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

Elaborazione di interrogazioni in un sistema multi-database

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Konstantin Eugeniev Kostenarov

Correlatore
Prof. Ing. Domenico Beneventano
Dott.ssa Federica Mandreoli

Anno Accademico 2001 - 2002

Parole chiave:

Query Processing
Mediator
FullDisjunction
Natural-OuterJoin
Query Rewriting

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi, il Dott. Ing. Domenico Beneventano, Dott.ssa Federica Mandreoli e tutti i ragazzi del gruppo MOMIS per l'aiuto fornito e la costante collaborazione.

Un grazie tutto speciale va alla mia ragazza Simona per avermi appoggiato in tutti momenti difficili.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

La crescita spasmodica e disordinata di dati reperibili e di sorgenti di informazione avvenuta negli ultimi anni, ha comportato una maggiore difficoltà nell'interrogazione e nel riconoscimento dei dati realmente utili da parte degli utenti. Questo problema è dovuto principalmente all'eterogeneità delle sorgenti, diverse le une dalle altre e disomogenee anche al loro interno, e dal fenomeno dell'information overload, cioè dall'abbondanza di informazioni disordinate e molto spesso duplicate. Diventa dunque difficile per l'utente effettuare una interrogazione tale da portare a buon esito la ricerca di dati desiderata, tenendo conto che la diffusione di massa dell'utilizzo della rete Internet accresce la necessità di facilitare l'accesso e il reperimento delle informazioni anche da parte di utenti poco specializzati e quindi non in grado di formulare interrogazioni rivolte alle singole sorgenti di informazione per poi trasformare i risultati parziali nella risposta desiderata. Da qui nasce dunque l'esigenza di progettare meccanismi in grado di interrogare sorgenti eterogenee e di selezionare le informazioni utili in modo automatizzato tale da facilitare l'utente nella sua ricerca. In tale direzione si dirige il progetto MOMIS (Mediator EnvirOnment for Multiple Information Sources) in cui questa Tesi si inserisce, il cui scopo è dunque quello di integrare sorgenti eterogenee e distribuite in modo tale da permettere all'utente di accedere ad una risposta esaustiva senza dover possedere conoscenze specifiche sulla natura delle sorgenti da cui viene estrapolata l'informazione cercata.

L'obiettivo della presente tesi è stata l'analisi delle tecniche di Query Processing e lo studio di algoritmi di elaborazione, esecuzione ed ottimizzazione algebrica di interrogazioni complesse. Questo studio ha portato alla definizione di *regole di riscrittura* e di algoritmi, che permettono di gestire, in modo semi-automatico, le informazioni.

La tesi è organizzata nel seguente modo:

Nel **Capitolo 1** viene introdotto il concetto di Integrazione Intelligente delle Informazioni, descrivendo l'architettura di riferimento I^3 e la struttura di un Mediatore. Si descrivono anche le scelte implementative adottate per il sistema MOMIS, con particolare riguardo per la sua architettura e per i tools utilizzati. Viene fatta inoltre una breve introduzione nelle specifiche dei linguaggi ODL_{I^3} e

OQL_{I^3}.

Nel **Capitolo 2** descriviamo la fase di integrazione degli schemi, illustrando le strutture dei dati rappresentanti la conoscenza intensionale ed estensionale prodotta. Inoltre, viene data una visione d'insieme della fase di Query Processing, senza entrare nei particolari, mostrando come questa sfrutti la conoscenza prodotta dalla fase di integrazione degli schemi.

Nel **Capitolo 3** introduciamo le basi della gestione delle query: Query Manager. Successivamente viene definito il concetto di Full Disjunction e presentate le definizioni e le tecniche dell'ottimizzazione algebrica di cui gli algoritmi di riscrittura delle query fanno uso.

Nel **Capitolo 4** si descrivono due algoritmi alternativi per il calcolo della Full Disjunction in casi di grafi cilici e aciclici dopodichè viene presentato l'algoritmo di riscrittura delle query.

Il **Capitolo 5** presenta lo stato attuale di Query Processing in MOMIS entrando nei particolari dei passi di integrazione intensionale ed estensionale e definizione del Piano di Esecuzione.

Il **Capitolo 6** realizza una panoramica sullo stato dell'arte: si provvede ad esporre approcci adottati da sistemi affini a MOMIS, nell'affrontare le problematiche inerenti al processo di Query Processing.

Sono presenti, inoltre, quattro appendici: in Appendice A viene riportato un glossario dei termini usati in ambito I^3; in Appendice B viene mostrata la grammatica ODL_{I^3}; in Appendice C viene illustrato l'esempio in ODL_{I^3}, che sarà utilizzato come riferimento; in Appendice D viene mostrata la grammatica OQL_{I^3}; in Appendice E è riportata la restrizione della grammatica OQL_{I^3} alle BasicQuery.

Capitolo 1

Integrazione Intelligente delle Informazioni ed il sistema MOMIS

L'accrescersi continuo di fonti di informazione eterogenee, l'eterogeneità stessa dei dati per quanto riguarda la loro natura, nonchè l'abbondanza di dati disordinati tra cui scernere le informazioni interessanti (information overload), hanno fatto si che gli standard esistenti (TCP/IP, ODBC, OLE, CORBA, SQL, ecc.) non siano più sufficienti alla risoluzione di tali problematiche in quanto se da un lato risolvono i problemi relativi alle diversità hardware e software, non si preoccupano della modellazione delle informazioni. Diventa dunque pressochè impossibile da parte di un utente poco esperto, come lo può essere il pubblico sempre più vasto di Internet, isolare i dati necessari per una ricerca fruttuosa.

In questo ambito, vari progetti stanno studiando nuovi sistemi di supporto alle decisioni (DSS, Decision Support System), di integrazione di Data Base eterogenei, i datawarehouse (magazzino di dati), e di isolamento delle applicazioni e dei dati integrati dalle sorgenti (repository indepenence).

1.1 Sistemi I^3

Integration Information (I^2) si differenzia dai metodi sopraccitati in quanto si prefigge di combinare tra di loro informazioni provenienti da intere sorgenti o da parti di esse basandosi sulle descrizioni dei dati senza dover ricorrere alla loro duplicazione fisica. Per selezionare le informazioni in modo da ottenere risultati utili è necessaria dunque conoscenza ed intelligenza applicabili alla scelta delle sorgenti e dei dati, alla loro fusione e sintesi. Se a ciò si aggiunge poi l'utilizzo di una Intelligenza Artificiale (IA) che sfrutta le conoscenze acquisite, si arriva non ad una semplice aggregazione di informazioni, ma ad un accrescimento del loro

valore, ottenendo nuove informazioni dai dati ricevuti: possiamo parlare allora di Intelligent Itegration of Information (I^3).

1.1.1 Il Programma I^3

Fondato e sponsorizzato dall'ARPA, l'agenzia che fa capo al Dipartimento di difesa americano, dal 1992, questo ambizioso progetto propone l'introduzione di architetture modulari sviluppabili secondo i principi proposti da uno standard che ponga le basi dei servizi da soddisfare dall'integrazione ed abbassi i costi di sviluppo e mantenimento. A questo scopo si riutilizzano tecnologie già sviluppate in quanto questo è più semplice e meno oneroso. Per la riusabilità è fondamentale l'esistenza di interfacce e procedure standard. Per questo motivo è stata proposta una partizione dei servizi e delle risorse che si articola su due dimensioni:

- orizzontalmente, divisa in tre livelli: livello utente, moduli intermedi che fanno uso di tecniche di IA, risorse di dati;
- verticalmente: diversi domini in cui raggruppare le sorgenti.

i domini nei vari livelli si scambiano dati ed informazioni a livello dell'utilizzatore benchè non strettamente connessi. In questo livello sono presenti vari moduli: I^3 si concentra sul livello intermedio della partizione, mediando tra gli utilizzatori e le sorgenti.
In questo livello sono presenti diversi moduli, tra cui:

- **Facilitator** e **Mediator**: (le cui diversità sono esigue e poco chiare in letteratura) hanno il compito di ricercare le fonti interessanti e di combinare i dati da esse ricevute;
- **Query Processor**: riformula le query aumentando le possibilità di successo di quest'ultime;
- **Data Miner**: analizza i dati per estrarre informazioni intensionali implicite.

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

1.1.2 Architettura di riferimento per sistemi I^3

L'obbiettivo principale del programma I^3 è apportare una considerevole diminuzione dei tempi di realizzazione di un integratore di informazioni, raccogliendo e strutturando le soluzioni prevalenti nel campo della ricerca. Alla base di tale

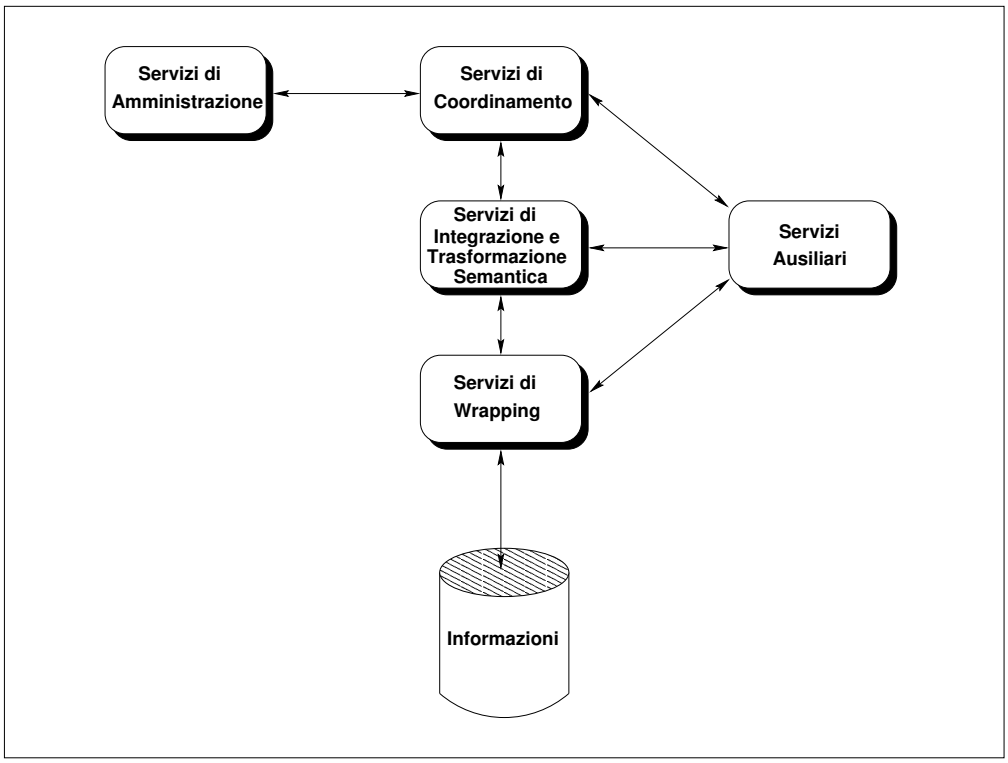

Figura 1.1: Diagramma dei servizi I^3

progetto si trovano la difficoltà di ricerca delle informazioni fra le molteplici sorgenti che le contengono e la complessità del processo di integrazione dovuto al fatto che le fonti di informazione e i sistemi informativi pur essendo correlati fra loro, non lo sono in forma semplice ne tanto meno preordinata.

I problemi che la Tecnologia I^3 deve risolvere

Passiamo ora all'individuazione dei problemi relativi al programma I^3 ai quali è necessario trovare una soluzione. L'architettura del sistema I^3 esalta l'importanza dei servizi di coordinamento che esercitano due ruoli principali: localizzano altri servizi I^3 e fonti di informazioni che possono essere utilizzate per costruire il sistema stesso; individuano ed invocano a run-time gli altri servizi necessari a dare risposta ad una specifica richiesta di dati. Il sistema I^3 individua cinque famiglie di attività omogenee unitamente ai loro legami. Due sono le chiavi di lettura: la prima segue l'asse orizzontale mentre la seconda quella verticale mettendo in evidenza i diversi aspetti e compiti del sistema.

Percorrendo l'asse orizzontale si nota il rapporto tra i servizi di coordinamento e di amministrazione ai quali spetta il compito di mantenere informazioni sulle capacità delle sorgenti (che tipo di dati possono fornire e come vanno interrogate). Seguendo invece l'asse verticale si intuisce come avviene lo scambio di informazioni nel sistema: i servizi di wrapping provvedono ad estrarre le informazioni dalle singole sorgenti. Tali informazioni sono poi impacchettate ed integrate dai servizi di integrazione e trasformazione semantica per poi essere passati ai servizi di coordinamento che avevano fatto la richiesta.

I servizi ausiliari invece forniscono funzionalità di supporto e sono responsabili dei servizi di arricchimento semantico delle sorgenti. Analizziamo in modo specifico le funzionalità di ogni servizio.

- **Servizi di Coordinamento**
  Ricoprono un'importanza centrale in quanto coordinano le operazioni da eseguire in fase di progettazione dei link fra i vari domini informativi ed in fase di esecuzione in tempo reale di una richiesta dell'utente. Sono servizi di alto livello che permetto l'individuazione di sorgenti di dati interessanti e costruiscono un'interfaccia con l'utente dandogli l'impressione di trattare con un sistema omogeneo.

  Si possono realizzare integratori di informazioni con diverse caratteristiche e proprietà. Esse possono essere rappresentati da meccanismi che spaziano dalle selezioni dinamiche delle sorgenti (o Brockering per integratori intelligenti) fino al semplice much-making nel quale il mappaggio tra le informazioni locali da integrare va svolto manualmente una volta per tutte. Conformemente col tipo di integratore che si è intenzionati di realizzare, i servizi di Coordinamento possono essere:

  a) **Facilitation e Brokering Services**: forniscono una selezione dinamica delle sorgenti in grado di soddisfare la richiesta dell'utente. Il sistema usa un deposito di metadati per individuare il modulo che può trattare direttamente questa richiesta, in particolare si parla di Brokering quando è coinvolto un modulo alla volta, oppure di Facilitatori o Mediatori se vi sono più moduli interessati. In quest'ultimo caso la query iniziale viene decomposta in un insieme di sottoquery da inviare ai differenti moduli che gestiscono sorgenti distinte. Successivamente vengono integrate le risposte per fornirne una presentazione globale all'utente. Questo viene realizzato utilizzando servizi di Query Decomposition e di tecniche di Inferenza, mutuate dall'Intelligenza Artificiale, per una determinazione dinamica delle sorgenti da interrogare.

b) **Matchmaking**: il mappaggio fra informazioni integrate e locali è effettuato manualmente da un'operatore in fase di inizializzazione. In questo caso tutte le richieste vengono trattate allo stesso modo.

- **Servizi di Amministrazione**
  Questi servizi sono utilizzati da quelli di Coordinamento per localizzare le sorgenti *utili*, determinare la loro capacità, creare ed interpretare i *Template*. I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per realizzare un determinato task. Queste strutture dati servono quindi per ridurre al minimo le possibilità di decisione del sistema, consentendo di definire a priori le azioni da eseguire a fronte di una determinata richiesta.
  Un metodo alternativo all'utilizzo dei Template è l'uso delle *Yellow pages*, ovvero servizi di directory che mantengono le informazioni sul contenuto delle sorgenti e sul loro stato. In questo modo, le Yellow pages consentono al Mediatore di inviare la richiesta di informazioni alla sorgente giusta o, se non fosse disponibile, ad una equivalente.
  Tra questi tipi di servizio vi sono il *Browsering*, che permette all'utente di "navigare" tra le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste, e gli *Iterative Query Formulation* che aiutano l'utente a rilassare o specificare meglio alcuni vincoli dell'interrogazione al fine di ottenere risposte più precise.

- **Servizi di Integrazione e Trasformazione Semantica**
  Questi servizi supportano le manipolazioni semantiche necessarie per l'integrazione e la trasformazione delle informazioni. Hanno in input una o più sorgenti di dati e restituiscono come output la "vista" integrata o trasformata di queste informazioni. Tra questi servizi si distinguono quelli relativi alla trasformazione degli schemi e quelli relativi alla trasformazione dei dati. Spesso sono indicati come servizi di *Mediazione*, essendo tipici dei moduli mediatori. I principali sono:

  – **Servizi di integrazione degli schemi**: supportano la trasformazione e l'integrazione degli schemi e delle informazioni derivanti da fonti di dati eterogenee; creano il vocabolario e le ontologie condivise dalle sorgenti; integrano gli schemi in una vista globale, mantengono il mapping tra schemi globali e sorgenti locali.
  – **Servizi di integrazione delle informazioni**: aggregano, riassumono ed astraggono i dati per fornire presentazioni analitiche significative. Possono inoltre occuparsi di unificare la "granularità" dei dati (come possono essere le discrepanze nelle unità di misura o le discrepanze temporali).

  - **Servizi di supporto al processo di integrazione**: sono utilizzati quando una query deve essere scomposta in più sottoquery da inviare a fonti differenti con la necessità di integrare poi i loro risultati.

- **Servizi di Wrapping**
  I servizi di Wrapping fungono da traduttori dei sistemi locali ai servizi di alto livello dell'integratore e viceversa. Il loro scopo è far si che le fonti di informazione aderiscano ad uno standard.
  I servizi di Wrapping permettono ai servizi di Coordinamento e di Mediazione di manipolare in modo uniforme le sorgenti locali, anche se questi non sono state esplicitamente pensate a far parte del sistema di interrogazione; forniscono un'interfaccia che, seguendo gli standard più diffusi (ad esempio: SQL come linguaggio di interrogazione, CORBA come protocollo di scambio), permette alle sorgenti estratte di essere accedute dal maggior numero possibile di sistemi mediatori.
  I servizi di Wrapping permettono ai servizi di alto livello di essere il più possibile riusabili. Questo permette alle sorgenti estratte da questi wrapper "universali" di essere accedute dal maggior numero possibile di moduli mediatori. In pratica il compito degli wrapper è quello di modificare l'interfaccia, i dati ed il comportamento di una sorgente per facilitare 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.

- **Servizi Ausiliari**
  I Servizi Ausiliari aumentano le funzionalità degli altri servizi descritti precedentemente. Possono svolgere varie funzioni, tra cui: monitoraggio del sistema, propagazione di aggiornamenti, attività di ottimizzazione, etc.

1.1.3 Il mediatore

Secondo la definizione proposta da Wiederhold in [1] 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. Il Mediatore a livello implementativo dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al massimo da pochi esperti.
Un mediatore ha quindi i seguenti compiti:

- 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 come obiettivo la progettazione e la realizzazione di un **Mediatore**, come descritto in [2, 3, 4]. L'ipotesi di avere a che fare esclusivamente con sorgenti di dati strutturati e semistrutturati, ha consentito di restringere il campo applicativo del sistema con una conseguente diminuzione delle problematiche riscontrate in fase di progettazione e realizzazione. L'approccio architetturale scelto è quello *classico*, che si sviluppa su tre livelli principali:

1. *utente*: attraverso un'interfaccia grafica l'utente pone delle interrogazioni su uno schema globale e riceve un'unica risposta, come se stesse interrogando un'unica sorgente di informazioni;

2. *mediatore*: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio di interrogazione) comune a tutte le fonti;

3. *wrapper*: ogni wrapper gestisce una singola sorgente, convertendo le richieste del mediatore in una forma (linguaggio) comprensibile dalla sorgente; successivamente elabora le informazioni estratte da tale sorgente trasformandole nel modello usato dal mediatore.

Facendo riferimento ai servizi descritti nella Sezione 1.1.2, l'architettura del mediatore progettato è riportata in Figura 1.2. In particolare sono stati sviluppati i servizi di Integrazione e Trasformazione Semantica. Inoltre l'impostazione architetturale mostra come il sistema mediatore progettato voglia distaccarsi dall'approccio *strutturale*, cioè sintattico, tuttora dominante tra i sistemi presenti sul mercato [5, 6, 7].
Quando si parla di approccio strutturale, si fa riferimento all'uso di un *self-describing model* per rappresentare gli oggetti da integrare, limitando l'uso delle informazioni semantiche alle regole predefinite dall'operatore. Il sistema non conosce a priori la semantica di un oggetto recuperato da una sorgente, bensì è l'oggetto stesso che, attraverso delle etichette, si autodescrive. Fra le caratteristiche di questo approccio sono la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo e la possibilità di trattare in modo omogeneo dati che descrivono lo stesso concetto o che hanno concetti in comune, ci si basa sulla definizione manuale di regole (rule), che permettino di identificare i termini che devono essere condivisi da più oggetti.

