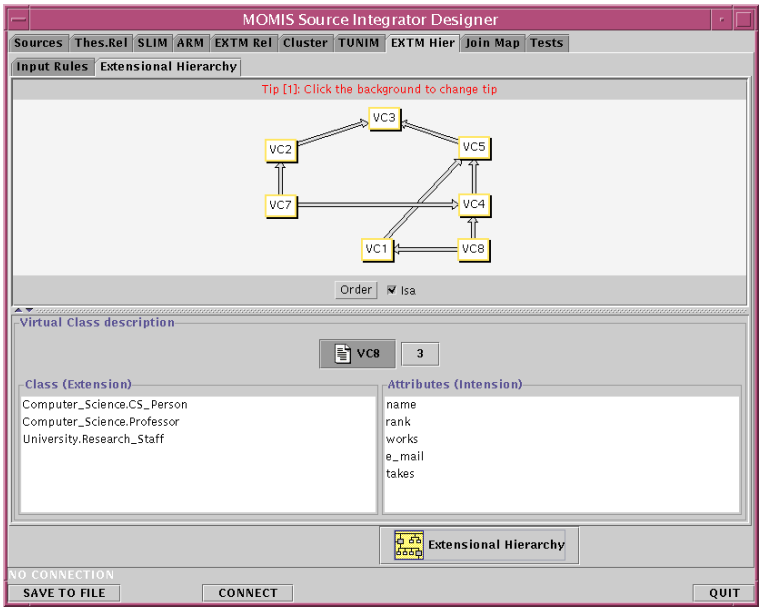


Figura 4.9: Pannello “EXTM Hier”: la gerarchia estensionale

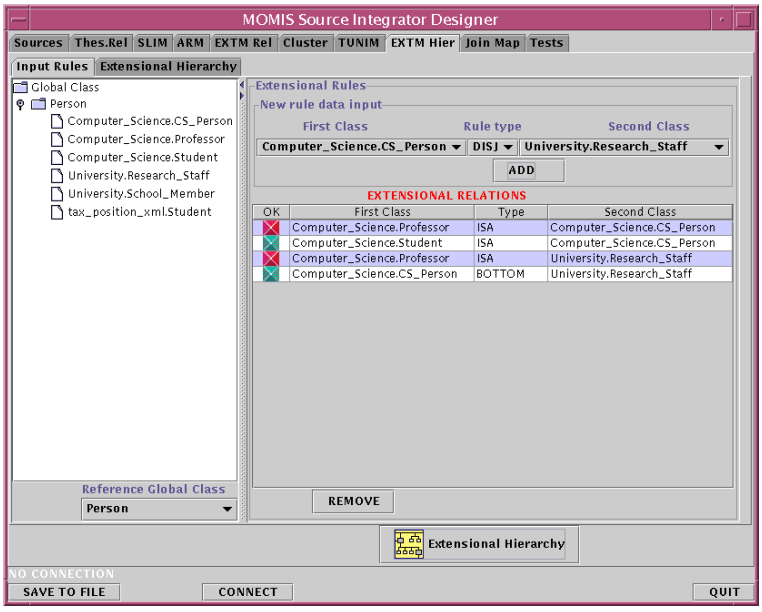


Figura 4.10: Pannello “EXTM Hier”: controllo di consistenza

## 4.5 Il software

Il software prodotto in questa tesi é stato implementato utilizzando la versione **jdk1.2** dell’interprete Java. Approssimativamente sono state prodotte 5000 linee

di codice commentato. I commenti sono stati rtealizzati utilizzando il formalismo richiesto da *Javadoc*, in modo da utilizzare questo componente per la generazione della corrispondente documentazione, i vantaggi sono:

- tutte le informazioni necessarie alla produzione della documentazione sono fornite durante la generazione del codice stesso. In questo modo, aumentano le dimensioni del codice, ma si é in grado di fornire descrizioni piú accurate e dettagliate
- possono essere descritti in modo accurato e completo tutti gli elementi(attributi, costruttori, metodi) dell'interfaccia di ogni classe implementata
- il componente *Javadoc* produce in modo automatico una documentazione in formato HTML con collegamenti ipertestuali che, in modo semplice e veloce, consentono di recuperare tutte le informazioni necessarie

In Appendice C viene riportato il codice relativo ad una delle classi implementate e lacorrispondente documentazione ottenuta mediante *Javadoc*.