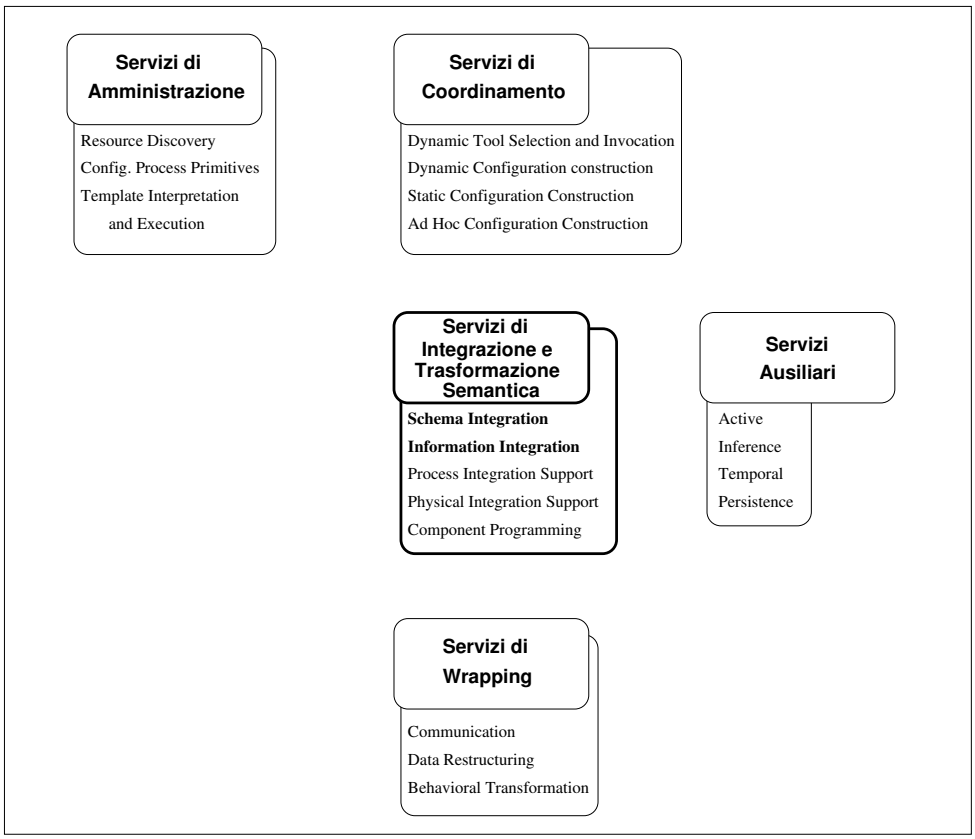

Figura 1.2: Servizi I^3 presenti nel mediatore

Altri progetti, tra cui MOMIS, seguono invece un approccio all'integrazione di tipo *semantico*, che prevede che siano soddisfatti i seguenti punti:

- il mediatore deve conoscere, per ogni sorgente, lo schema concettuale (metadati);
- le informazioni semantiche devono essere 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 le informazioni semantiche che necessariamente ogni schema sottintende, il mediatore può individuare i concetti comuni a più sorgenti e le relazioni che li legano.

1.1.4 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, nè tantomeno è semplice realizzare una loro coerente integrazione. Trascurando le differenze dei sistemi fisici (all'unificazione delle quali dovrebbero provvedere i moduli wrapper) i problemi a livello di mediazione che si è dovuto risolvere (o coi quali si è dovuti scendere a compromessi) sono:

- **Problemi ontologici**
  Come riportato in Appendice A A, per *ontologia* si intende, in questo ambito, "l'insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti". In sostanza con ontologia ci si riferisce a quell'insieme di termini che, in un certo dominio applicativo, denotano una particolare conoscenza e fra i quali non esiste ambiguità perchè 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, ma ci si limita a riportare una semplice classificazione delle ontologie per inquadrare l'ambiente in cui ci si muove. Fra i diversi livelli di ontologia esistenti (*top-level ontology, domain and task ontology, application ontology*, etc ...) [8, 9], ognuno con le proprie problematiche, si è assunto di muoversi all'interno delle *domain ontology*, ipotizzando quindi che tutte le fonti informative condividano almeno i concetti fondamentali (ed i termini con cui identificarli).

- **Problemi semantici**
  Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare, e quindi un insieme di concetti comuni, è improbabile che usino la stessa semantica, cioè gli stessi vocaboli e le stesse strutture dati per rappresentare questi concetti.
  Come riportato in [10] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo reale che persone distinte possono avere, ma non è l'unica. Le differenze nei sistemi di DBMS possono portare all'uso di differenti modelli per la rappresentazione della porzione di mondo in questione: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate attraverso un modello relazionale o ad oggetti. L'obiettivo dell'integratore, che è fornire un accesso integrato ad un insieme di sorgenti, si traduce allora nel non facile compito di identificare i concetti comuni all'interno di queste sorgenti e risolvere le differenze

semantiche che possono essere presenti tra di loro. Possiamo classificare queste contraddizioni semantiche in tre gruppi principali:

1. *eterogeneità tra le classi di oggetti*: benchè 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 possibile 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.

1.2 Architettura generale del sistema MOMIS

Considerando le problematiche descritte nel paragrafo precedente, nonchè alcuni sistemi preesistenti [5, 6, 7, 11, 12, 13, 14, 15, 16, 17, 18] , 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 nell'impiego di un approccio semantico e nell'uso di logiche descrittive per la rappresentazione degli schemi delle sorgenti, elementi che introducono comportamenti intelligenti in grado di rendere semi-automatica la fase di integrazione [2, 3]. Un lavoro approfondito è stato svolto anche riguardo alla fase di *query processing* [19, 20, 21, 22] , cioè per il processo che, dalla query posta sullo schema unificato, provvede a generare automaticamente le sottoquery da inviare alle sorgenti e successivamente 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.

1.2.1 L'approccio adottato

MOMIS adotta un approccio di integrazione delle sorgenti *semantico* e *virtuale* [19]. Il concetto di semantico è stato illustrato nella Sezione1.1.3. Con virtuale [23] 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 poste dall'utente e sull'individuazione delle sorgenti da interrogare per generare delle query locali eseguibili su ogni sorgente inerente all'interrogazione; lo schema globale dovrà inoltre disporre di tutte le informazioni atte al riperimento dei risultati utili per ottenere una risposta corretta, completa e minima e alla fusione adeguata dei risultati ottenuti localmente. L'obbiettivo di questa tesi è proprio la definizione di una serie di regole e algoritmi, che agevolino il sistema nel trattamento delle query ed alla successiva fusione delle informazioni utili.

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

1. la presenza di uno schema globale permette all'utente di formulare qualsiasi interrogazione che sia con esso consistente senza preocuparsi della rappresentazione strutturale e semantica delle sorgenti che contengono l'informazione cercata;

2. le informazioni semantiche che lo schema globale comprende possono contribuire ad una eventuale ottimizzazione delle interrogazioni;

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

4. la vista virtuale rende il sistema estremamente flessibile, in grado cioè di sopportare frequenti cambiamenti sia nel numero che nel tipo delle sorgenti, ed anche nei loro contenuti (non occorre prevedere onerose politiche di allineamento);

Si è deciso di adottare, sia per la rappresentazione degli schemi che per la formulazione delle interrogazioni, un unico modello dei dati basato sul paradigma ad oggetti. Il modello comune dei dati utilizzato nel sistema (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 ODL_{I^3}[19, 3, 4, 21] che si presenta come estensione del linguaggio standard ODL proposto dal gruppo di standardizzazione ODMG-93.

Per quanto riguarda il linguaggio di interrogazione si è adottato OQL_{I^3} che adotta la sintassi OQL senza discostarsi dallo standard. Questo linguaggio risulta estremamente versatile ed espressivo fornendo la possibiltà di sfruttare le informazioni rappresentate nello schema globale.

Inoltre si è cercato di definire uno standard comune di comunicazione tra i vari moduli MOMIS al fine di rendere ancora più agevole l'ampliamento futuro. Si è deciso di adottare lo standard CORBA (Common ORB Architecture) per le comunicazioni tra i moduli [24] . CORBA è una tecnologia per l'integrazione, inoltre è ad oggetti ed una modellazione di questo tipo permette di ridurre la complessità di MOMIS: esistono difatti metodologie consolidate per la rappresentazione e progettazione di sistemi ad oggetti (OMT, UML), ma soprattutto per utilizzare un oggetto è sufficiente conoscerne l'interfaccia pubblica e questo favorisce il lavoro degli sviluppatori successivi.

1.2.2 L'architettura generale di MOMIS

Momis è stato progettato per fornire un accesso integrato ad informazioni eterogenee memorizzate sia in sorgenti strutturate, come database relazionali, database ad oggetti e semplici file, sia in sorgenti semistrutturate, come le sorgenti descritte in XML.

Come si può vedere nella Figura 1.3, i componenti del sistema MOMIS sono disposti su tre livelli:

- **Livello Dati**
  A questo livello si trovano i **Wrapper**. Posti al di sopra di ciascuna sorgente, sono i moduli che fungono da interfaccia tra il mediatore vero e proprio e le sorgenti locali di dati. Le funzioni da loro svolte sono:
  - in fase di integrazione forniscono una descrizione delle informazioni contenute nelle sorgenti, utilizzando il linguaggio ODL_{I^3}.
  - in fase di Query Processing, traducono la query ricevuta dal mediatore (espressa in OQL) in una interrogazione comprensibile ed eseguibile dalla sorgente stessa. Inoltre, devono esportare i dati ricevuti in risposta all'interrogazione, presentandoli al mediatore attraverso il modello ODL_{I^3}.

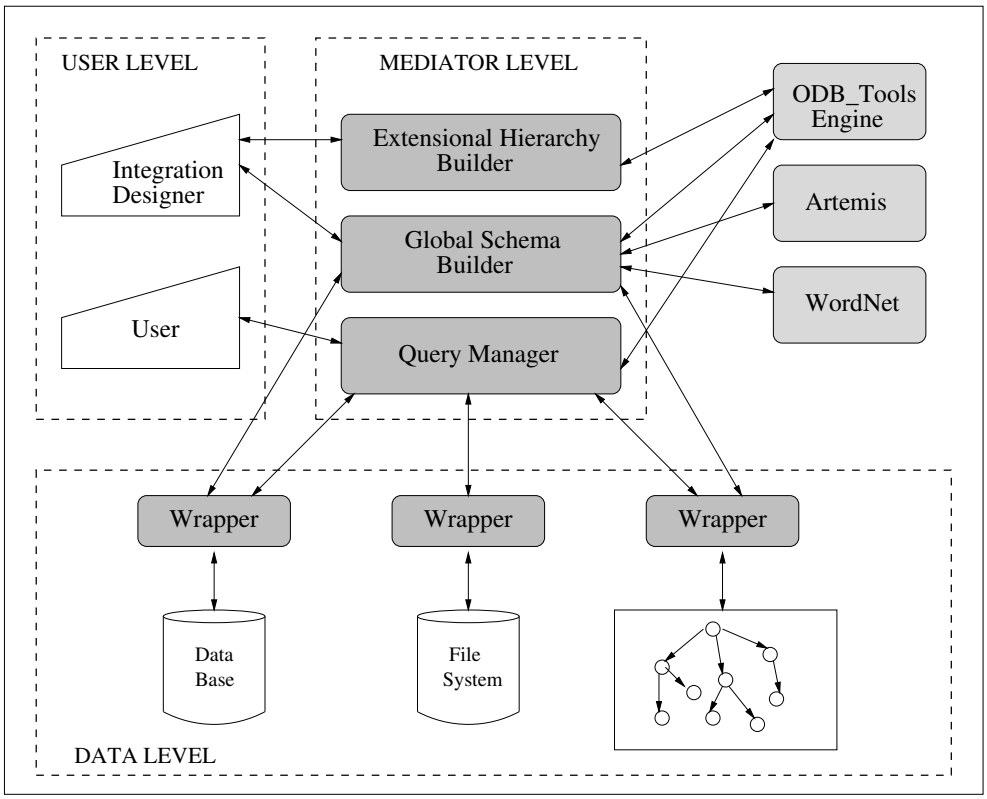

Figura 1.3: Architettura generale di MOMIS

Collegate ai Wrapper ci sono le sorgenti, per questo spesso si parla di quattro livelli.

- **Livello Mediatore**
  Il mediatore è il cuore del sistema ed è composto da tre moduli, ognuno preposto a funzionalità bel precise.

  – *Global Schema Builder*
  La sua funzione principale è quella di generare lo Schema Globale. Il modulo 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 auto-

maticamente dal sistema (come ad es. l'assegnazione dei nomi alle classi globali, la modifica di relazioni lessicali, ...).

- *Extensional Hierarchy Builder*
  Il modulo si occupa della gestione della conoscenza estensionale, calcolando strutture dette Base Extension e generando la Gerarchia Estensionale. Come questo avvenga sarà affrontato più dettagiatamente nel Capitolo 5.
- *Query Manager*
  È il modulo di gestione delle interrogazioni. In questa fase la singola query posta in OQL_{I^3} dall'utente sullo Schema Globale (che chiameremo *Global Query*) sarà suddivisa in più *Local Query* (anch'esse espresse in OQL_{I^3}) da inviare alle varie sorgenti, o meglio, come abbiamo visto, ai wrapper predisposti alla loro traduzione. Questa traduzione avviene in maniera automatica da parte del Query Manager utilizzando la conoscenza intensionale ed estensionale definita nella precedente fase di integrazione.

- **Livello Utente**
  L'utente del sistema può interrogare lo schema globale e per lui sarà come interrogare un database tradizionale. La query posta dall'utente sullo schema globale viene trasmessa come input al Query Manager, che interroga le sorgenti e fornisce all'utente la risposta cercata. Tutte queste operazioni, per l'utente, risultano completamente trasparenti.

1.3 Tools di supporto

Per realizzare il processo di integrazione degli schemi il sistema mediatore MOMIS sfrutta anche alcuni tool esterni descritti qui di seguito:

- **ODB-Tools**: è uno strumento software sviluppato presso il dipartimento di Ingegneria dell'Università di Modena e Reggio Emilia [25, 26, 27]. Esso si occupa della validazione di schemi e dell'ottimizzazione semantica di interrogazioni rivolte a Basi di Dati orientate agli Oggetti (OODB).
  L'architettura di ODB-Tools, come si vede in Figura 1.4 prevede vari componenti, tra cui:

  - *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 (verifica che non vi siano classi incoerenti e calcola relazioni di specializzazione non esplicitate dallo schema).

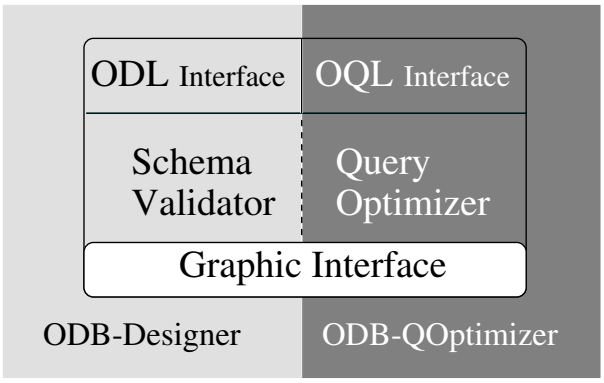

Figura 1.4: Architettura di ODB-Tools

  - *ODB-Qoptimizer* si occupa 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**: 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 [28].

- **ARTEMIS** (Analisys and Reconciliation Tool Enviroment for Multiple Information Sources) [29]: è uno strumento software sviluppato presso l'Università di Milano e Brescia. 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 con le altre classi. Questi coefficienti serviranno per raggruppare le classi locali in modo tale che ogni gruppo (*cluster*) comprenda solo classi con coefficienti di affinità simili.

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

Il linguaggio ODL$_{I^3}$ è il linguaggio di definizione attraverso il quale i wrapper comunicano al mediatore (ed in particolare al Global Schema Builder) le descrizioni delle sorgenti da loro servite. Punto di partenza per la definizione di questo linguaggio, è stato l'attenersi alle raccomandazioni della proposta di standard-

izzazione per linguaggi di mediazione[26](**inserire nella bibl**). Parallelamente a questa proposta (secondo la quale i diversi sistemi di mediazione avrebbero potuto supportare sorgenti con modelli complessi, come quelli ad oggetti, e sorgenti più semplici, come file di strutture), si è cercato di discostarsi il meno possibile dal linguaggio ODL, a sua volta proposto dal gruppo di standardizzazione ODMG-93 [23, 24]**inserire nella bibl**. Le caratteristiche peculiari di ODL, al pari di altri linguaggi basati sul paradigma ad oggetti, possono essere così riassunte:

- definizione di tipi-classe e tipi-valore;
- distinzione fra intensione ed estensione di una classe di oggetti;
- definizione di attributi semplici e complessi;
- definizione di attributi atomici e collezioni (set, list, bag);
- definizione di relazioni binarie con relazioni inverse;
- dichiarazione della signature dei metodi.

Il linguaggio ODL rappresenta il punto di partenza nel progetto di integrazione; pur essendo, infatti, ben progettato per rappresentare la conoscenza relativa ad un singolo schema ad oggetti, è certamente incompleto se calato in un contesto di integrazione di basi di dati eterogenee quale è quello descritto in questa tesi. Si è reso pertanto necessario definirne un'estensione, denominata OQL$_{I^3}$, in accordo con le raccomandazioni della proposta di standardizzazione per i linguaggi di mediazione, risultato del lavoro di workshop I^3 svoltosi presso l'università del Maryland [26]. Tuttavia, partendo da questa proposta, secondo cui i diversi sistemi di mediazione dovrebbero poter supportare sorgenti con modelli complessi (come quelli ad oggetti) e modelli più semplici (come file di strutture), si è comunque cercato di discostarsi il meno possibile dal linguaggio ODL. I propositi raggiunti dall'estensione di ODL che hanno portato al linguaggio ODL$_{I^3}$ sono i seguenti:

- per ogni classe è data al wrapper la possibilità di indicare nome e tipo del sorgente di appartenenza;
- per le classi appartenenti ai sorgenti relazionali è possibile definire le chiavi candidate ed eventuali foreign key;
- attraverso l'uso del costrutto union ogni classe può avere più strutture dati alternative, mentre il costrutto optional consente di indicare la natura opzionale di un attributo. Queste caratteristiche sono in accordo con la strategia utilizzata per la descrizione di dati semistrutturati;

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

- il linguaggio supporta la dichiarazione di regole di mapping (o mapping rule fra grandezze globali e grandezze locali;

- è data la possibilità di definire regole di integrità (o if then rule), sia sugli schemi locali, sia sullo schema globale;

- il linguaggio supporta la definizione di relazioni terminologiche di sinonimia (syn), ipernimia (bt), iponimia (nt) e associazione (rt);

- il linguaggio può essere automaticamente tradotto nella logica descrittiva OLCD usata da ODB-Tools, e quindi utilizzarne le capacità nei controlli di consistenza e nell'ottimizzazione semantica delle interrogazioni.

È da notare come nella fase di progettazione sia compito dei wrapper eliminare le eterogeneità legate al tipo di sorgenti e alla sintassi con cui sono definiti, traducendo la descrizione degli schemi, siano essi relazionali, ad oggetti, semistrutturati, etc.... in un unico linguaggio descrittivo comune. Utilizzando questo linguaggio, sono fornite dal wrapper al mediatore le descrizioni di tutte le classi da integrare: le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema e quindi rendere interrogabili. Non è detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente: ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa. Poichè nel sistema MOMIS si è scelto di descrivere i sorgenti basandosi sul paradigma a oggetti, indipendentemente dal modello originale adottato da ogni singolo database sorgente, ogni entità verrà descritta dal relativo wrapper utilizzando sempre il concetto di classe. Le sintassi dei linguaggi ODL ed ODL$_{I^3}$ sono riportate in BNF rispettivamente in Appendice B.

1.5 Il linguaggio OQL$_{I^3}$

Anche nella definizione del linguaggio di interrogazione OQL$_{I^3}$ si è deciso di adottare la sintassi OQL senza discostarsi dallo standard. Ciò perchè nel sistema MOMIS gli elementi di base del piano di esecuzione sono le Basic Query e queste, essendo rivolte ad una singola classe globale, non contengono operatori complessi (in particolare join tra classi). Le Basic Query sono espresse mediante un sottoinsieme del linguaggio OQL, poichè:

- è possibile navigare attraverso aggregazioni e associazioni per ricostruire oggetti complessi ma non sono ammessi join espliciti tra classi;

- non è prevista la presenza di subquery;
- non sono restituite strutture complesse come list, array o struct;
- non sono presenti operatori di ordinamento (order by) o di conversione come:
    1. listtoset: trasforma una lista di elementi in un set, quindi privo di oggetti duplicati. Ad esempio lo statement listtoset(list(1,2,3,2)) restituisce il set composto dagli elementi 1,2,3;
    2. element: data una collezione di oggetti, ritorna l'elemento in esso contenuto a condizione che sia unico;
    3. atten: trasforma un collezione di collezione in una collezione ad un solo livello. Ad esempio lo statement atten(list(list(1,2),list(1,2,3)) restituisce la lista list(1,2,1,2,3);

Il linguaggio OQL (Object Query Language) è un linguaggio d'interrogazione ad oggetti avente una sintassi eccezionalmente chiara e potente simile all'SQL, da cui peraltro deriva: per molti aspetti è un'estensione ad oggetti di SQL (l'unica vera innovazione rispetto a SQL è l'istruzione "define extent", che definisce una variabile contenente tutte le istanze di una classe). Questo linguaggio è estremamente versatile ed espressivo perciò se da un lato è necessario uno sforzo maggiore nello sviluppo di moduli per l'interpretazione e gestione delle interrogazioni (implementando le funzionalità proprie di un ODBMS), dall'altro si ha la possibilità di sfruttare al meglio le informazioni rappresentate nello schema globale. Le inevitabili complicazioni a livello implementativo sono, pertanto, ampiamente giustificate da una maggiore versatilità e facilità d'uso per l'utente (si impiega infatti un linguaggio standard e non un formalismo ad-hoc). Le principali caratteristiche di questo linguaggio sono:

- OQL è basato sul modello ad oggetti definito da ODMG.
- OQL utilizza una sintassi simile a quella definita per SQL 92. Rispetto a questa presenta delle estensioni finalizzate alla gestione degli aspetti object-oriented, in particolare gli oggetti complessi, l'identità degli oggetti, le espressioni di percorso, il polimorfismo, l'invocazione delle operazioni e il late binding.
- OQL fornisce delle primitive ad alto livello per manipolare insiemi di oggetti, ma non è strettamente legato a questo costrutto. Fornisce infatti anche le primitive per gestire, con la stessa efficienza, altri tipi strutturati come array, liste e strutture.

- OQL non fornisce in modo esplicito operatori per l'aggiornamento della base di dati, lasciando questo compito a operazioni opportune che fanno parte delle caratteristiche di ogni oggetto che popola il database. In questo modo si rispetta la semantica propria degli ODBMS che, per definizione, vengono gestiti attraverso i metodi definiti sugli oggetti.

- OQL permette di accedere agli oggetti in maniera dichiarativa. Questa caratteristica rende più immediata la formulazione dell'interrogazione da parte dell'utente e più semplice ottimizzare le interrogazioni.

Capitolo 2

Integrazione degli schemi e Query Processing in MOMIS

MOMIS è stato sviluppato col fine di realizzare un sistema mediatore versatile ed efficiente che permetta, ad un generico utente, di reperire informazioni su un insieme eterogeneo di sorgenti.
Questo obiettivo viene raggiunto agendo su due fronti: da un lato sono stati progettati strumenti che, sfruttando la conoscenza intensionale e la conoscenza estensionale, sono in grado di assistere il progettista nella complessa fase di *integrazione degli schemi* finalizzato alla generazione della *vista globale*; dall'altro è stato realizzato un Query Manager che, posta una interrogazione sulla vista globale ottenuta, automatizza il processo di reperimento ed integrazione delle informazioni. Questa fase prende il nome di *Query Processing*.

2.1 Integrazione degli schemi

Gli schemi locali vengono integrati in MOMIS secondo criteri sia intensionali che estensionali.
Gli aspetti intensionali a cui si fa riferimento riguardano il fatto che schemi locali parzialmente sovrapposti possono rappresentare uno stesso concetto adottando strutture diverse. È necessario quindi individuare ed eliminare questi conflitti, provvedendo a definire una rappresentazione unificata ed omogenea dei medesimi concetti descritti nelle varie sorgenti.
L'integrazione intesionale non è però l'unico aspetto che occorre gestire per ottenere un'effettiva integrazione di sorgenti eterogenee. Infatti, come descritto in [21, 30, 31], è necessario risolvere anche conflitti di natura estensionale, ovvero

derivanti dalle sovrapposizioni delle estensioni, cioè dalla presenza, in sorgenti diverse, di informazioni relative alla stessa entità del "mondo reale".

2.1.1 Esempio di riferimento

Il seguente esempio, che verrà utilizzato per illustrare le fasi di integrazione degli schemi e di Query Processing, fa riferimento alle definizioni degli schemi delle sorgenti espresse in ODL_{I^3} e riportate in Appendice C. 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.

Sorgente UNIVERSITY (UNI)

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

Sorgente COMPUTER_SCIENCE (CS)

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

Sorgente TAX_POSITION_XML (TP)

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

Figura 2.1: Esempio di riferimento

La prima sorgente, `University` (UNI), è un database di tipo relazionale, che contiene informazioni sullo staff e sugli studenti di una determinata università. Essa è composta da cinque tabelle: `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Per ogni professore (presente nella tabella `Research_Staff`), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key `dept_code`), sul suo indirizzo di posta elettronica (`e_mail`), e sul corso da lui tenuto (`section_code`). Per il corso inoltre viene memorizzata l'aula (`Room`) dove questo si svolge, mentre del dipartimento è descritto, oltre al nome (`dept_name`) ed al codice (`dept_code`),

il budget (budget) che ha a disposizione. Per gli studenti presenti nella tabella School_Member sono invece mantenuti il nome (name), la facoltà di appartenenza (faculty) e l'anno di corso (year).

La sorgente Computer_Science (CS) è un database ad oggetti e contiene invece informazioni sulle persone afferenti a questa facoltà. Sono presenti sei classi: CS_Person, Professor, Student, Office, Location e Course. I dati utilizzati sono comunque abbastanza simili a quelli della sorgente UNI: per quanto riguarda i professori, sono memorizzati il livello (rank), e l'ufficio di appartenenza (belong_to), che a sua volta fa parte di un dipartimento (e può quindi essere considerata una specializzazione); per gli studenti sono memorizzati i corsi seguiti (takes), l'anno di corso (year) e il livello (rank). Il corso ha poi un attributo complesso che lo lega al professore che ne è titolare (taught_by), mentre per l'ufficio si tiene l'indirizzo (address) e la descrizione (description).

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

2.1.2 Integrazione intensionale

Con integrazione intensionale si intende il processo di **Unificazione degli schemi** [32]. L'uso della logica descrittiva OLCD insieme all'uso di tecniche di clustering [33], permettono la realizzazione di una fase semi-automatica di integrazione degli schemi fino a pervenire alla definizione dello *Schema Globale*, direttamente interrogabile dall'utente e che rappresenti l'unione di tutti gli schemi locali, rimuovendone incongruenze e ridondanze.
Il Global Schema Buider è il modulo di MOMIS che si occupa dell'integrazione degli schemi per la generazione dell'unico schema globale da presentare all'utente. Questo modulo, interagendo col progettista, realizza una fase semi-automatica di integrazione, che, partendo dalla descrizione ODL_{I^3} delle sorgenti, porta alla creazione dello schema globale, passando attraverso quattro sottofasi.

Generazione del Thesaurus di relazioni terminologiche

Lo scopo di questa fase è la costruzione del *Common Thesaurus*, ovvero di un tesoro di relazioni terminologiche che rappresenti la conoscenza a disposizione sulle classi da integrare (sui nomi delle classi e sugli attributi). Queste relazioni terminologiche vengono derivate in modo semi-automatico dalla descrizione ODL_{I^3} delle sorgenti attraverso l'analisi strutturale e di contesto

delle classi coinvolte. Indicando genericamente con *termine* t_i il nome di una classe o di un attributo (per identificare in modo univoco una classe, è necessaria la coppia *nome_sorgente.nome_classe*, un attributo, è necessaria la coppia *nome_sorgente.nome_attributo*), si possono definire, all'interno del Thesaurus, le seguenti relazioni:

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

- BT (broader-term): questa relazione è definita tra due termini t_i e t_j, tali che t_i ha un significato più ampio, più generale di t_j. Un caso di BT, nel nostro esempio è $\langle$`UNI.Research_Staff` BT `CS.Professor`$\rangle$.

- NT (narrower-term): concettualmente è la relazione inversa di BT, dunque t_i BT $t_j \rightarrow t_j$ NT t_i. Lo stesso esempio potrebbe infatti essere: $\langle$`CS.Professor` NT `UNI.Research_Staff`$\rangle$.

- RT (related-term): questa relazione è definita tra due termini t_i e t_j che sono generalmente usati nello stesso contesto, tra i quali esiste comunque un legame generico. Un esempio può essere: $\langle$`CS.Student` RT `CS.Course`$\rangle$. La relazione è simmetrica.

L'intero processo che porta, partendo dalle descrizioni degli schemi in ODL_{I^3}, alla definizione di un Thesaurus comune, si articola in cinque passi:

1. *Estrazione automatica di relazioni intra-schema*
   Sfruttando le informazioni semantiche presenti negli schemi strutturati, può essere identificato in modo automatico un insieme di relazioni terminologiche (ad esempio relazioni NT e BT derivate dalle gerarchie di generalizzazione).

2. *Estrazione automatica di relazioni inter-schema*
   Le relazioni inter-schema, terminologiche ed intensionali, sono estratte analizzando gli schemi ODL_{I^3} nel loro insieme. La loro estrazione è basata sulle relazioni lessicali che sussistono tra nomi di classi ed attributi, derivanti dai significati delle parole usate.
   Le relazioni trovate sono sia inter-schema che intra-schema. Però, se gli schemi sono stati strutturati bene, non dovrebbero venire trovate relazioni intra-schema che non siano già state trovate al passo precedente.

3. *Revisione/Integrazione delle relazioni*
   Il progettista interagisce con il modulo, inserendo esplicitamente tutte le relazioni terminologiche che non sono state estratte precedentemente, ma che devono comunque essere presenti per pervenire ad una corretta integrazione degli schemi. Inoltre, il progettista ha il compito di inserire gli assiomi estensionali, che descrivono le relazione esistenti tra le estensioni delle classi. Queste relazioni estensionali vincolano le classi che coinvolgono ad avere anche un legame intensionale, che viene registrato nel Common Thesaurus.

4. *Validazione delle relazioni*
   In questa fase, ODB-Tools viene utilizzato per validare le relazioni terminologiche del Thesaurus definite tra due attributi. La validazione è basata sul controllo di compatibilità dei domini associati agli attributi.

5. *Inferenza di nuove relazioni*
   In questa fase vengono inferite nuove relazioni terminologiche utilizzando ODB-Tools, partendo da quelle già introdotte nel Common Thesaurus ed utilizzando le tecniche di inferenza di OLCD. Siccome le relazioni semantiche di equivalenza (SYN) e di generalizzazione (BT) stabilite nel Common Thesaurus tra i nomi di classi, possono entrare in conflitto con le descrizioni strutturali delle classi correlate, si produce uno *schema virtuale* espresso in ODL_{I^3}, che contiene una descrizione degli schemi sorgenti "ristrutturata" sulla base delle relazioni semantiche nel tesoro. Lo schema virtuale costituisce solo uno strumento tecnico per l'inferenza di nuove relazioni e rimane completamente trasparente all'utente.

Analisi delle affinità intensionali tra le classi

In questa fase vengono individuate classi, appartenenti a sorgenti diverse, che descrivono informazioni semanticamente equivalenti. Per questo scopo le classi vengono analizzate e valutate in base al concetto di *affinità*, in modo da individuare il livello di *similarità*. In particolare, vengono analizzate le relazioni che esistono tra i nomi delle classi (attraverso il *Name Affinity Coefficient*) e tra i loro attributi (per mezzo dello *Structural affinity Coefficient*), per arrivare ad un valore globale denominato *Global Affinity Coefficient*.
Il valore numerico assunto da questi coefficienti viene calcolato sulla base del peso (σ_{rel}) che viene associato ad ogni relazione terminologica. Ovviamente questo peso sarà tanto maggiore, quanto più questo tipo di relazione contribuisce a legare due termini. Nelle applicazioni presenti nel sistema MOMIS si è assunto: $\sigma_{syn} = 1$, $\sigma_{bt/nt} = 0.8$, $\sigma_{rt} = 0.5$.

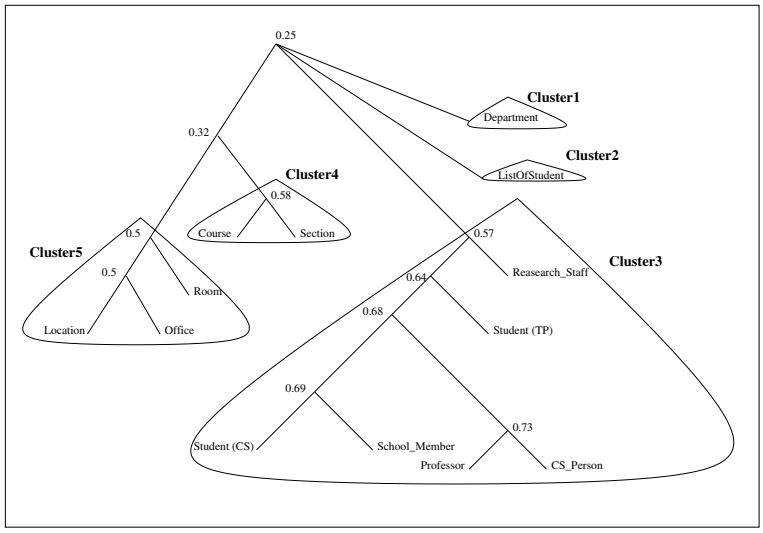

Figura 2.2: Albero di affinità

Questa attività è compiuta con il supporto dell'ambiente di ARTEMIS [34] che, attraverso un processo interattivo, permette di modificare i parametri di valutazione delle affinità e di validare le scelte fatte.

Creazione dei cluster

In questa fase si creano raggruppamenti di classi affini, ovvero insiemi di classi intensionalmente affini, per le quali si suppone esista anche una qualche sovrapposizione estensionale.
La procedura di clustering adottata opera in modo iterativo, andando a creare insiemi di classi, *cluster* appunto, di dimensioni via via crescenti. Questa operazione viene fatta sulla base dei coefficienti di affinità che legano le classi. Infatti, ad ogni passo viene costruito un nuovo cluster unendo quelli ottenuti al passo precedente ed aventi il valore massimo di affinità. Questo processo porta alla creazione di un *albero di affinità* (Figura 2.2), caratterizzato dall'avere come foglie le classi, come radice l'insieme di tutti i cluster, ed i cui nodi sono proprio i cluster individuati (con valore di affinità calante spostandosi verso la radice).
Dopo aver costruito l'albero di affinità, è necessario selezionare i cluster più appropriati per la definizione delle classi globali. La procedura di selezione dei cluster, in ARTEMIS (modulo preposto a questa fase), viene mantenuta interattiva attraverso la modifica del valore di soglia. Infatti il progettista specifica il valore di una soglia T, ed i cluster che prevedono un Global Affinity Cofficient superiore

o uguale a T sono selezionati e proposti. Ovviamente, più il valore di T è alto, più si ottengono cluster piccoli e molto omogenei tra di loro. In ARTEMIS il valore di dafault per T è 0.5.

Generazione dello Schema Globale del Mediatore

In questa fase, da ogni cluster si costruisce una *classe globale* la cui estensione è costituita dall'unione delle estensioni delle classi sorgenti che costituiscono il cluster, mentre l'intensione è ricavata dall'unione "ragionata" degli attributi delle stesse. Con "ragionata" si fa riferimento al fatto che, nel determinare l'intensione di una classe globale, vengono seguiti i seguenti passi:

- ad ogni $classe\_globale_i$ è associata l'unione degli attributi di tutte le classi appartenenti al $cluster_i$ dal quale è stata generata.
- all'interno dell'unione degli attributi sono identificati tutti gli insiemi di termini definiti sinonimi, e ne viene riportato solo uno tra essi (rimuovendo quindi tutti gli altri).
- all'interno dell'unione degli attributi sono identificati tutti gli insiemi di termini legati da relazioni di specializzazione e vengono riorganizzati all'interno di gerarchie. Per ognuna di queste gerarchie è mantenuto solo il termine più generale, mentre sono rimossi tutti gli altri.

Oltre a questa unione "ragionata" degli attributi è necessaria un'ulteriore fase di raffinamento, che aumenti l'espressività dello schema globale, portando alla generazione, per ogni classe globale, di una *Mapping Table*, cioè di una struttura dati contenente tutte le informazioni necessarie per il passaggio dalla rappresentazione globale agli schemi locali.
Questa fase prevede l'intervento del progettista, e nella scelta del nome da associare alla classe globale (il tool si limita a proporre un nome), e nella definizione dei *mapping* che caratterizzano la Mapping Table. Queste informazioni servono da un lato all'utente finale per poter utilizzare in modo efficace la vista globale, dall'altro al Query Manager per svolgere in maniera automatica la fase di Query Processing (come verrà spiegato in Sezione 2.4).
I tipi di mapping, tra gli attributi globali e quelli locali, gestiti finora dal sistema sono:

- *mapping semplice*
  L'attributo globale corrisponde ad un singolo attributo locale. L'analisi di relazioni di sinonimia porta alla fusione di più attributi, appartenenti a classi locali diverse, in un unico attributo. Analogamente nel caso di relazioni di

specializzazione che legano tra di loro attributi locali appartenenti a classi locali diverse.

- *and mapping*
  L'attributo globale corrisponde alla concatenazione, in uno specificato ordine, di più attributi locali appartenenti alla medesima classe locale.

- *complex mapping*
  L'attributo globale mappa su altre classi.
  Si ha questo mapping quando l'attributo globale ha come dominio un'altra classe locale, o quando gli attributi locali costituiscono una foreign key.

- *union mapping*
  Descrive la regola di *or mapping* che prevede la sostituzione dell'attributo globale con un solo attributo locale, scelto tra i possibili candidati della classe locale. Questa scelta viene fatta sulla base del valore assunto da un attributo di riferimento (*tag attribute*), appartenente all'insieme di attributi locali della classe locale considerata.

- *default mapping*
  L'attributo locale assume un valore predefinito. Si verifica la necessità di questa operazione (l'esplicitazione del valore viene effetuata dal progettista) quando alcune informazioni sono presenti negli schemi delle sorgenti sottoforma di metadato e il sistema non le considera.

- *null mapping*
  L'attributo globale non ha corrispondenza tra gli attributi locali.