Il codice prodotto e la relativa documentazione possono essere trovati nel direttorio */export/home/progetti.comuni/tesi/venuta/sw* del server "*Sparc20*" del dipartimento di Ingegneria.

# Conclusioni

L'obiettivo del progetto MOMIS é quello di realizzare un mediatore in grado di integrare sorgenti eterogenee di dati, al fine di poter interrogare uno schema globale integrato. Questo sistema si differenzia da altri progetti simili per l'automatizzazione della fase di integrazione degli schemi delle sorgenti: partendo da questi ultimi è in grado di ottenere in modo semi-automatico una vista globale sulle sorgenti stesse. È così possibile interrogare una moltitudine di sorgenti formulando delle query sullo schema della vista creata, lasciando al sistema il compito di formulare le interrogazioni per ogni singola sorgente nel linguaggio specifico, eseguendo anche una ottimizzazione dell'accesso. In questo modo l'utente finale é completamente svincolato dalla conoscenza delle singole sorgenti e dalla loro organizzazione e avrá l'impressione di interagire con un unico DataBase. La costruzione dello schema globale é unicamente a carico del progettista.

Il problema della costruzione di uno schema globale, trattato in questa tesi ed anche in lavori precedenti [17, 18, 23, 19, 20] é solo una delle due macrofunzioni individuabili nel sistema MOMIS: la *Creazione dello schema globale*, svolta dal modulo **Global Schema Builder**. Una volta individuato ed ottenuto lo schema globale occorre mettere a disposizione dell'utente finale gli strumenti per interrogarlo, quest'altra macrofunzione di MOMIS é svolta dal modulo **Query Manager**, trattato nei lavori [16, 22, 15]

In questa tesi si é svolta una parte della costruzione dello schema globale, concentrandosi principalmente sulla gestione della conoscenza estensionale, indispensabile in fase di *query processing* poiché permette di generare al contempo una risposta **corretta e completa**. Dapprima sono state analizzate a fondo le problematiche, le caratteristiche e le implicazioni della conoscenza estensionale(Capitolo 2), gestita attraverso l'uso di assiomi logici. In seguito é stato studiato e descritto un algoritmo in grado di calcolare e gestire al meglio gli assiomi estensionali(Capitolo 3). Tutto questo si é materializzato nella realizzazione del software(Capitolo 4) per il calcolo delle strutture dati che rappresentano la conoscenza estensionale (*base extension* e gerarchia estensionale). Per poter ren-

dere facilmente utilizzabile, da parte del progettista tale conoscenza, é stata realizzata un interfaccia grafica composta da due pannelli che sono stati inseriti all'interno di SI-Designer. L'aver scelto una architettura standard, quale è CORBA, per l'interazione con il mondo esterno, costituisce inoltre una buona base di partenza per rendere il sistema MOMIS fruibile anche da parte di altri sistemi di gestione di informazioni. In questo senso si potrebbero riprogettare le strutture dati utilizzate all'interno degli oggetti CORBA, ridefinendole come oggetti CORBA tramite il linguaggio IDL.

Gli sviluppi futuri dovranno essere incentrati soprattutto nelle fase di *ricostruzione delle entitá*, cioé una volta ottenuto un insieme di istanze provenienti da diverse classi locali in risposta ad una query globale, occorre essere in possesso delle informazioni per poter *fondere* tali istanze e ricostruire la descrizione, il piú completa possibile, delle entitá descritte.

# 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'Università di Stanford, nel 1995. Il glossario è 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 è 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 può 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, proprietà, 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 capacità, sui costi . . .

## A.2 Servizi

- Servizio = funzionalità fornita da uno o più 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 molteplicità di risorse, spesso eteorgenee.
- accoppiamento temporale = servizio di mediazione per riconoscere e risolvere differenze nelle unità di misura temporali utilizzate dalle risorse.

- accoppiamento spaziale = servizio di mediazione per riconoscere e risolvere differenze nelle unità 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 velocità 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.
- pubblicità (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) = abilità 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 qualità, rilevanza, costo.
- proprietario = individuo o organizzazione che ha creato, o ha i diritti di un oggetto, e lo può sfruttare.
- proprietario di un servizio = individuo o organizzazione responsabile di un servizio.
- database = risorsa che comprende un insieme di dati con uno schema descrittivo.

- warehouse = database che contiene o dà accesso a dati selezionati, astratti e integrati da una molteplicità 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.
- interoperabilità = capacità di interoperare.
- eterogeneità = incompatibilità 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, unità 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 molteplicità 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, ereditarietà, 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 molteplicità 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 conformità ad una ontologia.
- algebra dell'ontologia = insieme delle operazioni per definire relazioni tra ontologie.
- consistenza temporale = è raggiunta se tutti i dati si riferiscono alla stessa istanza temporale ed utilizzano la stessa granularità temporale.
- specifico ad un dominio = relativo ad un singolo dominio, presuppone l'assenza di incompatibilità semantiche.
- indipendente dal dominio = software, strumento o conoscenza globale applicabile ad una molteplicità di domini.

# Appendice B

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

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

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

```
interface Research_Staff
( source relational University
  extent Research_Staffers
  keys first_name, last_name
  foreign_key dept_code, section_code )
{ attribute string first_name;
  attribute string last_name;
  attribute string relation;
  attribute string e_mail;
  attribute integer dept_code;
  attribute integer section_code; };
```

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

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

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

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

```
interface University_Worker
( source relational University
  extent University_Workers
  keys first_name, last_name
  foreign_key dept_code )
{ attribute string first_name;
  attribute string last_name;
  attribute integer dept_code;
  attribute integer pay;
```

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

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

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

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 Professor : CS_Person
 ( source object Computer_Science
   extent Professors )
{ attribute string title;
  attribute Division belongs_to;
  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 Course
( source object Computer_Science
  extent Courses
  key course_name )
{ attribute string course_name;
  attribute Professor taught_by; };
```

Sorgente `Tax_Position` (a volte abbreviata in TP):

```
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 C

# La classe Java Cnf

Come già accennato nel Capitolo 4, il software prodotto in questa sede è disponibile nella directory /export/home/progetti.comuni/tesi/venuta/sw del server Sparc20 presso il dipartimento di Ingegneria.

## C.1 Il codice

A titolo esemplificativo viene riportato qui di seguito il codice relativo ad una classe Java particolarmente significativa tra quelle implementate: la classe *Cnf*.