Il processo di unificazione degli schemi applicato all'esempio di riferimento porta all'individuazione di cinque classi globali:

Global Schema **University_Schema**

- University_Person (name,dept,e_mail,section,faculty,year,belong_to,takes,rank,studcode,tax)
- Department (dept_name,dept_code,budjet)
- List (Student)
- Section (section_name,section_code,taught_by,length,room_code)
- Location (description,address,city,number,street,county,room_code,seats_number,notes)

University_Person	name	dept	e_mail	section	faculty	
UNI.Research_Staff	name	dept_code	e_mail	section_code	null	...
UNI.School_Member	name	null	null	null	faculty	...
CS.CS_Person	first_name and last_name	null	null	null	null	...
CS.Student	first_name and last_name	null	null	null	faculty_name	...
CS.Professor	first_name and last_name	null	null	null	null	...
TP.Student	name	null	null	null	null	...

	year	belong_to	takes	rank	studcode	tax
...	null	null	null	"professor"	null	null
...	year	null	null	"student"	null	null
...	null	null	null	null	null	null
...	year	null	takes	rank	null	null
...	null	belong_to	null	rank	null	null
...	null	null	null	"student"	student_code	tax_fee

Figura 2.3: Mapping Table di University_Person

Inoltre, per ognuna di queste classi globali viene definita la corrispondente Mapping Table. La Mapping Table corripondente alla classe globale University_Person e rappresentata in Figura 2.3.
Questa fase si conclude con la realizzazione della descrizione in ODL_{I^3} delle classi globali generate. In particolare, per ogni attributo globale, oltre alla specifica del nome e del tipo, viene aggiunta una serie di *mapping rule* in ODL_{I^3}, attraverso le quali vengono specificate le informazioni su come questo attributo verrà accoppiato con attributi locali, come pure informazioni su valori di default o nulli.
Per la classe globale University_Person, ad esempio, si ha:

```
interface University_Person
(extent Research_Staffers, School_Members, Students, CS_Person,
 Professors, Students
 key name)
{attribute string name
     mapping rule University.Research_Staff.name,
                  University.Scool_Member.name,
                  (Computer_Science.CS_Person.first_name and
                   Computer_Science.CS_Person.last_name),
                  (Computer_Science.Professor.first_name and
                   Computer_Science.Professor.last_name),
                  (Computer_Science.Student.first_name and
                   Computer_Science.Student.last_name),
                  Tax_Position_xml.Student.name)
 attribute string rank
     mapping rule  University.Research_Staff = "Professor",

     ...}
```

2.2 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.

2.2.1 Schema Globale

Sia $\mathcal{L}$ un insieme di *nomi di classi locali* (denotati da $L_1, L_2, \ldots$) e $\mathcal{AL}$ 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 $A_L : \mathcal{L} \rightarrow 2^{\mathcal{AL}}$.

Sia $\mathcal{G}$ un insieme di *nomi di classi globali* (denotati da $G_1, G_2, \ldots$) e $\mathcal{AG}$ 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 $A_G : \mathcal{G} \rightarrow 2^{\mathcal{AG}}$.

Definizione 2.2.1 (Schema Globale) *Data un insieme $\mathcal{G}$ ed un insieme $\mathcal{L}$, uno* schema globale ***S(G)** di $\mathcal{L}$, è una funzione **S(G)***

$$\textbf{\textit{S(G)}} : \mathcal{G} \rightarrow 2^{\mathcal{L}}$$

*tale che **S(G)**(G_1), **S(G)**$(G_2), \ldots,$ **S(G)**(G_k) sia un partizionamento di $\mathcal{L}$.*

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

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

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* $\exists^1\, a_i \in A(c_k) : a_i \leftrightarrow ga_h) \vee (\exists\, a_i \in A(c_k) :$ $a_i = ga_h =$ `<costante>`)
- 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 $i = 1, \ldots, M \Rightarrow al_i \leftrightarrow ag_h$

2.3 Dalle asserzioni estensionali alle Base Extension

L'integrazione intensionale non è però l'unico aspetto che occorre gestire per ottenere un'effettiva integrazione di sorgenti eterogenee, infatti è necessario risolvere anche conflitti derivanti dalle sovrapposizioni delle estensioni, cioè dalla presenza, in sorgenti diverse, di informazioni relative alla stessa entità del mondo reale.
Con integrazione estensionale si fa riferimento al processo di **Fusione delle istanze** [21], ovvero al processo di *"fusione"* degli oggetti recuperati dalle varie sorgenti, col fine di ricostruire le estensioni delle classi di entità del *"dominio applicativo"*. Il modulo che si occupa di realizzare ciò è l'Extensional Hierarchy Builder. L'approccio seguito in MOMIS si basa sulla teoria della *formal context analysis* [35], che è volta alla generazione di una gerarchia di ereditarietà in cui viene rappresentata la conoscenza disponibile, nell'insieme di schemi locali, su di un determinato aspetto della realtà. Gli elementi che caratterizzano questo approccio teorico sono i seguenti:

1. Definizione degli assiomi estensionali
2. Individuazione delle Base Extension
3. Generazione della Gerarchia Estensionale

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

2.3.1 Assiomi estensionali e Base Extension

Definizione degli assiomi estensionali

Gli *assiomi estensionali* descrivono le relazioni insiemistiche esistenti tra le estensioni delle sorgenti.

Definizione 2.3.1 (Stato di una classe) *Lo stato di una classe* **C**[2] *all'istante t, scritto* $Stato_C^t$*, è 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".*

Date due classi A e B, sono individuabili quattro tipologie di relazioni estensionali tra di esse:

- *sovrapposizione*: $\forall t : S_A^t \cap S_B^t \neq \emptyset$
- *inclusione*: $\forall t : S_A^t \subseteq S_B^t$
- *equivalenza*: $\forall t : S_A^t = S_B^t$
- *disgiunzione*: $\forall t : S_A^t \cap S_B^t = \emptyset$

Una parte degli assiomi estensionali viene ricavata direttamente dalla descrizione degli schemi. Infatti una relazione ISA viene espressa tramite l'assioma di inclusione. Però la maggior parte degli assiomi estensionali, presenti sulle classi, viene esplicitata dal progettista.
L'analisi estensionale fatta in MOMIS si basa su due presupposti:

1. per classi appartenenti ad uno stesso cluster, e per le quali non è specificata nessuna relazione (e non è possibile ricavare nessuna relazione implicita), si assume che le loro estensioni siano sovrapposte;
2. tra classi appartenenti a cluster diversi si suppone sussista una relazione di disgiunzione estensionale.

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:

[2] C può essere definito come un'espressione logica che coinvolge più classi.

- $A \ \ SYN_{Ext} \ \ B$: le istanze della classe A sono le stesse della classe B. Questo implica una relazione intensionale di tipo SYN tra le due classi e due relazioni ISA;

- $A \ NT_{Ext} \ B$: le istanze della classe A sono un sottoinsieme di quelle della classe B. Questo assioma viene scomposto in una relazione intensionale di tipo NT e una relazione ISA;

- $A \ BT_{Ext} \ B$: le istanze della classe A sono un sovrainsieme di quelle della classe B. Viene generata una relazione intensionale di tipo BT ed una ISA tra le classi;

- $A \ DISJ_{Ext}$[3] B: le istanze della classe A sono diverse da quelle della classe B. Questo assioma non implica nessun tipo di relazione intensionale, esiste solamente un legame di tipo BOTTOM[4] 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.

In MOMIS le relazioni estensionali vengono espresse come rule nel linguaggio ODL_{I^3}:

- relazione di *inclusione*:
  ```
  rule RE1 forall x in B then x in A
  ```

- relazione di *disgiunzione*:
  ```
  rule RE2 forall x in (A and B) then x in bottom
  ```

- relazione di *equivalenza*:
  ```
  rule RE3 forall x in A then x in B
  rule RE4 forall x in B then x in A
  ```

Si può notare come le relazioni di sovrapposizione non debbano essere espresse in modo esplicito, questo deriva dal precedente presupposto 1.
Per come sono stati definiti gli assiomi estensionali di tipo SYN_{Ext}, NT_{Ext} e BT_{Ext}, si intuisce che il legame che generano tra le classi coinvolte è molto forte poichè implica sia un legame tra gli schemi che un legame tra le istanze. Per questo motivo, le classi legate da relazioni di questo tipo validate, devono appartenere necessariamente allo stesso cluster. Infatti, le relazioni intensionali implicate dagli assiomi estensionali, vengono registrate nel Common Thesaurus con

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

[4] 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.

peso pari ad uno (in modo da forzare le classi locali coinvolte ad appartenere allo stesso cluster).
Le relazioni intensionali implicate dagli assiomi estensionali vengono validate automaticamente quando non determinano inconsistenze nelle descrizioni degli schemi (il progettista ha comunque la possibilità di validarle manualmente). Quindi, risulta evidente che l'ipotesi che afferma che, classi locali appartenenti a cluster diversi sono disgiunte, può, in casi sfortunati, non essere vera. Infatti, la non validazione della relazione estensionale a livello di Common Thesaurus, potrebbe portare classi locali parzialmente sovrapposte a finire in cluster diversi, per il fatto che hanno coefficienti di affinità bassi.
Quindi, la conoscenza estensionale è fondamentale nel processo di determinazione dei cluster (*Arricchimento dei cluster*).

La gestione della conoscenza estensionale ha bisogno dell'intervento umano. Il progettista di un database deve dichiarare il contenuto informativo di ogni ogetto istanziato nelle sorgenti di informazione attraverso la definizione di alcune asserzioni estensionali.
Siccome lo Schema Globale rappresenta la conoscenza contenuta nella Global Class generata a partire dale classi locali inerenti all'interrogazione in esame, essa rispecchia l'informazione che tali classi locali contengono riguardo degli oggetti nella realtà, allora noi dobbiamo essere in grado di distinguere tali oggetti se essi risultassero mappati in due classi distinte.
Sia $\mathcal{O}$ un insieme di oggetti istanziati nelle sorgenti interessate. Questo insieme di oggetti è disponibile nel livello del Mediatore. Definiamo una funzione $\mathcal{E}$, chiamata extension nel seguente modo: ($\mathcal{E} : \mathcal{L} \rightarrow 2^{\mathcal{O}}$). Questa funzione associa ogni classe locale L nell'insieme $\mathcal{L}$ al sotto insieme $\mathcal{O}$ rappresentato da L. Abbiamo adottato la notazione con il punto per denotare il valore di un attributo, cioè: se $o \in \mathcal{E}$(C) e $A \in S(C)$ allora o[C].A rappresenta il valore dell'attributo A nell'oggetto o appartenente alla funzione $\mathcal{E}$(C). Poichè sono possibili differenti modi per istanziare lo stesso oggetto, non ci soffermeremo sul problema che riguarda l'integrazione dell'informazione che proviene da sorgenti diverse. Per soddisfare la proprietà di omogeneità semantica abbiamo supposto che lo stesso oggetto istanziato in diverse classi sia rappresentato nello stesso modo. Formalmente:

$$o[C_1].A = o[C_2].A \qquad \begin{array}{l} \forall C_1, C_2, \\ \forall o \in \mathcal{E}(C_1) \cap \mathcal{E}(C_2), \\ \forall A \in S^{\mathcal{G}}(C_1) \cap S^{\mathcal{G}}(C_2) \end{array}$$

Tenendo conto dell'omogeneità semantica abbiamo definito i valori corrispondenti ad ogni oggetto $o \in \mathcal{O}$ a livello del Mediatore nel seguente modo: $\forall A \in S(\mathcal{G})$

$$o.A = \begin{cases} o[C_1].A' & \text{se} \exists L \in \mathcal{L} \mid \mathcal{M}[L, A] = A' \\ \text{null} & \text{in caso contrario} \end{cases} \tag{2.1}$$

Definizione 2.3.2 (Asserzione Estensionale) *Dato un insieme $\mathcal{L}$ di classi locali, un insieme di asserzioni estensionali $\mathcal{A}$ su $\mathcal{L}$ è il set finito di questa forma:*

E' EQ E" *(equivalenza)*
E' IN E" *(inclusione)*
E' DIS E" *(disgiunzione)*

dove E', E" sono espressioni su $\mathcal{L}$ composte per induzione seguendo la seguente sintassi astratta:

E ::= L *(classe locale)*
| E_1 *NOT* E_2 *(differenza)*
| E_1 *AND* E_2 *(intersezione)*
| E_1 *OR* E_2 *(unione)*

Ogni asserzione estensionale di fatto specifica una condizione che tale estensione $\mathcal{E}$ deve soddisfare. Per chiarire questo aspetto dimostreremo che la risposta ad una query globale Q definita sullo Schema Globale **S($\mathcal{G}$)** e presentata in forma normale congiuntiva (CNF) è la stessa della risposta della query definita su una relazione r che deriva da un sottoinsieme L' di L delle classi locali coinvolte nell'interrogazione.
Detto questo definiamo con $r_{|_{L'}}$ la relazione globale che deriva dal sottoinsieme di classi locali $L' \subset L$. Tale relazione è calcolata dall'equazione:

$$\forall A \in \mathbf{S(\mathcal{G})} : t_G(o)[A] = \begin{cases} t_L^G(o)[A] & \text{se} \exists L \in \mathcal{L} : A \in S^G(L), o \in \mathcal{E}(L) \\ \text{null} & \text{in caso contrario} \end{cases} \tag{2.2}$$

dove ogni classe locale L rappresenta una relazione r con schema S(L) e ogni oggetto istanziato in L tramite l'estensione E è rappresentato da una tupla t_L(o) con schema S(L). Supponiamo inoltre che le classi locali soddisfano la proprietà di omogeneità semantica, cosicchè tuple che riferiscono gli stessi oggetti istanziati in classi diversi abbiano lo stesso contenuto.
Diciamo che un sottoinsieme di classi locali $\mathcal{L}' \in \mathcal{L}$ supporta una query congiuntiva se lo schema della query è contenuto nello schema globale corrispondente al sottoinsieme $\mathcal{L}'$ dopo la sua integrazione intensionale ed estensionale.

Proposizione 1 *Dato un insieme di asserzioni estensionali $\mathcal{A}$ sull'insieme di classi locali $\mathcal{L}$ ed una query congiuntiva $\hat{Q}$, allora per ogni estensione $\mathcal{E}$ legale rispetto ad $\mathcal{A}$ abbiamo che:*

$\hat{Q}^r = \hat{Q}^{r_{|\mathcal{L}'(\mathcal{Q})}}$
dove r *e* $r_{|\mathcal{L}'(\mathcal{Q})}$ *sono le relazioni globali che provvengono da* $\mathcal{E}$ *rispetto* $\mathcal{L}$ *e* $\mathcal{L}'(\mathcal{Q})$ *rispettivamente.*

Dimostrazione. Il fatto che $\hat{Q}^r \supseteq \hat{Q}^{r_{|\mathcal{L}'(\mathcal{Q})}}$ è ovvio poichè ogni tupla di $r_{|\mathcal{L}'(\mathcal{Q})}$ è contenuta anche in r. Allora supponiamo di avere una tupla $t_G(o) \in \hat{Q}^r$ dobbiamo dimostrare che $t_G(o) \in \hat{Q}^r_{|\mathcal{L}'(\mathcal{Q})}$. A questo proposito se $t_G(o) \in \hat{Q}^r$ allora segue che $t_G(o) \in r$ dunque l'nterrogazione $Q(t_G(o))$ = true. Questo vuol dire che per ogni $\tau_i \in Q(\forall i = 1, \ldots, n)$ al massimo esiste un unico sottoinsieme $\mathcal{L}'$ di $\mathcal{L}$ tale per cui $\tau_i^{\mathcal{L}_i}(t_{\mathcal{L}_i}(o))$ = true, dove $\tau_i^{\mathcal{L}_i}$ rappresenta il termine τ_i riscritto secondo il vocabolario di $\mathcal{L}_i$.
Da quanto detto prima segue che $o \in \bigcap_{i=1}^{n} \mathcal{L}_i$ ed esiste un sottoinsieme $\mathcal{L}' \subset \mathcal{L}$ tale che $L' \in \mathcal{L}$.In più $S(\hat{Q}) \subset$ **S($\mathcal{G}$)**, poichè le classi coinvolte nella query sono tali che $\mathcal{L}' \subset \mathcal{L}$. Segue che $t_G(o) \in r_{|\mathcal{L}'(\mathcal{Q})}$ e dunque $t_G(o) \in \hat{Q}^{r_{|\mathcal{L}'(\mathcal{Q})}}$ □

Definizione 2.3.3 (Asserzione Legale) *Dato un insieme di classi locali* $\mathcal{L}$*, un insieme di asserzioni estensionale* $\mathcal{A}$ *su* $\mathcal{L}$ *ed una funzione estensionale* $\mathcal{E}$ *di* $\mathcal{L}$*, si dice che* $\mathcal{E}$ *è legale rispetto ad* $\mathcal{A}$ *se e solo se:*

$\mathcal{E}$(E') = $\mathcal{E}$(E")	$\forall E' EQE'' \in \mathcal{A}$
$\mathcal{E}$(E') = $\mathcal{E}$(E")	$\forall E' INE'' \in \mathcal{A}$
$\mathcal{E}$(E') = $\mathcal{E}$(E")	$\forall E' DISJE'' \in \mathcal{A}$

dove $\mathcal{E}$*(E) è intuitivamente definita come segue:*

$$\mathcal{E}(E_1 NOT E_2) = \mathcal{E}(E_1) \backslash \mathcal{E}(E_2),$$
$$\mathcal{E}(E_1 AND E_2) = \mathcal{E}(E_1) \cap \mathcal{E}(E_2)$$
$$\mathcal{E}(E_1 OR E_2) = \mathcal{E}(E_1) \cup \mathcal{E}(E_2).$$

Le asserzioni estensionali ci permettono di derivare l'informazione disponibile al livello del mediatore tramite gli oggetti istanziati in $\mathcal{O}$. Infatti il mediatore presenta una vista integrata dell'informazione disponibile nelle sorgenti locali per le interrogazioni. Il Query Processing è effettuato sulla base di dati virtuali i cui informazione per quanto riguarda gli oggetti di $\mathcal{O}$ sono completamente ricavati dalle sorgenti locali. Le asserzioni estensionali specificano i legami tra le estensioni delle classi locali in cui gli oggetti sono istaniati tramite i valori assegnati agli attributi delle classi stesse.

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 $\mathcal{B}$ delle classi $\mathcal{L} = L_1, L_2, \ldots, L_n$ viene definito da una formula booleana in DCF (Forma Canonica Disgiuntiva) sulle variabili $A_1, A_2, ..., A_n$. Ogni *minterm* della cosi 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 2.3.4 (Istanza di una classe globale) *Data una classe globale G ed un dominio **D**, un'istanza di G sul dominio **D** è una funzione $\mathcal{I}$*

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

Un'istanza della classe globale G verrà detta *legale* se soddisfa gli assiomi estensionali definiti sulle classi locali $\mathbf{S}(\mathcal{G})(G)$.
Adesso siamo in grado di dare una definizione per le Base Extension.

Definizione 2.3.5 (Base Extension sulle classi locali) *Dato un insieme di classi locali $\mathcal{L}$ e una funzione estensione $\mathcal{E}$, una base extension B è una combinazione di i classi locali di $\mathcal{L}$ ($B \in C(\mathcal{L}, i)$[5]. Presa l'estensione $\mathcal{E}$, l'estensione delle Base Extension $\mathcal{E}(B)$ sono:*

$$\mathcal{E}(B) = \bigcap_{L \in B} \mathcal{E}(L)$$

d'altro canto partendo dalle $\mathcal{E}(B)$ definiamo il seguente insieme:

$$\chi_{\mathcal{E}}(B) = \mathcal{E} - \bigcup_{L \in (\mathcal{L}-B)} \mathcal{E}(C)$$

Le Base Extension denotano una possibile intersezione delle classi locali che soddisfa la seguente condizione.

Definizione 2.3.6 (Insieme di Base Extension) *Dato un insieme di asserzioni $\mathcal{A}$ e un insieme di oggetti$\mathcal{O}$, un insieme di Base Extension $\mathcal{B} \subseteq \bigcup_{i=1}^{|\mathcal{L}|} C(\mathcal{L}, i)$, tale per cui $\bigcup_{B \in \mathcal{B}} B = \mathcal{L}$ è legale rispetto ad $\mathcal{A}$ se e solo se:*

- *per ogni $\mathcal{E}$ legale per $\mathcal{A}$:*

$$\chi_{\mathcal{E}}(B) | B \in \mathcal{B} \text{ è una partizione di } \mathcal{O} \tag{2.3}$$

- *esiste $\mathcal{E}$ legale per $\mathcal{A}$ tale che:*

$$\chi_{\mathcal{E}}(B) \neq \emptyset \ perogniB \ \in \mathcal{B} \tag{2.4}$$

[5]la $C(\mathcal{L}, i)$ denota tutte le possibili combinazioni disordinate di i elementi presi dall'insieme di classi locali $\mathcal{L}$

La proposizione che segue dimostra la unicità di un insieme di Base Extension.