**Cnf.java**

```
package odli3.EXTM;
import globalschema.*;
import java.io.*;
import java.util.*;
import odli3.*;
import java.io.Serializable;
//import Parser;
/**
 *This class implements the formula in Disjunctive Normal Form expres
 */
class Cnf extends MomisObject implements Serializable
 {
     /**
      *Reference global class
      */
     GlobalClass globalClass;
```

```
/**
 *Odli3 schema
 */
Schema schema;

/**
 *List of Extensional Rules defined on local classes.
 */
TreeMap ruleList= new TreeMap();

/**
 *Conjunctive normal form
 */
String conjForm;

/**
 *Disjunctive normal form computed from the Conjunctive normal
 *form and simplified using boolean formulas.
 */
String disjForm;

/**
 *List of Existence Requirements defined on reference global class
 */
Vector exReqList=new Vector();


/**
 *@param sch Odli3 schema
 *@param glClass global class of which Disjunctive normal form wanted to
 */
public Cnf(Schema sch,GlobalClass glClass) throws Exception
{
    schema = sch;
    globalClass = glClass;
    exReqList = new Vector();
    Vector classList,vc;
    TreeMap source = new TreeMap();
    String nm = "";
    //ricavo la lista delle classi locali (nomeSorgente.nomeClasse)
    classList = glClass.mappingTable.getClasses();
```

```
//devo separare il nome sorgente dal nome classe
for (int e =0;e< classList.size();e++)
    {
       String  n =(String)classList.elementAt(e);
       //divido il nome sorgente dal nome classe
        String s = "", s1 = "", s2 = "";
        StringTokenizer pt = new StringTokenizer(n,".",false
        //nome sorgente
        s1 = pt.nextToken();
        //nome classe
        s2 = pt.nextToken();
        if (!(source.containsKey(s1)))
            {
                vc = new Vector();
                source.put(s1,vc);
                vc.addElement(s2);
            }
        else
            {
                vc = (Vector)source.get(s1);
                vc.addElement(s2);
            }

    }


Source sr;
TreeMap tr;
ExtRule extR;
String cnf= new String();
Object[] sources= sch.getSources();
for (int i =0;i<sources.length; i++)
    {
        sr = (Source)sources[i];
        if (source.containsKey(sr.name))
            {
               System.out.println("Rule trovate sulla sorgen
                                   +sr.getName());
                //le rule che legano classi locali di sorgen
               //diverse vengono trovate due volte
                tr = sch.getExtRule(sr,false);
```

```
                    //stampa debug
                    System.out.println(tr.toString());
                    //solo se ha trovato almeno una rule
                    if (!(tr.isEmpty()))
                        {
                            for (Iterator f = tr.values().iterator();
                                 f.hasNext();)
                                {

                                    extR = (ExtRule)f.next();
                                    //aggiungo la rule alla lista
                                    //solamente
                                    //se non erano gia state inserite
                                    //e se coinvolge classi locali
                                    //che entrambe appartengono alla
                                    //classe globale passata
                                    //come parametro
                                    if ((!(ruleList.containsValue
                                           (extR))) &&
                                        (classList.contains
                                         (extR.fstClass.
                                          getDottedName()))
                                        && (classList.contains(
                                           extR.scdClass.
                                           getDottedName())))
                                        {

                                //costruisce la forma CNF
                                           cnf=cnf +extR.toCNF();
                                           ruleList.put(extR.name,
                                                        extR);

                                        }

                                }
                        }
                }
            }
        conjForm = cnf;
        //calcolo la forma DNF
        disjForm = toDNF(cnf);
```

```
        //Calcolo gli existence requirement
        setRuleExRequirement();
        //Calcolo exist. req. di default.
        setOverlapExReq();
    }

    /**
     *From CNF compute the DNF
     *If no rule are defined return a NULL value
     *@param cnf A String that represent the CNF
     *@return A String that represent the DNF
     */
     public String toDNF (String cnf)
   {
       String delim,tmp1,tmp2;
       String finale="";
       Vector dnf,temp1,temp2,dnftot;
       temp1=new Vector();
       temp2=new Vector();
       dnf=new Vector();
       //Vettore che contiene alla fine tutti gli elementi della for
       dnftot = new Vector();
       String str = "";
       int i = 0;
       //imposto i delimitatori di stringa
       delim = "()+";
       //Definisco l'analizzatore lessicale
       StringTokenizer tok = new StringTokenizer(cnf,delim,false);
       //riempio il vettore di stringhe con tutte
       //le classi che compaiono nella forma CNF
       while (tok.hasMoreTokens())
           {
               str = tok.nextToken();
               dnf.addElement(str);
           }
       dnf.trimToSize();
       int k = dnf.size();
       //inserisco i primi due elementi della lista
       //se esiste almeno una rule definita
       if (k>=2)
           {
```

```
            tmp1=(String)dnf.elementAt(0);
            tmp2=(String)dnf.elementAt(1);
            temp1.addElement(tmp1);
            temp2.addElement(tmp2);

            //itero su tutti gli altri elementi rimasti
            for (int e=2;e<k;e=e+2)
                {
                    tmp1=(String)dnf.elementAt(e);
                    tmp2=(String)dnf.elementAt(e+1);
                    temp1 = duplicateAndAdd(temp1,tmp1,tmp2);
                    temp2 = duplicateAndAdd(temp2,tmp1,tmp2);
                }
            //Eseguo le operazioni di semplificazione
            temp1=reduce(temp1);
            temp2=reduce(temp2);
            //Sommando il contenuto di temp1 e temp2 ottengo