Proposizione 2 *Dato un insieme di asserzioni $\mathcal{A}$, esiste al massimo un solo insieme di Base Extension $\mathcal{B}$ che è legale rispetto ad $\mathcal{A}$.*

Dimostrazione. Per prima cosa notiamo che l'insieme di Base Extension possa essere costruito considerando l'insieme di tutte le possibili combinazioni delle classi locali che appartengono ad $\mathcal{L}$ (al massimo sono $\sum_{i=1}^{|\mathcal{L}|}$) e rimouvendo dall'insieme delle combinazini quelle che sono in contradizione con le assiomi in $\mathcal{A}$. Ovviamente la BE deve soddisfare la proprietà 2.3 e in più l'estensione $\mathcal{E}$ della proprietà 2.4 è l'unica che soddisfa le assiomi in $\mathcal{A}$ e questo è implicitamente rappresentato da $\mathcal{A}$ stessa. □

In seguito con $\mathcal{B}_{\mathcal{A}}$ indicheremo l'insieme di Base Extension di $\mathcal{A}$. Per chiarire meglio il concetto di Base Extension diamo il seguente esempio.

Esempio 2.3.1 *Consideriamo l'integrazione di due sorgenti, la prima fornisca le informazioni neccesssarie alla definizione della classe Student (name, email, year, tax) mentre la seconda fornisca le informazioni neccesssarie alla definizione della classe Professor (name, email, dept). Assumiamo che dopo la fase di integrazione venga generato lo schema globale con la classe globale UniversityPerson (name, email, year, tax, dept) (per semplicità consideriamo che gli attributi sono rappresentati nello stesso modo nelle due sorgenti). Aggiungendo la classe locale Assistant(name, email, yaer), possiamo generare la classe globale $\mathcal{G}$ = { Universyti_Person } a partire dall'insieme di classi locali $\mathcal{L}$ = { Student, Professor, Assistant } ed esprimere tutto tramite il seguente mapping:*

	name	***email***	***year***	***tax***	***dept***
Student	*name*	*email*	*year*	*tax*	*null*
Professor	*name*	*email*	*null*	*null*	*dept*
Assistant	*name*	*email*	*year*	*null*	*null*

Detto ciò, considerando la seguente asserzione estensionale $\mathcal{A}$ = { Student DISJ Proffessor, Assistant IN Professor}, risulta che la $\mathcal{B}_{\mathcal{A}}$ = $\{B_1, B_2, B_3\}$ dove

$$B_1 = \{Student\}$$
$$B_2 = \{Professor, Assistant\}$$
$$B_3 = \{Professor\}$$

Con gli schemi:

$$S(B_1) = \{name, email, year, tax\}$$

$$S(B_2) = \{name, email, year, dept\}$$
$$S(B_3) = \{name, email, dept\}$$

Prendendo in considerazione la riscrittura della seguente query:

Q: select email
from University_Person
where year = 2002
and (dept = 'cs' or $tax < 200$*)*

notiamo che l'insieme degli attributi della query congiuntiva non è presente in nessuna delle Base Extension. Infatti questa query spedita nella sua forma iniziale ritornerà un insieme di risposte vuoto.

Un insieme di base extension di una classe globale G viene rappresentato tramite una tabella le cui righe riportano le classi locali della classe globale, cioè **S($\mathcal{G}$)**(G), le colonne riportano le base extension, cioè gli elementi di **B**: una X in corrispondenza dell'elemento della tabella (L, B) indicherà che $L \in F(B)$.

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

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

2.3.2 Dominazione tra le Base Extension

Supponiamo che un generico utente abbia posto una interrogazione che richiede informazioni condivise negli schemi di due Base Extension: B_1 B_2. Supponiamo inoltre che l'estensione di B_1 contiene sempre l'estensione di B_2, in altre parole ogni oggetto di B_2 è anche un oggetto di B_1, allora possiamo prendere la risposta all'interrogazione semplicemente applicando la query alle classi locali che fanno parte della base estension B_1. Denotiamo questa proprietà come *dominazione tra Base Extension*. Nel nostro caso risulta che $B_1 \text{dom} B_2$. Più in generale, dato un insieme di Base Extension $\mathcal{B}_\mathcal{A}$ allora per ogni coppia di Base Extension $(B_i, B_j) \in \mathcal{B}_\mathcal{A}$ possiamo dire che $B_i \text{dom} B_j$ se $\mathcal{E}(B_i) \supset \mathcal{E}(B_j)$ per ogni estensione $\mathcal{E}$ legale rispetto ad $\mathcal{A}$. Il problema è che controllare le dominazioni tra le Base Extension, tramite le asserzioni estensionali, ha poca utilità pratica dato il grande dispenso di

tempo e risorse. Tale problema si può ovviare osservando che le dominazioni tra le Base Extension si possono calcolare tramite l'esplorazione diretta delle Base Extension stesse. Il seguente Lemma dimostra che questo sia possibile.

Lemma 1 *Sia $\mathcal{A}$ un insieme di asserzioni estensionali, $\mathcal{B}_\mathcal{A}$ un insieme di Base Extension, $B_1 \in \mathcal{B}_\mathcal{A}$ e $B_2 \in \mathcal{B}_\mathcal{A}$, allora $\mathcal{E}(B_1) \supset \mathcal{E}(B_2)$ per ogni $\mathcal{E}$ legale rispetto ad $\mathcal{A}$ se e solo se $B_1 \subset B_2$.*

Dimostrazione. La condizione neccessaria è ovvia poichè segue dalla definizione di Base Extension. Per la dimostrazione della condizione sufficiente supponiamo che $\mathcal{E}(B_1) \supset \mathcal{E}(B_2)$ per ogni $\mathcal{E}$ legale rispetto ad $\mathcal{A}$, allora considerando i due casi:

- $B_1 \subset B_2$;
- $B_1 \nsubseteq B_2$.

Nel primo caso il Lemma è sicuramente soddisfatto, esplorando le inclusioni insiemistiche si arriva al risultato desiderato. Nel secondo caso se $B_1 \nsubseteq B_2$, allora esiste una classe locale C tale che $C \in (\mathcal{L} - B_2)$. A questo punto possiamo dire che per ogni $\mathcal{E}$ vale la seguente espresione insiemistica:
$\mathcal{E}(B_2) \subset \mathcal{E}(B_1) \subseteq \bigcap_{CinB_1 \wedge Cin(\mathcal{L}-B_2)} \mathcal{E}(C)$. In altre parole segue che or ogni estensione $\mathcal{E}$ vale $\chi_\mathcal{E}(B_2) = \mathcal{E}(B_2) - \bigcup_{L \in (\mathcal{L}-B_2)} \mathcal{E}(L) = \emptyset$. Ma quest'ultima affermazine è in contradizione con la proprietà 2.4 della Definizione 2.3.6.

□

Una volta definita la proprietà di dominazione tra due Base Extension possiamo estenderla per un insieme di Base Extension. Per questo scopo prima introduciamo la notazione di *composizione di Base Extension*. Il simbolo *BC* denota una composizione di Base Extension costruita secondo questa sintassi:
$BC \longrightarrow B|BC_1 \cup \ldots \cup BC_n|BC_1 \cap \ldots \cap BC_n$ dove B è una Base Extension con la seguente sintassi:

$$\mathcal{E}(B_1 \cup B_2) = \mathcal{E}(B_1) \cup \mathcal{E}(B_2)$$
$$\mathcal{E}(B_1 \cap B_2) = \mathcal{E}(B_1) \cap \mathcal{E}(B_2)$$

Possiamo facilmente estendere questa sintassi ad un inieme di Base Extension dando la seguente definizione.

Definizione 2.3.8 *Sia $\mathcal{A}$ un insieme di asserzioni estensinali e $\{B_1 \ldots, B_n\} \subseteq \mathcal{B}_\mathcal{A}$. Allora per ogni $\mathcal{E}$ legale rispetto ad $\mathcal{A}$ si ha:*

$$\mathcal{E}(B_1 \cup \ldots \cup B_n) = \cup_{i=1}^{n} \mathcal{E}(B_i) \tag{2.5}$$
$$\mathcal{E}(B_1 \cap \ldots \cap B_n) = \cap_{i=1}^{n} \mathcal{E}(B_i) \tag{2.6}$$

La seguente proposizione mostra che controllando la dominazione coinvolta tramite le composizione di Base Extension essa può essere ridotta per trovare una dominazione binaria.

Proposizione 3 *Sia $\mathcal{B}_\mathcal{A}$ un insieme di Base Extension di $\mathcal{A}$, allora:*

$$\nexists B_k, B_i, B_j \in \mathcal{B}_\mathcal{A} : B_i \cup B_j dom B_k \tag{2.7}$$
$$\forall B_k, B_i, B_j \in \mathcal{B}_\mathcal{A} : B_k dom B_i \cup B_j \rightleftarrows (B_k dom B_i), (B_k dom B_j) \tag{2.8}$$
$$\forall B_k, B_i, B_j \in \mathcal{B}_\mathcal{A} : B_i \cap B_j dom B_k \rightleftarrows (B_i dom B_k), (B_j dom B_k) \tag{2.9}$$
$$\forall B_k, B_i, B_j \in \mathcal{B}_\mathcal{A} : B_k dom B_i \cap B_j \rightarrow \exists B_h : B_k dom B_h \tag{2.10}$$
$$\exists B_k, B_i, B_j \in \mathcal{B}_\mathcal{A} : B_k dom B_i \cap B_j \leftarrow \forall B_k, B_h : B_k dom B_h \tag{2.11}$$

Dimostrazione. Per l'equazione 2.7 possiamo dire che se per ogni $\mathcal{E}$, $\mathcal{E}(B_k) \subset \mathcal{E}(B_i) \cup \mathcal{E}(B_j)$ allora $\mathcal{E}(B_k)$ può sempre essere espresso nella seguente forma:

$$\mathcal{E}(B_k) = (\mathcal{E}(B_k) \cap \mathcal{E}(B_i) \cap \mathcal{E}(B_j)) \cup (\mathcal{E}(B_k) \cap \mathcal{E}(B_i)) \cup (\mathcal{E}(B_k)\mathcal{E}(B_j)).$$

Notate che seguendo la definizione di Base Extension, una base extension con questa estensione non può esistere poichè è l'unione di intersezioni di estensioni di classi locali. L'equazioni 2.8 e 2.9 sono ovvie. Per dimostrare l'equazioni 2.10 e 2.11 facciamo in questo modo: per la condizione sufficiente supponiamo che $\mathcal{E}(B_i) \cap \mathcal{E}(B_j) \neq \emptyset$ e sia $B_h = B_i \cap B_j$ allora $\mathcal{E}(B_h) = \mathcal{E}(B_i) \cap \mathcal{E}(B_j) \subset \mathcal{E}(B_k)$ per ipotesi, allora risulta che B_k dom B_h. Inoltre se $\mathcal{E}(B_h) \neq \emptyset$, cioè il set di Base Extension secondo la proprietà 2.4 della Definizione 2.3.6 è una partizione se e solo se $B_h \in \mathcal{B}_\mathcal{A}$. Per la condizione necessaria notiamo che per ogni Base Extension $B_h \in \mathcal{B}_\mathcal{A}$, esistono sempre due BE $B_i \in \mathcal{B}_\mathcal{A}$ e $B_j \in \mathcal{B}_\mathcal{A}$ cosi che $B_h = B_i \cap B_j$

□

Da come detto sopra segue che la definizione di dominazine tra le Base Ectension è pienamente specificata dalla sua forma binaria. Poichè questo tipo di relazione è transitiva, antisimmetrica e anti transitiva deriviamo un DAG basato sull'esistenza di dominazioni fra le BE. Più precisamente, durante fase di integrazione costruiamo una matrice M_{dom} di dominazioni comparando ogni coppia di BE. Il valore di $M_{dom}[B_i, B_j]$ è uno se B_j dom B_i e zero altrimenti. La computazione M_{dom} = M_{dom} - $M_{dom} \times M_{dom}$ rimouove delle dominazioni transitive. Solo il valore uno in campo di M_{dom} rappresenta una diretta dominazione fra due BE. L'uscita di M_{dom} sarà cioè aggiunta alla conoscenza disponibile per la definizione del Query Plan.

2.3.3 Individuazione delle base extension

Il prerequisito fondamentale per il calcolo delle *base extension* è che tra le sorgenti coinvolte siano stati risolti tutti i conflitti intensionali, relativi ai nomi ed ai tipi degli attributi.
Una base extension rappresenta un sottinsieme di entità appartenenti ad uno stesso concetto del dominio applicativo. Presa quindi una classe di entità, 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 proprietà. Le base extension sono individuate dalle relazioni estensionali presenti tra le classi locali e sono caratterizzate dall'avere:

- *estensione*: l'insieme di entità formate dall'intersezione delle classi locali che le compongono;
- *intensione*: l'unione degli attributi globali descritti nelle classi locali dell'insieme.

Uno degli scopi principali dell'introduzione del concetto di base extension è dovuto al fatto di poter considerare una singola istanza come componente di una ed una sola base extension, mentre, se consideriamo le singole classi locali, una generica istanza può appartenere a più di una classe.
Chiariamo il concetto di base extension con un esempio. Si supponga che, per la classe globale Univerity_Person dell'esempio di riferimento, siano stati definiti i seguenti assiomi estensionali:

1. `UNI.School_Member` SYN_{Ext} `TP.Student`
2. `CS.Student` NT_{Ext} `UNI.School_Member`
3. `CS.Professor` NT_{Ext} `UNI.Research_Staff`
4. `CS.Professor` $DISJ_{Ext}$ `UNI.School_Member`
5. `UNI.Research_Staff` $DISJ_{Ext}$ `TP.Student`
6. `UNI.Research_Staff` $DISJ_{Ext}$ `CS.Student`

   Analizzando la sorgente `Computer_Science`, dalle relazioni di ereditarietà, si possono ricavare gli assiomi:

7. `CS.Student` NT_{Ext} `CS.CS_Person`

8. `CS.Professor` NT_{Ext} `CS.CS_Person`

   L'insieme delle base extension che si ottiene, attraverso la definizione delle rule sopraelencate, è rappresentato in Figura 2.4.

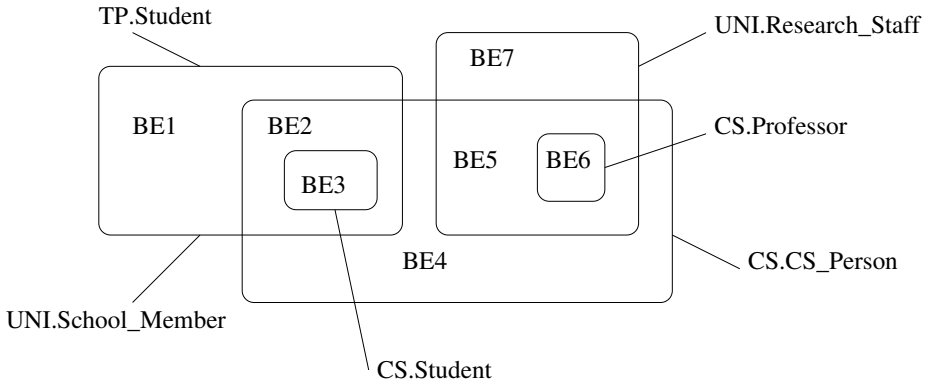

Figura 2.4: Base extension per la classe University_Person

L'insieme delle base extension può avere anche una rappresentazione di natura tabellare (Figura 2.5), dove vengono messe in risalto le singole classi locali componenti. Leggendo la tabella per colonne, si ha: '1', se la classe della riga corrispondente, comprende nella propria estensione la bese extension; '0' in caso contrario. Il procedimento che porta alla individuazione delle base extension è di tipo incrementale: partendo da un insieme non completo di rule estensionali (non si pretende che il progettista sia in grado di specificare, sin dal principio, l'intero insieme degli assiomi estensionali) è possibile calcolare il relativo insieme di base extension, che può essere utilizzato per la deduzione di nuova conoscenza estensionale, da integrare a quella già sfruttata. Il processo viene ripetuto iterativamente, fino al raggiungimento di una soluzione soddisfacente. Alla base dell'algoritmo che permette di individuare le base extension [36], vi è il concetto di *existence requirement*: un'espressione logica che deve essere soddisfatta da almeno una base extension dell'insieme calcolato in base alle rule estensionali definite. Deve essere verificato in ogni momento la : $State^{t}_{ER} \neq \emptyset$, dove ER è un existence requirement. Occorre naturalmente prevedere un algoritmo che sia in grado di controllare la correttezza degli assiomi specificati dal progettista, e di individuare eventuali incongruenze.

Base Extension	1	2	3	4	5	6	7	8	9	10	11	12
University_Worker				X	X	X	X	X		X	X	X
University_Student	X	X	X	X	X	X						
School_Member	X	X	X	X	X	X						
CS_Person		X	X	X	X			X	X	X		X
Student		X		X								
Professor												X
Research_Staff										X	X	X

Figura 2.5: Tabella delle base extension

2.3.4 Generazione della gerarchia estensionale

Ad ogni classe globale viene associata una *gerarchia estensionale* (o *concept lattice*), costruita sulla base delle informazioni relative alle base extension, e costituita da un insieme di *classi virtuali* totalmente nuove.

Lo scopo principale è 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 le query. Si ricava in questo modo un insieme di istanze che devono essere combinate attraverso determinate chiavi di join, per arrivare a ricostruire le entità descritte nelle varie sorgenti.

Infatti se ci si limitasse, per ottenere la risposta globale, alle informazioni intensionali presenti nella Mapping Table, si recupererebbero le proprietà richieste dalla Basic Query, ma non si riuscirebbe a ricostruire gli oggetti virtuali rappresentanti le entità descritte in termini globali. La gerarchia estensionale permette il passaggio da un'informazione di tipo intensionale ad una di tipo estensionale. Le classi virtuali che la costituiscono sono caratterizzate dall'avere:

- *intensione*: corrisponde all'intersezione degli schemi delle base extension che le costituiscono;
- *estensione*: corrisponde all'unione di tutte le base extension che hanno almeno tutti gli attributi presenti nell'intensione della classe.

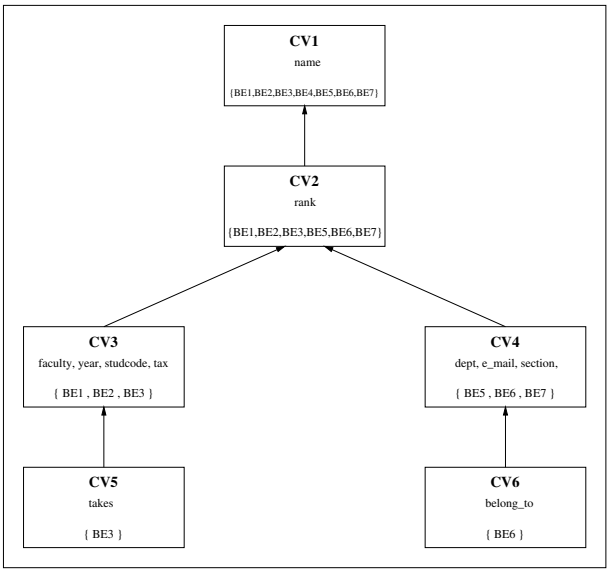

Figura 2.6: Gerarchia estensionale di University_Person

Le varie classi virtuali vengono organizzate in una struttura gerarchica, sulla base di relazioni di ereditarietà, col fine di ottimizzare la procedura di ricerca della *classe virtuale target*.
Facendo riferimento alle rule estensionali ed alle base extension individuate in precedenza, per la classe globale University_Person, viene generata la gerarchia estensionale rappresentata in Figura 2.6. Il processo che per ogni classe globale, partendo dalle basse extension, arriva a creare il concept lattice rappresentante la gerarchia estensionale è stato codificato in un algoritmo ed esaurientemente descritto in [21].
Per capire l'utilità della gerarchia estensionale, si consideri la seguente query:

```
select name, e_mail, belong_to
from University_Person
```

Prendendo in considerazione solo la Mapping Table, si nota che nessuna delle classi locali di University_Person possiede entrambi gli attributi `e_mail` e `belong_to`. Sfruttando solamente la conoscenza intensionale fornitaci dalla Mapping Table (Figura 2.3), potremmo arrivare a generare una risposta recuperando `e_mail` da `UNI.Research_Staff` e `belong_to` da `CS.Professor`, ma tale risposta risulterà inevitabilmente incompleta, in quanto ottenuta facendo riferimento a istanze singole che non hanno un criterio di fusione definito.