            //la stringa finale
            for (int y=0;y<temp1.size();y++)
                {
                    String s;
                    s=(String)temp1.elementAt(y);
                    if (finale=="")
                        finale = finale +s;
                    else
                        finale = finale +"+" +s;
                    dnftot.addElement(s);
                }
            for (int y=0;y<temp2.size();y++)
                {
                    String s;
                    s=(String)temp2.elementAt(y);
                    if(finale=="")
                        finale = finale +s;
                    else
                        finale = finale +"+" +s;
                    dnftot.addElement(s);
                }
        }
            //Stringa che contiene la forma DNF semplificata
            return finale;
```

```
    }


     /**
      *Print the list of Existence Requirements
      */
public void printExRequirement()
    {
        for (int r=0;r<exReqList.size();r++)
            {
                ExistenceRequirement req = (ExistenceRequirement)
                   exReqList.elementAt(r);
                Interface intr = (Interface)req.classList.elementAt(0
                String str = (String)intr.getName();
                intr = (Interface)req.classList.elementAt(1);
                str = str +"  " +(String)intr.getName();
                int q=str.length();
                String blank= " ";
                q = 35-q;
                for (int w=0;w<q;w++)
                    blank = blank +" ";
                System.out.print(str +blank +"\t");
                for (int h=0;h<req.exReqState.size();h++)
                    {
                        ExReqMap mp = (ExReqMap)req.exReqState.elemen

                        System.out.print(mp.exReqArray.toString() +"\
                    }
                System.out.println();
            }
    }

      //metodo interno che dato un vettore di stringhe
      //ne raddoppia la capacita
      //e duplica i dati contenuti e poi somma la stringa
      //st1 alla prima meta del vettore
      // e somma st2 alla seconda meta del vettore
     private Vector duplicateAndAdd(Vector vct,String st1,String st2)
    {
        Vector ret = new Vector();
        String value;
```

```
        ret = vct;
        Vector fin = new Vector();
        ret.trimToSize();
        int k=ret.size();
        for (int h=0;h< k;h++)
            {
                value = (String)ret.elementAt(h);
                ret.addElement(value);
            }
        ret.trimToSize();
        k=ret.size();
        //trovo il punto medio del vettore
        k=(k/2);
        for (int g=0;g<k;g++)
            {
                value = (String)ret.elementAt(g);
                value = value +"*" +st1;
                fin.addElement(value);
            }
        for (int g= k;g<(k*2);g++)
            {
                value = (String)ret.elementAt(g);
                value = value +"*" +st2;
                fin.addElement(value);
            }
        return fin;
    }
     //Metodo privato che presa in input un vettore con tutti gli elementi
     //che compongono la forma DNF la semplifica utilizando
     //regole di logica booleana
     private Vector reduce(Vector stred)
    {
        Vector vec;
        vec = stred;
        Vector ret =new Vector();
        Vector elementList = new Vector();
        String st="";
        String sts="";
        boolean complement =false;
        String element="";
        StringTokenizer strtok;
```

```
//Controllo la formula : Idempotenza A*A = A
//                       Complemento A*~A = false
//                       Identita    A*false = false
for (int x=0;x<vec.size();x++)
    {
        st = (String)vec.elementAt(x);
        strtok = new StringTokenizer(st,"*",false);
        elementList=new Vector();
        while (strtok.hasMoreTokens())
            {
                element = strtok.nextToken();
                //stampa per il debug
                //System.out.println(element);
                elementList.addElement(element);
            }
        //devo confrontare tra loro tutti gli elementi regist
        for (int g=0;g<elementList.size();g++)
            {
                String strn =(String)elementList.elementAt(g)
                //Devo confrontare la stringa contenuta
                //in strn con tutte quelle
                //registrate in elementList tranne se stessa
                for (int a=g+1;a<elementList.size();a++)
                    {
                        element=(String)elementList.elementAt
                        //Idempotenza A*A = A
                        if (element.equals(strn))
                            {
                                elementList.setElementAt("nul
                                break;
                            }
                        //Complemento A*~A=false
                        //Identita A*false=false
                        if (element.charAt(0)=='~')
                                 {
                                String sub = element.substrin
                                if(sub.equals(strn))
                                    {
                                       //ret.addElement("fals
                                       //posso uscire
                                       //da tutti i cicli
```

```
                                        elementList.
                                           removeAllElements();
                                        complement = true;
                                        break;
                                     }
                                  }
                           //Complemento ~A*A=false
                           //Identita false*A=false
                           if (strn.charAt(0)=='~')
                               {
                                   String sub = strn.substring(1);
                                   if(sub.equals(element))
                                       {
                                          //ret.addElement("false");
                                          //posso uscire da
                                          //tutti i cicli
                                          elementList.
                                             removeAllElements();