Avendo a disposizione anche la conoscenza estensionale fornita dalla gerarchia estensionale in Figura 2.6, si può pervenire ad una risposta più completa. Dalla gerarchia estensionale si evince infatti, che la classe virtuale $CV6$ ha nella propria intensione tutti gli attributi cercati. L'estensione di questa classe è fornita dalla base extension $BE6$ e quindi fondendo le classi `UNI.Research_Staff`, `CS.CS_Person` e `CS.Professor` (che costituiscono l'estensione della base extension $BE6$), si ottiene la risposta desiderata (priva di duplicazioni, che determinerebbero una risposta errata). La minimalità di tale risposta è dovuta allo sfruttamento delle informazioni derivanti dalle relazioni estensionali introdotte.

2.3.5 Schema Virtuale e Gerarchia Estensionale

Nella sezione precedente una classe globale è stata caratterizzata tramite il suo insieme 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 sono 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 2.3.9 (Schema Virtuale) *Data una classe globale G ed un suo* set di base extension $\mathcal{B} = (\boldsymbol{B}, F)$*, 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. $\forall B \in \boldsymbol{B}, \exists V \in \boldsymbol{V} : B \in EST(V)$

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 2.3.10 (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:

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

Nella sezione seguente verrà mostrato come lo schema virtuale e la relativa gerarchia estensionale di una classe globale sono utili per l'eleborazione di una query posta dall'utente.

2.4 Query Processing

Il risultato della fase di integrazione degli schemi, eseguito dai due moduli preposti Global Schema Builder e Extensional Hierarchy Builder, è costituito da: lo Schema Globale, le Mapping Table, Le Join Table, le Base Extension e le gerarchie estensionali.
Il Query Manager, il modulo di MOMIS che si occupa della gestione delle query, utilizza tutte le informazioni, sia intensionali che estensionali, prodotte dalla fase di integrazione degli schemi, per realizzare la fase di *Query Processing*. La fase di Query Processing ha come fine ultimo l'ottenimento di una risposta corretta, e quanto più possibile completa e minima, alla query globale.

Lo svolgimento della tesi ha riportato allo studio di diverse tecniche e alla generazione di metodi di elaborazione delle interrogazioni in un sistema multidatabase come lo è MOMIS. Tali metodi si collocano nel modulo Query Manager nelle fasi di *Definizione del Query Plan* e di *Esecuzione della Query*. Inoltre viene proposto un metodi di Query Processing senza l'uso delle Base Extension.

La fase di *Query Proccessing* prevede, come è mostrato in Figura 2.7, l'esecuzione in sequenza di tre attività:

1. *Acquisizione della Query*;
2. *Definizione del Query Plan*;
3. *Esecuzione della Query*.

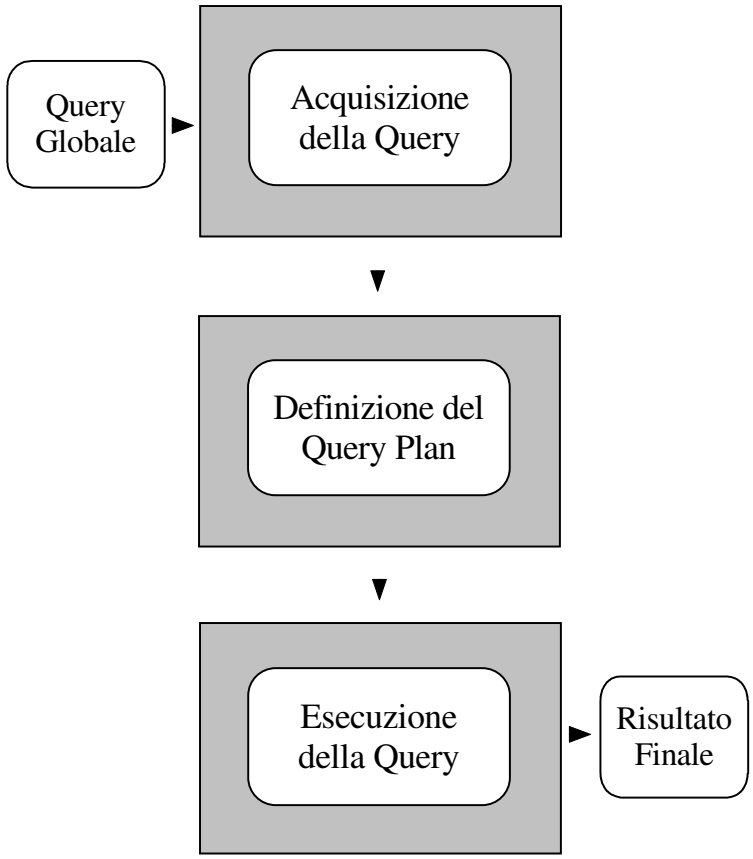

Figura 2.7: Attività di Query Processing

La prima attività è l'*Acquisizione della Query*, che a sua volta è suddivisa in tre momenti distinti.
Inizialmente avviene il *parsing e la validazione* della query, in cui si verifica la correttezza di essa, sia da un punto di vista sintattico che semantico. In questa attività viene impiegato un modulo di riconoscimento grammaticale, che ha il compito di verificare la correttezza sintattica della query rispetto alla grammatica OQL e di produrne poi un'immagine in memoria centrale. La validazione della query prevede la verifica della correttezza semantica di questa, accertando che le

classi e gli attributi coinvolti appartengano effettivamente allo schema integrato a cui la query è rivolta.
In secondo luogo avviene l'*Ottimizzazione semantica globale*. Questa é realizzabile se sono presenti regole di integritá inter-sorgenti definite sullo schema globale. Questi vincoli sono espressi con regole (rule) ODL_{I^3} e vengono impiegati da ODB_Tools per riformulare la query producendone una semanticamente equivalente, ma eseguibile in modo piú efficiente. Ad esempio vengono eliminati predicati ridondanti oppure vengono aggiunte nuove condizioni che possono abbreviare i tempi di risposta per la possibile presenza, nelle sorgenti, di indici sui predicati introdotti.
Infine vi è la *Normalizzazione della clausola where*. Alla fine delle due fasi precedenti, la clausola where è rappresentata da un'espressione booleana innestata ricorsivamente. Per poter proseguire, il Query Manager deve trasformare questa generica struttura booleana in forma normale congiuntiva. Avere la clausola where in questa forma, consente di ottimizzare la fase di esecuzione della query, in quanto vengono valutati il maggior numero di predicati in *and* e quindi viene minimizzata la quantità di risultati restituiti dalle sorgenti locali, con una conseguente velocizzazione della fase di integrazione. Dall'altro canto nell'algoritmo di Query Processing senza l'uso delle Base Extension, la forma CNF della query permette di individuare i predicati separabili e facilita il trattamento della query stessa.

La seconda attività svolta dal Query Manager nella fase di Query Processing, è la *Definizione del Query Plan*, in cui la query posta in termini globali viene ricondotta agli schemi locali delle singole sorgenti. In questa fase viene generato il *plan*, che contiene le informazioni per generare le query locali e successivamente per ricostruire la risposta globale.
Qui avviene il primo distacco nel trattamento della query secondo i due algoritmi.

- Nell'algoritmo di riscrittura con l'uso delle Base Extension (RWBE) il primo passo della Definizione del Query Plan che viene svolto è la decomposizione della Global Query (espressa sull'intero schema globale) in Basic Query, ovvero in una query contenente tutte le richieste rivolte ad una singola classe globale. In una Basic Query non sono ammessi: join espliciti tra classi diverse (è però possibile la navigazione attraverso aggregazioni ed associazioni per ricostruire oggetti complessi), subquery, operatori di ordinamento o di conversione, restituzione di strutture complesse. In questa fase di decomposizione, viene anche creata la *Global Assembler Query*, che si occupa di generare il risultato finale della Global Query, partendo dai risultati parziali delle singole Basic Query.
  Arrivati a questo punto la Basic Query viene analizzata, per ricercare gli attributi globali contenuti nei predicati di proiezione e di selezione. Il passo successivo consiste nell'esplorare la gerarchia estensionale alla ricerca delle

Classi Virtuali Target. Queste classi sono le classi virtuali più generali che dispongono di tutte le proprietà richieste. Quando le classi virtuali target individuate sono più di una, si seleziona la più generalizzata (cioè con estensione maggiore, in termini di numero di base extension). Essendo note le base extension interessate dalla query, sono automaticamente note le classi locali a cui rivolgersi per ottenere i dati.

Una volta determinati l'insieme delle base extension e l'insieme delle classi locali interessati, si procede ad una semplificazione (eliminazione di base extension,eliminazione di classi locali) del Query Plan, per minimizzare il numero di query da inviare alle sorgenti. Questa semplificazione si attua nei casi in cui si verificano delle dominazioni tra base extension, e nei casi in cui si hanno relazioni estensioni particolari (equivalenza o specializzazione, in cui nessuna delle due classi coinvolte determina un contributo informativo maggiore) tra classi locali.
Noto l'insieme minimo di classi da interrogare, si procede, con l'ausilio della Mapping Table, alla generazione delle Query Locali. Il Query Manager esprime le query locali in OQL, lasciando ai wrapper l'incombenza di tradurle nel linguaggio adatto in ogni sorgente.
In questa fase di generazione delle query locali, è necessario però considerare anche quei predicati che non sono mappati interamente in nessuna classe. Per questo motivo ad ogni Basic Query viene associata una struttura detta *Basic Query Assembler*, che contiene all'interno della propria clausola where i predicati "esclusi". Il Query Manager, dopo aver concluso la fase di *Esecuzione della Query* (che verrà descritta di seguito) ed aver quindi integrato i risultati provenienti dalle sorgenti locali, dovrà eseguire su di essi la Basic Query Assembler per ottenere la risposta desiderata.
Il piano d'accesso ottenuto fino ad adesso, può subire dei miglioramenti. Questi miglioramenti si possono ottenere analizzando eventuali predicati di selezione della Basic Query, che contengono valori di default. Un'altra semplificazione è ottenibile nel caso in cui sia possibile raggruppare più Local Query da inviare ad una stessa sorgente, effettuando localmente i join.
Inoltre è possibile realizzare l'*ottimizzazione semantica locale* (che consente di realizzare query meno onerose), sfruttando, tramite l'impiego di ODB-Tools, le regole di integrità definite sulle singole sorgenti. Quest'ultima fase di semplificazione è posta a livello di mediatore, in quanto non tutte le sorgenti potrebbero essere in grado di realizzarla.

- Nell'algoritmo di riscrittura senza l'uso delle Base Extension (RWOBE) nel primo passo avviene la riscrittura di tutti i predicati che contengono condizioni ligiche in predicati atomici. Successivamente la query viene trascritta in forma normale disgiuntiva (DNF), che facilita la generazioni degli

alberi binari rappresentativi. Quest'ultima trasformazione è neccessaria perchè nella fase precedente la query rappresentata in una forma booleana generica era stata trasformata in CNF inoltre inoltre la forma DNF permette la separazione dei predicati atomici. Segue il passo di riscrittura di tali predicati per tutte le classi locali interressati all'interrogazione. La trasformazione della *Global Query* (GQ) in *Local Query* (LQ) avviene tramite la trascrizzione dell'albero binario Master su ogni classe locale individuata nel passo precedente. Contemporaneamente alla generazione delle LQ si genera il Filtro Globale che sarà utilizzato durante la fase di Object Fusion dei risultati. Quest'ultimo è costituito da tutti i predicati residui individuati tramite l'aposito algoritmo ed è simile al *Basic Query Assembler*. Il piano di accesso ottenuto a questo punto viene semplificato ulteriormente in analogia all'algoritmo RWBE eliminando dalle LQ i predicati di default ed effettuando l'otimizzazione semantica locale.

La terza attività che costituisce il *Query Processing* è l'*Esecuzione della Query* [22], dove il Query Manager utilizza le informazioni del plan, generato nelle fasi precedenti, per ricomporre i risultati ottenuti dalle sottoquery, presentando all'utente una risposta integrata. Questa fase prevede tre livelli nell'algoritmo RWBE: esecuzione delle Local Query, esecuzione delle Basic Query ed esecuzione delle Global Query. Per quanto riguarda invece l'algoritmo RWNBE i livelli della terza attività sono: esecuzione delle Local Query, recupero delle Informazioni, applicazione del Filtro Globale ed esecuzione della Global Query.
Per realizzare questa attività il Query Manager sfrutta un DBMS in tutti e due algoritmi, che gli permette di creare delle tabelle in grado di memorizzare i risultati temporanei degli stadi intermedi della ricostruzione della risposta. Così il Query Manager può eseguire varie operazioni di ricostruzione e fusione attraverso query (join e outer join) poste su queste tabelle temporanee.

Come effettivamente il Query Manager realizzi l'*Esecuzione della Query* sarà esposto nelle Sezioni 5.3 e 3.4.4. In particolare, si presterà maggiore attenzione alle fasi di riscrittura e ricostruzione delle query e fusione dei risultati, mettendo in evidenza, essendo l'argomento di questa tesi, quali siano le informazioni sulla base delle quali sia possibile realizzare le operazioni di join ed natural outer join e quali algoritmi vengono usati per la generazione delle sequenze valide di operatori di natural outer join.

Capitolo 3

Query Processing in MOMIS

Introduzione

Il sistema mediatore MOMIS è stato studiato e sviluppato con l'intento di permettere ad un generico utente di formulare interrogazioni e ricerche ed in conseguenza di poter raccogliere informazioni in un contesto eterogeneo e distribuito. Perchè ciò sia possibile è neccessario risolvere molteplici conflitti intensionali, fornendo una rappresentazione unificata ed omogenea delle diverse sorgenti. In questo capitolo analizzeremo le problematiche inerenti la gestione della conoscenza intensionale ed all'elaborazione di espressioni di algebra relazionale. Infine presenteremo le definizioni in base alle quali definiremo nel successivo Capitolo 4 l'algoritmo di riscrittura delle query globali basato sui vocabolari delle classi locali e le conseguenti operazioni di Data Merging ed Object Fusion. Ci si è posti dunque il problema di studiare ed elaborare un metodo per l'esecuzione delle interrogazioni definite sullo schema globale. Le ricerche hanno portato alla compilazione di un algoritmo che a partire dalla query globale, formulata sullo schema globale del mediatore che ne rappresenta il vocabolario globale, individua le classi locali coinvolte nell'interrogazione, suddivide la query in query locali e le invia alle sorgenti corrispondenti. Successivamente recupera le informazioni contenute nelle risposte di ogni query locale e calcola il modo di combinarle per ottenere il risultato globale desiderato. Il nostro scopo è quello di calcolare una istanza dello schema globale paragonabile al concetto di Full Disjunction presente in letteratura.

Tale algoritmo si colloca nella fase di Definizione del Query Plan e nell'Esecuzione della Query ed è costituito dai seguenti passi:

1. Atomizzazione dei predicati logici contenuti nella query globale (GQ):

   - Individuazione dei predicati logici;
   - Scelta delle regole di riscrittura per i predicati atomici;
   - Riscrittura dei termini atomici;

2. Normalizzazione della GQ;
3. Generazione dell'albero Master rappresentativo;
4. Individuazione delle classi locali coinvolte nella riscrittura;
5. Riscrittura della GQ rispetto alle classi locali;
   - Individuazione dei predicati appartinenti alla GQ e le classi locali per cui tali predicati devono essere riscritti;
   - Generazione dei sotto-alberi per la classe locale idividuata in 4;
   - Generazione delle query locali;
6. Mapping logico;
7. Individuazione dei predicati residui e generazione del Filtro Globale (GF);
8. Recupero delle informazioni e fusione dei risultati (Object Fusion);
9. Generazione della Full Disjunction (Completa Disgiunzione, FD) ed esecuzione della GQ;

Prima di entrare nello specifico fornendo le particolarità di ogni passo dell'algoritmo come faremo nel paragrafo 4.3, diamo ora alcune definizioni che riteniamo importanti.

3.1 Istanza della Global Class

Il mediatore, tramite le strutture definite nel Capitolo precedente, rappresenta la conoscenza disponibile per quanto riguarda gli oggetti del mondo reale istanziati nelle sorgenti locali. Dobbiamo subito notare che tali oggetti del mondo reale non devono essere confusi con quelli che individuano l'oggetto stesso definiti in [37]. Infatti se lo stesso oggetto del mondo reale viene rappresentato in due sorgenti distinte esso deve essere rappresentato a livello del mediatore una sola volta mentre le sue due rappresentazioni nelle due sorgenti locali avranno sicuramente due diversi identificatori. Denotiamo con $\mathcal{O}$ un insieme di oggetti del mondo reale (detti

semplicemente *oggetti* nel seguito) rappresentati al livello del mediatore e introduciamo una funzione $\mathcal{E}$, chiamata extension, nel seguente modo: ($\mathcal{E} : \mathcal{L} \rightarrow 2^{\mathcal{O}}$). Questa funzione associa ogni classe locale L che appartine all'insieme di classi locali $\mathcal{L}$ ad un sottoiniseme di classi di $\mathcal{O}$ i cui oggetti sono rappresentati in L. Inoltre con $S(L)$ rappresentiamo lo schema della classe locale L e con A un'attributo di $S(L)$. Abbiamo adottato la notazione "dot" per denotare il valore di un attributo, cioè: se $o \in \mathcal{E}(L)$ e $A \in S(L)$ allora $o[L]$.A rappresenta il valore dell'attributo A nell'oggetto o appartenente all'estensione $\mathcal{E}(L)$.

In questa sezione investigheremo su alcune tecniche adottando come riferimento per uno schema e la corrispondente query il modello relazionale. Cioè, come detto sopra, ogni classe locale L rappresenta un nome di una relazione con schema $S(L)$ e ogni oggetto è rappresentato in L tramite una funzione di extension (estensione) $\mathcal{E}$. In altre parole ogni $o \in \mathcal{E}(L)$ è rappresentato da una tupla $t_L(o)$ nello schema $S(L)$. In particolare associamo ogni estensione $\mathcal{E}(L)$ di L con la relazione l di L che ha la seguente estensione $\{t_L(o) \mid o \in \mathcal{E}(L)\}$. Introduciamo anche la notazione $l^{\mathcal{G}}$ per la relazione con lo schema $S^{\mathcal{G}}(L)$ ottenuta rinominando gli attributi locali di l in quelli globali di $\mathcal{G}$ seguendo l'informazione contenuta nella Mapping Table $\mathcal{M}$ e convertendo opportunamente i dati.

Ad esempio se consideriamo le due classi locali L_1 ed L_2 nel seguente modo:

L_1(firstn, lastn, year, e_mail)
L_2(name, e_mail, dept_code, s_code)

dopo la fase di integrazione otterremo la classe globale G:

	Name	**E_mail**	**Section**	**Year**	**Dept**
L_1	firstn **and** lastn	e_mail	null	year	null
L_2	name	e_mail	s_code	null	dept_code

che dunque ha il seguente schema globale:
$S(G) = (Name, E\_mail, Section, Year, Dept)$, mentre gli schemi delle classi locali rispetto allo schema globale sono:
$S(L_1) = (Name, E\_mail, Year)$
$S(L_2) = (Name, E\_mail, Dept, Section)$.