                                          complement = true;
                                          break;
                                       }
                               }
                       }
                   if (complement)
                   {
                      complement=false;
                      break;
                   }
            }
        for (int s=0;s<elementList.size();s++)
        {
           element=(String)elementList.elementAt(s);
           if (element.equals("null"))
              continue;
           if(sts=="")
              sts = element;
           else
              sts = sts +"*" +element;
        }
        if (!(elementList.isEmpty()))
        {
```

```
                    ret.addElement(sts);
                    sts="";
                 }
              }
          return ret;
    }
       /**
      *Generate the default existence requirement from
      *overlap rules (default rules for local class
      *of a given global class) and add it to the list exReqList
      */
public void setOverlapExReq() throws Exception
       {
        //lista delle classi locali che compongono la classe globale
          Vector list =  globalClass.mappingTable.getClasses();
          Vector listInterface = new Vector();
          Interface swap,log;
          Source src;
          DefaultExRequirement def;
          for (int g=0;g<list.size();g++)
          {
             String item = (String)list.elementAt(g);
             StringTokenizer strTok = new StringTokenizer(item,".",fa
             //Nome sorgente
             String s1 = strTok.nextToken();
             //nome classe
             String s2 =strTok.nextToken();
             src = (Source)schema.getSource(s1);
             swap = (Interface)src.getInterface(s2);
             //creo una lista delle Interface che formano la classe g
             listInterface.addElement(swap);
             //stampa debug
             //System.out.println("Interfacce" +listInterface.toStrin
          }
          for(int y=0;y<listInterface.size();y++)
          {
             swap = (Interface)listInterface.elementAt(y);
             for(int e=(y+1);e<listInterface.size();e++)
             {
                log = (Interface)listInterface.elementAt(e);
                //se le due interface non sono legate da una rule
```

```
               if(!(checkRule(swap,log)))
               {
                  def = new DefaultExRequirement(swap,log,"OVERLAP");
                              //aggiungo exist req alla lista
                  exReqList.addElement(def);
               }
            }
         }
      }

      /**
       *Scan the rule extensional list
       *and generate the Existence requirements for every rule
       */
    public void setRuleExRequirement()
   {
      ExtRule testRule1,testRule2;
      RuleExRequirement exRequir;
      TreeMap list = (TreeMap)ruleList.clone();
      Interface inf1,inf2,tar1,tar2;
      boolean find = false;
      //se e' stata definita almeno una rule
      if(!(list.isEmpty()))
      {
         for(Iterator w=list.values().iterator();w.hasNext();)
         {
            find=false;
            testRule1=(ExtRule)w.next();
            if (testRule1.type == "BOTTOM")
            {
               exRequir = testRule1.setRuleExRequirement();
                              //Aggiungo ex req. alla lista
               exReqList.addElement(exRequir);

            }
            //Se trova due rule ISA tra le stesse classi
            //deve creare una rule EQU
            if (testRule1.type == "ISA")
            {
               inf1= (Interface)testRule1.fstClass;
               inf2 = (Interface)testRule1.scdClass;
```

```
                for (Iterator y=list.values().iterator();y.hasNext();
                                        {
                                           testRule2=(ExtRule)y.next();
                                           tar1=(Interface)testRule2.fstC
                                           tar2=(Interface)testRule2.scdC
                                           if((tar1.equals(inf2)) &&
                                              (tar2.equals(inf1)) &&
                                              ((String)testRule2.type=="I
                                           {
                                              exRequir = new RuleExRequir
                                                 (inf1,inf2,"EQU");
                                              exReqList.addElement(exRequ
                                              //entrambe le rule ISA punt
                                              //allo stesso ex. req.
                                              testRule1.exReq = exRequir;
                                              testRule2.exReq = exRequir;
                                              find=true;
                                           }



                                        }
                                     //e' una rule ISA non EQU
                if(!(find))
                {
                   exRequir = testRule1.setRuleExRequirement();
                   //Aggiungo ex req. alla lista
                   exReqList.addElement(exRequir);

                }
             }
          }
       }
    }
       //Metodo privato che date due classi locali mi dice
       //se esiste una o piu'
       //rule che lega le due classi
private boolean checkRule(Interface cl1,Interface cl2)
       {
          ExtRule rul;
          Interface in1,in2;
```

```
          for (Iterator f=ruleList.values().iterator();f.hasNext();)
          {
             rul = (ExtRule)f.next();
             in1 = (Interface)rul.fstClass;
             in2 = (Interface)rul.scdClass;
             if (((in1.equals(cl1)) && (in2.equals(cl2))) ||
                 ((in1.equals(cl2))&&(in2.equals(cl1))))
                return true;
          }
          return false;
       }

       /**
        *Return the DNF form
        */
public String getDNF()
       {
          return disjForm;
       }
       /**
        *Return the CNF form
        */
public String getCNF()
       {
          return conjForm;
       }
/**
 *return the list of extensional rules defined on global class
 */
public TreeMap getRules()
       {
          return ruleList;
       }
}
```