Per meglio comprendere il concetto diamo anche un esempio con istanze l_1 e l_2:

l_1

firstn	lastn	e_mail	year
Rita	Verde	pv@i.it	2
Ada	Rossi	ra@i.it	1

l_2

name	e_mail	dept_c	s_code
Rossi_Ada	ra@i.it	dept1	12345
Po_Ugo	up@i.it	dept1	2345

tali relazioni dopo le fasi di recupero delle informazioni dalle singole sorgenti locali, di conversione degli attributi locali e di modifica dei dati risultano essere con le rispettive notazioni:

l_1^G

Name	E_mail	Year
Rita Verde	pv@i.it	2
Ada Rossi	ra@i.it	1

l_2^G

Name	E_mail	Dept	Section
Ada Rossi	ra@i.it	dept1	12345
Ugo Po	up@i.it	dept1	2345

Poichè sono possibili diversi modi per istanziare lo stesso oggetto, non ci soffermeremo sul problema che riguarda l'integrazione dell'informazione che proviene da sorgenti diverse trattato nel capitolo precedente. Supponiamo che le classi locali soddisfano la proprietà di *omogeneità semantica*, cioè tuple che si riferiscono allo stesso oggetto istanziato in diverse classi siano rappresentate nello stesso modo e non vengono mai confuse (vedi Figura 3.1). Formalmente per ogni coppia di classi locali (L_1, L_2), per ogni $o \in \mathcal{E}(L_1) \cap \mathcal{E}(L_2)$ e per ogni $A \in S^{\mathcal{G}}(L_1) \cap S^{\mathcal{G}}(L_2)$ segue che:

$$t^{\mathcal{G}}_{L_1}(o)[A] = t^{\mathcal{G}}_{L_2}(o)[A]$$

oppure:

$$o[L_1].A = o[L_2].A \qquad \begin{array}{l} \forall L_1, L_2, \\ \forall o \in \mathcal{E}(L_1) \cap \mathcal{E}(L_2), \\ \forall A \in S^{\mathcal{G}}(L_1) \cap S^{\mathcal{G}}(L_2) \end{array}$$

Ad esempio, considerando le istanze l_1, l_2 e le loro elaborazioni in l_1^G ed l_2^G dell'esempio precedente, possiamo osservare che l'oggetto o presente in entrambe le sorgenti, per la proprietà di omogeneità semantica ha lo stesso valori per l'attributo E_mail, cioè $l_1^G.E\_mail = l_2^G.E\_mail$: vedi Figura 3.1.
Inoltre ipotizziamo che sia soddisfatta anche l'omogeneità semantica indiretta, cioè se le coppie di classi locali (L_1, L_2) e (L_2, L_3) soddisfano la proprietà di omogeneità semantica allora questo è valido anche per (L_1, L_3).

Più formalmente dato un insieme di classi globali $\mathcal{G}$ allora una classe $G \in \mathcal{G}$ è una relazione con schema $S(G)$. Data una funzione estensione $\mathcal{E}$ definita su un insieme di oggetti $\mathcal{O}$ e le corrispondenti relazioni $l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}}$ noi possiamo derivare la relazione globale g nel seguente modo:

$$g = \{t_G(o) | o \in \mathcal{O}\},$$

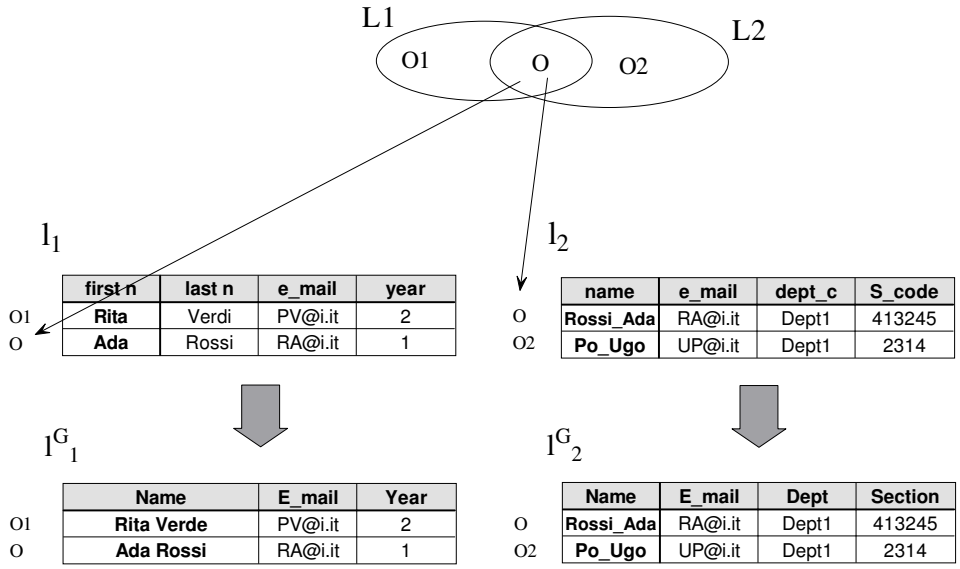

Figura 3.1: Integrazione degli schemi

dove la tupla $t_G(o)$ è ottenuta applicando la proprietà di omogeneità semantica. Più formalmente:

$$\forall A \in \mathbf{S(\mathcal{G})} : t_G(o)[A] = \begin{cases} t_L^G(o)[A] & \text{se} \exists L \in \mathcal{L} : A \in S^G(L), o \in \mathcal{E}(L) \\ \text{null} & \text{in caso contrario} \end{cases} \quad (3.1)$$

Per il nostro problema consideriamo di esaminare query congiuntive specificate sul vocabolario delle classi globali rappresentato dallo schema globale $S(\mathcal{G})$. Tali query saranno denominate con $\hat{Q} = \sum_{i=1}^{n} \tau_i$ dove τ_i è una condizione positiva di selezione, cioè senza negazioni. La risposta ad una query congiuntiva $\hat{Q}$ rispetto alla relazione globale g è denotata con $\hat{Q}^g$ e corrisponde al segunte insieme: $\{t_G(o) \in g \mid \hat{Q}(t_G(o)) = true\}$.

Lo scopo del Query Processing in generale è la generazione di un valido piano di esecuzione di una query. A questo proposito assumiamo di avere delle generiche interrogazioni SQL-like sullo schema globale $S(\mathcal{G})$ della forma :

```
select  AL
```

```
from    G
where   Q
```

oppure algebricamente:

$$\pi_{AL}(\sigma_{Q}(\mathcal{G})) \tag{3.2}$$

dove AL è una lista di attributi globali e Q è la condizione di selezione della query specificata tramite una espressione booleana di condizioni positive, cioè senza negazioni. In altre parole Q ha la seguente forma $Q = Q' \wedge (\wedge_{A \in AL}$ A not *null*$)$[1]. Per generare la risposta ad una query globale dunque si dovra prima riscrivere la Q sugli schemi delle classi locali. Siccome durante tale traduzione espandiamo lo schema globale $\mathcal{G}$ nei termini di $\mathcal{L}$, nella fase di recupero delle informazioni per ricostruire la risposta dobbiamo prendere in considerazione il problema dell'Object Fusion. Questo richiede di identificare delle istanze che fanno riferimento allo stesso oggetto e fonderli in un'unica istanza. Per la risoluzione di tale problema la maggioranza degli approcci assumono che esiste una *condizione di join* definita sul vocabolario dello schema globale $\mathcal{G}$ tra ogni coppia di relazioni che si sovrappongono. Noi introduciamo la seguente definizione di *condizione di join* che è strettamente legata con la notazione di oggetti introdotta per il nostro approccio.

Definizione 3.1.1 (Condizione di Join (Join Map)) *Sia $\mathcal{G}$ un insieme di classi globali, $G \in \mathcal{G}$ e siano $\mathcal{L}$ l'insieme di classi locali corrispondenti a G. Data una coppia di classi locali $(L_1, L_2) \in \mathcal{L}$ la condizione di join tra L_1 e L_2, denotata con $F_{1,2}$, è la condizione espressa sullo schema $S^{\mathcal{G}}(L_1) \cup S^{\mathcal{G}}(L_2)$ tale che per ogni estensione $\forall \mathcal{E}, \forall t_1 \in l_1^{\mathcal{G}}, \forall t_1 \in l_2^{\mathcal{G}}$, $F_{1,2}(t_1, t_2)$ è vera se e solo se $\exists o \in \mathcal{E}(L_1) \cap \mathcal{E}(L_2)$ per cui $t_1 = t_{L_1}(o)$ e $t_2 = t_{L_2}(o)$.*
*Con $\mathcal{F}$ indicheremo l'insieme di condizioni di join tra tutte le coppie di classi locali che appartengono ad $\mathcal{L}$. $\mathcal{F}$ è chiamato **Join Table** della classe globale G. Formalmente:*
$\mathcal{F}$ = $\{F_{i,j} | F_{i,j}$ è una condizione di join tra le classi L_i e L_j, con $(L_i, L_j) \in \mathcal{L}$ e $i < j, 1 \leq i < n, 1 < j \leq n\}$.

Ad esempio, in riferimento alla Figura 3.1, per il nostro esempio la Join Map è la seguente:

Join Map JM(L_1, L_2)
$L_1.Name = L_2.Name$

[1]Notiamo che nonostante noi non supportiamo la negazione, eseguire tali condizioni è solamente rivolto all'includere nella condizione di selezione della query gli attributi inclusi sia nella condizione di selezione che nella costruzione della risposta.

Attraverso le condizioni di join espresse tramite $F_{i,j}$ ci poniamo come obbiettivo di calcolare una particolare risposta che estrapola dalle sorgenti locali la maggiore quantità di informazioni possibile. L'approccio ed il metodo sono esposti qui di seguito.

3.2 Full Disjunction

Un problema molto simile al nostro è stato trattato da A. Rajaraman e J. Ulman in [38]: quello di fondere insieme la grande quantità d'informazione proveniente da molteplici sorgenti eterogenee e distribuite in una struttura che può rendere tale informazione utilizzabile. Gli autori affermano che il problema è legato al fatto di calcolare il natural outer join tra tante relazioni in modo da preservare tutte le possibili connessioni tra le relazioni. Il risultato di tale calcolo era stato precedentemente chiamato ***"Full Disjunction"*** (FD, Completa Disgiunzione) da Galindo-Legaria in [39]. Nei lavori appena citati è stato svolto lo studio di quando la Full Disjunction può essere calcolata tramite una sequenza di natural outer join. La risposta a tale quesito coinvolge il concetto introdotto da Fagin [40] della rappresentazione di schemi di relazioni tramite ipergrafi detto "ipergrafo γ-aciclico".

In questa sezione mostreremo che la relazione globale g definita dall'equazione 3.1 corrisponde alla notazione di Full Disjunction delle relazioni locali calcolata rispetto le condizioni di join espresse tramite $\mathcal{F}$.
Per poter dare la definizione della Full Disjunction abbiamo bisogno di richiamare velocemente i concetti di schemi e tuple coinvolte e della sussunzione tra tuple.

Si definisce uno schema come un insieme finito di attributi. Una tupla che appartiene ad uno schema S si ottiene assegnando dei valori agli attributi di S. Lo schema della tupla t si denota con *schm(t)*. Alcune tuple possono contenere dei valori nulli e le tuple che contengono solamente tali valori vengono denominate come tuple nulle. Due tuple t_1 e t_2 che appartegono rispettivamente a S_1 e S_2 possono essere concatenate se gli schemi sono disgiunti oppure se gli assegnamenti coincidono per tutti gli attributi dell'intersezioni $S_1 \cap S_2$ in tutte e due le tuple. La concatenazione è una tupla t_c tale che $t_c \in S_1 \cup S_2$.
Se $t \in S$ possiamo ottenere $t' \in S'$ con $S \subseteq S'$ aggiungendo a t tutti i termini nulli di $S' - S$. Questo procedimento viene chiamato *padding*.
Una relazione l sullo schema S è un insieme finito di tuple t_i con schema S(t_i). Denotiamo con *schm(l)* lo schema della relazione l. In relazione con questi concetti possiamo dire che un data base è un insieme di relazioni i cui schemi sono mutuamente disgiunti. Queste relazioni saranno chiamati *Relazioni Base*. Diciamo che una tupla t_1 sussume un'altra t_2 se esse hanno lo stesso schema ed inoltre t_2 contiene più valori nulli di t_1 e t_1 coincide con t_2 per tutti gli attributi non nulli.

In altre parole t_1 è ottenuta da t_2 sostituendo uno o più valori nulli con valori concreti. Questo vuol dire che tali tuple sussunte rappresentano una ridondanza nelle relazioni che fanno parte della risposta che cerchiamo e dunque devono essere rimosse. La rimozione delle tuple sussunte di una relazione l ritorna tutte le tuple di l che non sono incluse in nessuna delle altre tuple di l. Possiamo definire che date due relazioni l_1 e l_2 l'unione minima tra di loro è espressa da $l_1 \bigoplus l_2 := (l_1 \biguplus l_2)\downarrow$ e contiene tutte le tuple del risultato dell'outer union rimuovendo le tuple sussunte.

Definizione 3.2.1 (Full Disjunction) *Sia G una classe globale e sia $\mathcal{L}$= $\{L_1, \ldots, L_n\}$ l'insieme di classi locali corrispondenti. Sia $\mathcal{E}$ una estensione e siano $\{l_1^G, \ldots, l_n^G\}$ le corrispondenti relazioni le cui tuple non contengono valori nulli. Sia $\mathcal{F}$ una Join Table di G.*
Diremo che una relazione l con lo schema $S(G)$ è la Full Disjunction di $\{l_1^G, \ldots, l_n^G\}$ basata su $\mathcal{F}$ se soddisfa le seguenti condizioni:

1. *Non ci sono ridondanze ed inconsistenze, cioè non esistono delle tuple sussunte.*

2. *Le tuple di l provvengono da pezzi connessi di $l_i^{\mathcal{E}}$: sia t una tupla di l, allora esiste un sottoinsieme di relazioni connesse di $l_i^{\mathcal{E}}$ tale che t, ristretta ai suoi componenti non nulli, è il risultato dell'operazione di join tra tutte queste relazioni effettuato sulla base della Join Table.*

3. *Tutte le connessioni sono presenti:*

   a) *siano $t_1, \ldots, t_k$ tuple scelte dalle relazioni distinte $l_{i_1}^{\mathcal{E}}, \ldots, l_{i_k}^{\mathcal{E}}$ rispettivamente, tali che $R_{i_1}, \ldots, R_{i_k}$ formano un ipergrafo connesso (la definizione del quale sarà data di seguito).*

   b) *sia t la tupla di l che si mappa (corrisponde) con ognuna delle tuple t_i negli attributi che compaiono in ogni $l_{i_1}^{\mathcal{E}}, \ldots, l_{i_k}^{\mathcal{E}}$ e abbia dei valori nulli negli altri attributi sullo schema di $S(G)$[2].*

   Allora t è sussunta da qualche tupla di l.

Come dimostrato in [39, 40] la Full Disjunction è unica. Nel nostro contesto si può verificare che la relazione globale g definita dall'equazione 3.1 soddisfa le condizioni di Full Disjunction, indipendentemente dalla Join Table considerata. Il nostro problema fondamentale dunque sarà quello di calcolare la Full Disjunction e questo fatto costituisce uno degli argomenti principali della tesi. Nel prossimo

[2]Notiamo che secondo la proprietà di omogenetà semantica questa tupla è definita correttamente.

Capitolo 4 tratteremo il calcolo della Full Disjunction basato sull'uso del solo operatore SQL di natural outer join (NOJ).

Ad esempio, riconsiderando le relazioni l_1^G e l_2^G introdotte nell'esempio della sezione precedente abbiamo che la Full Disjunction FD(l_1^G, l_2^G) è:

Name	**E_mail**	**Year**	**Dept**	**Section**
Rita Verde	pv@i.it	2	null	null
Ada Rossi	ra@i.it	1	Dept1	12345
Ugo Po	up@i.it	null	Dept1	2345

Nel seguito di questo capitolo riportiamo una definizione alternativa ed equivalente della Full Disjunction, basata sull'algebra relazionale ed introdotta da Galindo-Legaria in [39].
Successivamente discuteremo delle possibili ottimizzazioni algebriche che si possono effettuare in presenza della Full Disjunction.

3.3 Definizione algebrica di Full Disjunction

In questa sezione esporremo una rappresentazione alternativa della FD basta sull'algebra relazionale ed introdotta da Galindo-Legaria in [39]. Siccome nel seguito faremo un confronto con il concetto di Outerjoin introdotto in [39] per semplicità riportiamo le definizioni con la simbologia introdotta in tale lavoro.
Nel lavoro citato Galindo-Legaria propone un metodo di riscrittura di una query che contiene operatori di outer join trasformando la sua condizione di selezione in una sequenza che include al suo interno soltanto operatori di join naturale e outer union. Quest'ultimo operatore viene definito cui di seguito.

Definizione 3.3.1 (OuterUnion.) *Siano R_1 e R_2 due relazioni con i rispettivi schemi schm(R_1) = S_1 e schm(R_2) = S_2, definiamo l'*OuterUnion*, denotato come $R_1 \biguplus R_2$, l'operazione di generazione dello schema di $S_1 \cup S_2$, l'esecuzione del* padding *sullo schema risultante ed il successivo inserimento di tutte le tuple per ogni relazione; dopodichè si calcola l'unione dell'insieme risultante.*

Date due relazioni R_1 e R_2 con schemi disgiunti S_1 e S_2 rispettivamente e dato un predicato P tale che: schm(P)$\subseteq$ (schm(R_1)$\cup$schm(R_2)) allora denotiamo l'operatori di *join*($\bowtie$), *antijoin*($\triangleright$), *left outerjoin*($\rightarrow$) e *right outerjoin*($\leftarrow$) rispettivamente nel seguente modo:

$$R_1 \bowtie^p R_2 = \{(t_1, t_2) : t_1 \in R_1, t_2 \in R_2, p(t_1, t_2)\}$$

$$R_1 \triangleright^p R_2 = \{t_1 : t_1 \in R_1, (\neg\exists t_2 \in R_2), (p(t_1, t_2))\}$$

$$R_1 \rightarrow^p R_2 = R_1 \bowtie^p R_2 \biguplus R_1 \triangleright^p R_2$$

$$R_1 \leftarrow^p R_2 = R_2 \rightarrow^p R_1$$

Utilizzando questi due operatori insieme all'operatore di join possiamo definire l'operazione di *full natural outer join* come segue:

$$R_1 =\bowtie= R_2 = R_1 \bowtie R_2 \biguplus R_1 \triangleright_p R_2 \biguplus R_2 \triangleright_p R_1 \tag{3.3}$$

Le query in generale possono essere viste come espressioni di algebra relazionale. Di solito le rappresentiamo come un albero di operatori, le foglie del quale rappresentano variabili relazionali mentre i nodi interni rappresentano operatori di join, outerjoin ed altri operatori algebrici.

Definizione 3.3.2 (Query join-disgiuntive.) *Una query è join-disgiuntiva se il risultato dell'esecuzione della query contiene l'unione minima*[3] *dei risultati dell'esecuzione dei diversi predicati di join al suo interno.*

Per poter dare la definizione formale della query join-disgiuntiva faremo riferimento alle definizioni grafi deinite nel prossimo Capitolo 4 e in [40, **?**]. In generale diremo che una grafo è una coppia (V,E). Dove V è l'insieme dei nodi ed E è l'insieme degli archi. Il calcolo del risultato di una query rappresentata tramite un grafo si può denotare come $\bowtie(G) = \sigma_{p_1 \wedge \ldots \wedge p_n}(R_1 \times \ldots \times R_m)$, dove $p_1 \wedge \ldots \wedge p_n$ rappresentano i nomi degli archi e gli $R_1 \times \ldots \times R_m$ sono gli elementi dei nodi. Possiamo definire che dato un grafo G = (V,E) ed un sottoinsieme V' di V ($V' \subseteq V$) il grafo indotto da G, denotato come G—$_{V'}$, è (V',E'), dove E' = $\{(u,v) : (u,v) \in E, u \in V', v \in V'\}$. A questo punto siamo in grado di dare la definizione di query join-disgiuntiva secondo Galindo-Legaria.

Definizione 3.3.3 *Sia G = (V,E) un grafo che rappresenta una query Q e siano* $\{V_1 \ldots, V_n\} \in V$ *un insieme di nodi per cui risulta che ogni grafo indotto G—*$_{V_1}$ *...* *G—*$_{V_n}$ *sia un grafo connesso. Allora la query ottenuta valutando il risultato di ogni grafo indotto separatamente viene chiamata* join-disgiuntiva.

Nel [39] Galindo-Legaria da le seguenti regole di riscrittura della condizione di selezione di una query join-disgiuntiva, il grafo della quale risulta essere γ-aciclico, tramite le quali ottiene una sequenza di operatori di join ed unione minima per calcolare la Full Disjunction.

$$R_1 \rightarrow^p R_2 = R_1 \bowtie^p R_2 \oplus R_1; incui R_1 = R_1 \downarrow e R_2 = R_2 \downarrow \tag{3.4}$$

$$R_1 \leftrightarrow^p R_2 = R_1 \bowtie^p R_2 \oplus R_1 \oplus R_2; incui R_1 = R_1 \downarrow e R_2 = R_2 \downarrow \tag{3.5}$$

[3] La definizione di questo termine sarà data in seguito.

$$(R_1 \oplus R_2) \bowtie^p R_3 = R_1 \bowtie^p R_3 \oplus R_2 \bowtie R_3; incui R_3 = R_3 \downarrow \quad (3.6)$$

Per effettuare il calcolo della Full Disjunction in [38, 39], tramite l'operatore di natural outer join (NOJ), si fa l'ipotesi di aciclicità del grafo che rappresenta la condizione di selezione di una query. Tale ipotesi è inapplicabile nel nostro contesto poichè l'integrazione dell'informazione che noi consideriamo ci ritorna uno schema globale su cui le query risulteranno sempre cicliche. Per questo motivo nel prossimo capitolo presenteremo un metodo alternativo per il calcolo della Full Disjunction che funzioni in tutti i casi di ciclicità.