## C.2 La documentazione

Come si può notare dal listato riportato nella sezione precedente, numerosi commenti sono stati redatti rispettando un preciso formalismo. Questo permette, uti-

lizzando il comando *javadoc*, di ottenere una significativa documentazione in formato HTML.
Più precisamente il comando che permette di produrre la documentazione JavaDoc relativa a tutte le classi è:

```
javadoc -d doc ‘find . -name ’*.java’‘
```

Nella subdirectory *doc* sono adesso contenuti tutti i documenti HTML relativi al software.

In questa sezione viene presentato un esempio di documentazione, sempre relativo alla classe Cnf.java; come si potrà notare sono presenti solo i metodi definiti *public*, se si volesse invece la descrizione di tutti bisognerebbe aggiungere l’opzione `-private` nel comando precedentemente riportato.

# Appendice D

# L'architettura CORBA

CORBA [53] (*Common Object Request Broker Architecture*) è un'architettura standard, distribuita e ad oggetti sviluppata dall'Object Management Group (OMG). Dal 1989 l'obiettivo del gruppo OMG è stato la progettazione di una architettura per un *software bus* aperto, chiamato *Object Request Broker* (ORB), sul quale oggetti diversi potessero interagire via rete, indipendentemente dal sistema operativo in cui sono stati implementati. Questo standard permette a più oggetti di invocarne altri senza conoscerne l'esatta locazione o in quale linguaggio sono stati implementati.
Il linguaggio utilizzato per definire un oggetto CORBA è l'IDL (*Interface Definition Language*) mentre gli ORB comunicano attraverso il protocollo standard IIOP (*Internet InterORB Protocol*), definito sempre dall'OMG.

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

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

### Come funziona

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

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

1. il client invoca un metodo dell’oggetto CORBA utilizzando l’object reference;

2. la stub function puntata dall’object reference identifica, attraverso l’ORB locale, la macchina sulla quale si trova il servant-object CORBA, che è in attesa di ricevere messaggi;

3. l’ORB locale chiede all’ORB remoto di stabilire una connessione con l’oggetto CORBA;

4. ottenuta la connessione, l’ORB locale manda all’ORB remoto l’object reference della stub function e i parametri per il metodo da invocare;

5. l’ORB remoto passa la richiesta di esecuzione del metodo, assieme ai parametri, al servant-object che eseguirà il metodo invocato;

6. i risultati ed eventuali eccezioni vengono ritornate all’ORB locale lungo lo stesso percorso.

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

### Il Naming Service

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

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

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

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

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

### La creazione di oggetti CORBA in Java

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

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

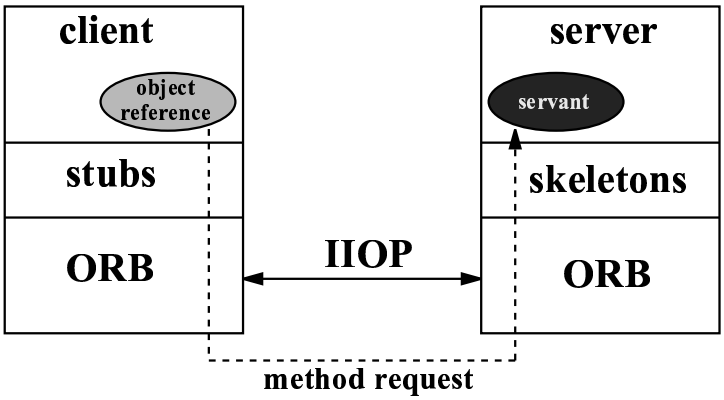


Figura D.1: La invocazione di un metodo di un oggetto CORBA remoto

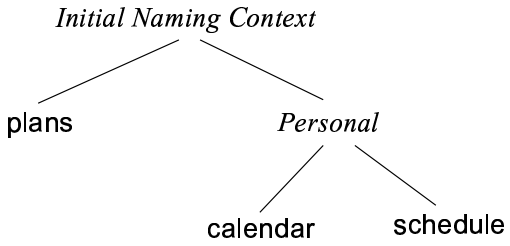


Figura D.2: Esempio di albero creato dal naming server

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

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

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

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

# Bibliografia


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

[2] 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.

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

[4] 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.

[5] Alon Levy, Dana Florescu, Jaewoo Kang, Anand Rajaraman, and Joanne J. Ordille. The information manifold project.

[6] S. Chawathe, Garcia Molina, H., J. Hammer, K.Ireland, Y. Papakostantinou, J.Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *IPSJ Conference, Tokyo, Japan*, 1994. ftp://db.stanford.edu/ pub/chawathe/1994/ tsimmis-overview.ps.

[7] H. Garcia-Molina et al. The tsimmis approach to mediation: Data models and languages. In *NGITS workshop*, 1995.

[8] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. IEEE TSE, 1984.

[9] Y.Papakonstantinou H.Garcia-Molina and J.Ullman. Medmaker: a mediation system based on declarative specification. Technical report, Stanford University, 1995.

[10] H. Garcia-Molina Y.Papakonstantinou. Object fusion in mediator systems (extended version). 1997.

[11] 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.

[12] 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.

[13] 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.

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

[15] Andrea Zaccaria. MOMIS: Il componente Query Manager. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[16] Alberta Rabitti. Architettura di un mediatore per un sistema di integrazione di sorgenti distribuite ed autonome. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[17] Alberto Zanoli. Si-designer, un tool di ausilio all'integrazione di sorgenti di dati eterogenee distribuite: progetto e realizzazione. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

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

[19] Giovanni Malvezzi. Estrazione di relazioni lessicali con WordNet nel sistema MOMIS. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1999-2000.

[20] Francesco Guerra. MOMIS: il wrapper per sorgenti di dati XML. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1999-2000.

[21] Rossano Guidetti. SI-Designer: un tool per l'integrazione di sorgenti distribuite ed eterogenee. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1999-2000.

[22] M. Franceschi. Il componente query manager di momis: utilizzo della conoscenza estensionale. Tesi di Laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 2000.

[23] Gianni Pio Grifa. Analisi di affinità strutturali fra classi $ODL_{I^3}$ nel sistema MOMIS. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[24] Maurizio Vincini. Utilizzo di tecniche di intelligenza artificiale nell'integrazione di sorgenti informative eterogenee. Tesi di dottorato, Università di Modena, Facoltà di Ingegneria, dottorato di ricerca in Ingegneria Informatica, 1997-1998.

[25] 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*.

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

[27] Arpa $i^3$ reference architecture. Available at http://dc.isx.com/I3/.

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

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

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

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

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

[33] 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.

[34] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. a description logics based tool for schema validation and semanti query optimization in object oriented databases. Technical report, sesto convegno AIIA, 1997.

[35] I. Schmitt and C. Türker. An Incremental Approach to Schema Integration by Refining Extensional Relationships. In G. Gardarin, J. French, N. Pissinou, K. Makki, and L. Bougamin, editors, *Proc. of the 7th ACM CIKM Int. Conf. on Information and Knowledge Management, November 3–7, 1998, Bethesda, Maryland, USA*, pages 322–330, New York, 1998. ACM Press.

[36] 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.

[37] A.G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[38] S. Castano and V. De Antonellis. Deriving global conceptual views from multiple information sources. In *preProc. of ER'97 Preconference Symposium on Conceptual Modeling, Historical Perspectives and Future Directions*, 1997.

[39] M.R. Cohen and E. Nagel. An itroduction to logic. Indianapolis, Indiana: Hackett Publishing Company, 1993.

[40] 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.

[41] I. Schmitt and G. Saake. Merging inheritance hierarchies for database integration. *IEEE Trans. on Knowledge and Data Engineering*.

[42] C. Carpineto and G. Romano. Galois: An order-theoretic approach to conceptual clustering. In *Macine Learning Conference*, pages 33–40, 1993.

[43] Byacc/java. Available at http://www.lincom-asg.com/ rjamison/byacc/.

[44] A. Cerfogli MOMIS: stato di lavori. Available in the directory /export/home/progetti.comuni/Momis/documenti/ of the sparc20.dsi.unimo.it Server.

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

[46] S.Bergamaschi, S.Castano, S.De Capitani di Vimercati, S.Montanari, and M.Vincini. An intelligent system for the integration of semistructured and strucurer data. *Submitted for: Information Systems. special Issue on Semistructured Data*.

[47] S.Bergamaschi, S.Castano, D.Beneventano, and M.Vincini. Semanitc integration and query of heterogenous information sources. *Journal of Data and Knowledge Engineering 1,* 1999.

[48] Norman Walsh. What is XML-QL? Available at http://www.xml.com/pub/98/10/guide0.html

[49] Jon Bosak. XML, Java, and the future of the Web. Available at http://www.xml.com/xml/pub/r/XML,_Java,_and_the_future_of_the_Web/

[50] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu. A Query Language for XML. Available at http://www.w3.org/TR/NOTE-xml-ql/

[51] A. Layman, E. Jung, E. Maler, H. S. Thompson, J. Paoli, J. Tigue, N. H. Mikula, S. De Rose. XML-Data. Available at http://www.w3.org/TR/1998/NOTE-XML-data/

[52] Introduction to CORBA Available at http://developer.java.sun.com/developer/onlineTraining/corba/corba.html

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