3.4 Riscrittura della Global Query rispetto alle sorgenti

Nelle sezioni precedenti abbiamo considerato una generica Global Query rappresentata nel seguente modo:

$$\sigma_F(FD(l_1, \ldots, l_n))$$

abbiamo definito l'operatore di Full Disjunction e illustrato una tecnica algebrica per calcolare la Full Disjunction basata sull'operatore di natural outer join (NOJ). In questa sezione vogliamo affrontare il problema della riscrittura della Global Query rispetto alle sorgenti locali (*query rewriting*) che consiste nel tradurre e "spingere", il più possibile, le condizioni espresse in F a livello delle sorgenti locali. In altre parole, vogliamo riscrivere la condizione F dal contesto globale al contesto locale di ogni singola sorgente.

Formalmente, denotiamo con $S_i(F)$ il mapping (riscrittura) di F per la classe L_i; sulla classe locale L_i verrà quindi eseguita l'interrogazione $\sigma_{S_i(F)}(L_i)$; il risultato di questa interrogazione deve essere quindi trasformato rispetto allo schema globale: denotiamo il risultato di tale operazione con $\sigma^{\mathcal{G}}_{F_i}(l_i)$. Il risultato della query globale si ottiene effettuando la Full Disjunction di tali risultati ed applicando un *Filtro Globale* F_r costituito da tutti i *predicati residui* che non sono stati pienamente espressi a livello locale. In definitiva la Global Query verrà riscritta come:

$$\sigma_{F_r}(FD(\sigma^{\mathcal{G}}_{S_i(F)}(l_1), \ldots, \sigma^{\mathcal{G}}_{S_n(F)}(l_n))) \quad (3.7)$$

Il procedimento è il seguente. Primo, si considera per la condizione F la sua Forma Normale Disgiuntiva (DNF) (vedi sezione3.4.1). Secondo, in

sezione 3.4.3, si riporta un risultato di letteratura molto interessante: se F è in DNF allora la riscrittura per le differenti classi può essere effettuata *separatamente*. Terzo, in sezione 3.4.4, consideriamo la riscrittura rispetto ad una singola sorgente e discutiamo dei problemi che sorgono quando la sorgente non ha la *capacità* di eseguire le operazioni espresse nella query globale. Infine, in sezione 3.4.5, verrà illustrato il procedimento per determinare le riscritture rispetto alle singole sorgenti ed il Filtro Globale dei predicati residui.

3.4.1 Normalizzazione della clausola where

Atomizzazione dei predicati

Il primo passo da effettuare durante il Query Processing nel nostro contesto è la divisione dei predicati che fanno parte della condizione di selezione e che contengono delle congiunzioni o delle disgiunzioni logiche in predicati atomici. Tale devisione è necessaria poichè in questo modo riusciamo a dividere i predicati che sono supportati da una certa sorgente da quei predicati che non lo sono. Un predicato atomico è tale se non può essere suddiviso in altri predicati più semplici. In altre parole dato un generico predicato $P = \Phi(F, E)$, dove Φ è un generico operatore sul predicato, F è il campo di destinazione del risultato ed E è una espressione da valutare, tale predicato P è atomico se in E non ci sono predicati logici di AND o OR che possono essere rimossi senza alterare il significato dell'operatore. Nel caso generale questa fase richiede una definizione di regole di riscrittura dei predicati stessi. Per decomporre i generici predicati in atomici (dove necessario) applichiamo la proprietà distributiva degli operatori. Si noti che l'operazione di riscrittura non è sempre possibile [**?**]. Dobbiamo inoltre stare attenti nella traduzione perchè l'AND non sempre gode della proprietà distributiva su certi operatori. Per esempio le seguenti due trasformazioni sono equivalenti:

Contains(title, multiprocessor AND (distributed (W) system))
=
Contains(title,multiprocessor) AND Contains(title, distributed (W) system)

Ma non è vera la seguente trascrizione:

Contains(title,(multiprocessor AND distributed) (W) system)
=
Contains(title,multiprocessor(W) system) AND Contains (title, distributed (W) system)

Per spiegare tutto ciò basta osservare che i vari operatori godono di diverse proprietà e che l'operatore W (significa che l'attrib1 deve essere ad una distanza massima di una parola dall'attrib2) non gode della proprietà distributiva. Nel nostro caso supponiamo che la condizione *where* della Global Query sia formata a partire da predicati del tipo [attrib1 *operatore* val] oppure [attrib1 *operatore* attrib2] dove *operatore* e uno dell'insieme $\{<, =, >, \leq, \geq, \neq$, etc.$\}$. Quindi tutti i predicati presenti sono già atomici.

Introduciamo la definizione della forma normale disgiuntiva (DNF).

Definizione 3.4.1 (Forma Normale Disgiuntiva.) *Una formula booleana viene detta in forma normale disgiuntiva o DNF quando è formata da una sommatoria di termini, ciscuno dei quali è costituito da un prodotto di predicati atomici P.*

Per ottenere la forma normale, la clausola where viene analizzata ricorsivamente e trasformata utilizzando i **teoremi dell'algebra di commutazione**[**?**]:

- Valutazione di una clausola *NOT*:

(T1)	$\overline{x \vee y} = \bar{x} \wedge \bar{y}$	(De Morgan)
(T1')	$\overline{x \wedge y} = \bar{x} \vee \bar{y}$	
(T2)	$\overline{\bar{x}} = x$	(Involuzione)

  Per quanto riguarda le espressioni *not*, utilizzando i teoremi elencati, queste possono essere considerate *and* o *or* e valutate come i due casi descritti in seguito.

- Valutazione di una clausola *AND*:

(T3)	$x \wedge 1 = x$	(Identità)
(T4)	$x \wedge 0 = 0$	(Elemento nullo)
(T5)	$x \wedge x = x$	(Idempotenza)
(T6)	$x \wedge \bar{x} = 0$	(Complemento)
(T7)	$x \wedge (x \vee y) = x$	(Assorbimento)

  I teoremi precedenti sono valutati nell'ordine descritto e, se nessuno è applicabile, i due fattori in and vengono trasformati in modo indipendente.

- Valutazione di una clausola *OR*:

(T3')	$x \vee 0 = x$	(Identità)
(T4')	$x \vee 1 = 1$	(Elemento nullo)
(T5')	$x \vee x = x$	(Idempotenza)
(T6')	$x \vee \bar{x} = 1$	(Complemento)
(T7')	$x \vee (x \wedge y) = x$	(Assorbimento)

Nel caso in cui questi teoremi non siano applicabili viene utilizzata, quando è possibile, la proprietà distributiva che consente la scomposizione della clausola in due fattori in *and* analizzabili separatamente:

(T8) $(x \wedge y) \vee z = (x \vee z) \wedge (y \vee z)$ (Proprietà distributiva)
(T8') $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Se (T8) e (T8') non sono applicabili, i due operandi vengono tradotti ricorsivamente, prima di applicare nuovamente la proprietà distributiva alla nuova clausola *or* così ottenuta.

Il seguente esempio evidenzia la trasformazione in forma normale disgiuntiva di una espressione booleana, dove x, y, v, w rappresentano i predicati semplici .

$$((\overline{x \wedge y}) \wedge v) \wedge (w)$$

1. l'analisi inizia dall'*and* esterno e, non essendo applicabile la proprietà distributiva (T8), i due operandi vengono tradotti ricorsivamente;
   (a) il primo fattore $((\overline{x \wedge y}) \wedge v)$ viene trasformato tramite (T1) e succesivamente (T8) generando :
   $(\bar{x} \wedge v) \vee (\bar{y} \wedge v)$
   (b) il secondo fattore rimane w;
2. l'*and* iniziale diventa quindi:
   $((\bar{x} \wedge v) \vee (\bar{y} \wedge v)) \wedge w$
3. viene riapplicata la proprietà (T8) che genera la forma finale:
   $(\bar{x} \wedge v \wedge w) \vee (\bar{y} \wedge v \wedge w)$

A questo punto la clausola where viene ulteriormente semplificata eliminando completamente la negazione: la negazione di una condizione composta è stata trasformata utilizzando i teoremi (T1), (T1')e (T2) visti in precedenza, se invece l'operatore *not* precede una condizione semplice si sostituisce la condizione con la sua complementare.
Per fare ciò devono essere valutati sia l'operatore che il quantificatore presenti all'interno dell'espressione di confronto:

- sostituzione del quantificatore:

  some, any $\Leftrightarrow$ *all*

- sostituzione dell'operatore:

  $'>' \Leftrightarrow '<='$
  $'<' \Leftrightarrow '>='$
  $'=' \Leftrightarrow '!='$
  'like' $\Leftrightarrow$ 'not like'

Un esempio di normalizzazione di una clausola WHERE puó essere il seguente:
Data una classe S_1 con schema S che contiene quattro attributi A, B, C e D poniamo la seguente query:

```
Q:  select A
    from S
    where not((A = 'a1') or (C > 100))
    and    (B = 'b1' or D < 5)
```

La clausola where della query `Q` viene posta in forma normale disgiuntiva, generando `Q'`:

```
Q': select A
    from S
    where ((A != 'a1')
    and    (C <= 10000))
    or     (B = 'b1')
    and    (D < 5 ))
```

3.4.2 Regole di equivalenza dell'Algebra Relazionale

Come ogni algebra, anche in *Algebra Relazionale (AR)* si pone il problema di stabilire se due espressioni sono equivalenti, ovvero se producono lo stesso risultato per ogni stato (legale o meno) del data base. Questo vuol dire che generando le quary locali partendo dalla query globale otterremo delle query che abiano elementi in comune. Cioè una query locale deve essere equivalente ad una parte della query globale ed il risultato dell'esecuzione di questi deve essere lo stesso.
Nel caso generale di espressioni *non monotone* (che includono anche l'operatore di differenza) verificare l'equivalenza di due espressioni dell'AR è un problema

indicidibile, anche nel caso in cui non sono presenti operatori di selezione. Viceversa, se si escludesse l'operatore di differenza, il problema diventa NP-copleto. Un problema si dice NP-completo se si può risolvere in un tempo polinominale. Possiamo osservare che l'AR è, ad un certo livello di astrazione, un linguaggio procedurale. Dunque definita una metrica e data una espressione Q, si possono porre due problemi differenti: il primo di determinare un'espressione Q' equivalente a Q che sia "ottima" ed il secondo data anche l'espressione Q' determinare se è equivalente all'espresione Q. Il processo che porta alla risoluzine di questi problemi è detto **Ottimizzazione Algebrica**.
Se nel processo di trasformazione vengono presi in considerazione anche i vincoli che definiscono l'insieme degli stati legali del data base su cui viene effettuata l'interrogazione, allora si parla più propriamente di **Ottimizzazine Semantica**.

Si distingue generalmente tra:

equivalenza dipendente dallo schema del data base

$Q' \equiv_{schm} Q''$ se Q'(ist_db) = Q"(ist_db)
per ogni istanza ist_db dello schema (*schm*) del data base

equivalenza assoluta

$Q' \equiv Q''$ se $Q' \equiv_{schm} Q''$ per ogni schema *schm* del data base

Con riferimento all'equivalenza assoluta riportiamo di seguito le principali regole di equivalenza dell'algebra relazionale:

* **Commutatività e associatività di join, unione e prodotto cartesiano:**

$$E \bowtie E' \equiv E' \bowtie E$$
$$(E \bowtie E') \bowtie E'' \equiv E \bowtie (E' \bowtie E'')$$

dove E,E' ed E'' sono arbitrarie espressioni.

* **Raggruppamento di proiezioni:**

$$\pi_X(\pi_{XY}(E)) \equiv \pi_X(E)$$

* **Raggruppamento di selezioni:**

$$\sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2}(E)$$

* **Commutatività di selezione e proiezione:**

$$\pi_X(\sigma_F(E)) \equiv \sigma_F(\pi_X(E))$$

a condizione che il filtro F si riferisce solo ad attributi appartinenti ad X.

* **Distributività della selezione:**

$$\sigma_{F1 \wedge F2}(E_1 \times E_2) \equiv \sigma_{F1}(E_1) \times \sigma_{F2}(E_2)$$
$$\sigma_{F1 \wedge F2}(E_1 \bowtie E_2) \equiv \sigma_F 1(E_1) \bowtie \sigma_{F2}(E_2)$$
dove F_1 ed F_2 si riferiscono solo ad attributi di E_1 ed E_2 rispettivamente
$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2); \sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

* **Distributività della proiezione:**

$$\pi_{X_1X_2}(E_1[X_1Z] \times E_2[X_2W]) \equiv \pi_{X_1}(E_1[X_1Z]) \times \pi_{X_2}(E_2[X_2W])$$
$$\pi_{X_1X_2}(E_1[Y_1] \bowtie E_2[Y_2]) \equiv \pi_{X_1}(E_1[Y_1]) \bowtie \pi_{X_2}(E_2[Y_2])$$
con $X_1 \subseteq Y_1, X_2 \subseteq Y_2, Y_1 \cap Y_2 \subseteq X_1$ e $Y_1 \cap Y_2 \subseteq X_2$
$$\pi_X(E_1 \cup E_2) \equiv \pi_X(E_1) \cup \pi_X(E_2)$$

Nel nostro contesto assumiamo che queste regole sono valide ed applicabili in presenza della Full Disjunction. Ci permettiamo di fare questa ipotesi poichè come abbiamo definito nell'equazione 3.3 esiste un modo per riscrivere l'operatore di NOJ tramite l'operatori di join naturale e unione minima e per questi due operatori le regole esposte sopra sono sempre valide.

Come premessa alla sezione 3.4.5 possiamo dire che ogni espressione dell'algebra relazionale può essere rappresentata da un albero, che fornisce indicazioni sull'ordine di valutazione degli operatori. Ogni nodo, a eccezione delle foglie, rappresenta un operatore. Gli operatori unari hanno solo un ramo in uscita, mentre gli operatori binari hanno due rami in ingresso e uno solo in uscita. Le foglie rappresentano schemi relazionali o relazioni se si ragiona a livello di istanze.

3.4.3 Separabilità di una forma disgiuntiva

Un punto critico nella traduzione dei predicati è il mapping tra le diverse condizioni di selezione che non è semplice e molto spesso non si riescono a mappare condizioni singole. Certe condizioni sono inter-dipendenti e devono essere tradotte insieme. In generale si può dire che la traduzione è "molti-a-molti" e che senza rispettare le dipendenze tra i diversi vincoli non si riesce ad ottenere un mapping minimale che abbia un coefficiente selettivo massimale. Selezionare quale predicati tradurre e come farlo non è semplice e può coinvolgere tutta la query.

Secondo gli studi effettuati da Chen-Chuan K. Cheng e Héctor García-Molina ([41], [42]) per facilitare il trattamento di query complesse possiamo decomporre o suddividere la condizione di selezione della query iniziale, rappresentata tramite la sua forma booleana, in termini più semplici per trattarli separatamente. Per iniziare a chiarire questo concetto diamo la definizione di decomponibilità di un'espressione booleana in forma congiuntiva e la relativa notazione di scomposizione. In seguito specificheremo che esiste una sostanziale differenza nella possibilità di scomporre una condizione di selezione in relazione con la sua forma booleana.

Definizione 3.4.2 (Separabilità di una forma disgiuntiva) *Data un'espressione booleana in forma normale disgiuntiva* $\check{Q} = D_1 \vee \ldots \vee D_n$ *con* $n > 1$, *dove le* D_i *sono arbitrarie condizioni di selezione (o in altre parole sono termini disgiuntivi arbitrari) e una funzione di mapping* S *allora* $\check{Q}$ *è decomponibile se esiste un sotto-insieme proprio* $B_1, \ldots, B_m$ *di* $\{D_1, \ldots, D_n\}$, *per esempio* $B_j \subset \{D_1, \ldots, D_n\}$, *tale che* $S(\check{Q}) = S(\vee(B_1)) \ldots S(\vee(B_m))$. *In caso contrario l'espressione viene detta non decomponibile.*

Inoltre, se $\check{Q}$ *risultasse decomponibile per* $\{D_1\}, \ldots, \{D_n\}$ *risulterà che* $S(\check{Q}) = S(D_1) \vee \ldots \vee S(D_n)$, *allora* $\check{Q}$ *è anche separabile.*

Tale definizione afferma che se noi cerchiamo la migliore traduzione per la query globale Q, ci si può concentrare sulle "ottimizzazioni locali". In altre parole la ricerca della migliore traduzione per la query Q non sarà disturbata dal fatto che alcuni termini della sua condizione di selezione appaiono nella query trascritta in termini diversi.

Possiamo notare che in generale un'espressione booleana in forma normale congiuntiva non sempre può risultare separabile, mente la sua trasformazione in forma normale disgiuntiva lo è sempre. Siccome le disgiunzioni sono sempre separabili un approccio per la traduzione di una generica espressione booleana è quello di trasformala in DNF. Dopodichè una volta determinata la decomposizione della condizione di selezione della query Q possiamo trattare le sub-query separatamente. Cioè quando la query Q è decomponibile e separabile il suo mapping può essere la forma sintetizzata dei mapping delle sue sotto-query. Inoltre la separabilità è semplicemente un caso particolare di decomponibilità in cui l'elemento congiunto può essere separato individualmente.

Per esempio in riferimento alla Figura 3.4.3 poichè le condizioni su $t1$ e $anno$ sono indipendenti, $S(f_{t_1} \wedge f_y)$ = $S(f_{t_1}) \wedge S(f_y)$ e ($f_{t_1} \wedge f_y$) è decomponibile e in questo caso è anche separabile. D'altro canto non possiamo decomporre ($f_y \wedge f_m$) poichè $S(f_y \wedge f_m)$ è diverso da $S(f_y) \wedge S(f_m)$ per il fatto della interdipendenza dei termini. Inoltre possiammo notare che la query $f_{t_1} \wedge f_y \wedge f_m$ è comunque decomponibile perchè è valido che : $S(f_{t_1} \wedge f_y \wedge f_m)$ = $S(f_{t_1}) \wedge S(f_y \wedge f_m)$. Questo vuol dire che possiamo effettuare la partizione dell'insieme $\{f_{t_1}, f_y, f_m\}$

Query originale
$\check{Q}_1 = f_l \wedge f_{t1} \wedge f_y \wedge f_m \wedge f_k$
f_l: [In = "Rossi"]
f_{t1}: [t1 contiene "espressione"]
f_y: [anno = 1999]
f_m: [mese = 5]
f_k: [keyword contiene www]

Figura 3.2: Mapping di una query semplice congiuntiva.

nei propri sottoinsiemi $\{f_{t_1}\}$,$\{f_y, f_m\}$. Comunque la query non è separabile poichè l'insieme $\{f_y, f_m\}$ non è separabile.

3.4.4 Riscrittura rispetto ad una singola sorgente

Il punto di partenza per la traduzione della query è la sua forma normale disgiuntiva (DNF) che, come detto nella sezione precedente è sempre separabile e quindi possiamo effettuare il mapping per le diverse sorgenti separatamente.

A titolo di esempio consideriamo un predicato atomico e discutiamo della sua riscrittura rispetto ad una singola sorgente. Siano dati il predicato (A like "$P\star$") e la sorgente S che non supporti l'operazione di "like". Invece di non considerare tale predicato e rischiare di perdere informazioni che possono risultare utili possiamo cercare di tradurre tale predicato al meglio, ad esempio supponendo che la sorgente S fornisca l'operatore "contains" il predicato in esame può essere riscritto cosi: (A contains "P").

Definizione 3.4.3 (Mapping minimo) *Data un'espressione booleana in forma normale disgiuntiva $\check{Q}$ e data una funzione M che mappa gli attributi globali appartenenti a $\check{Q}$ negli attributi della sorgente locale S, allora diciamo che $M_S(\check{Q})$ è il mapping minimo se sono soddisfatte le seguenti condizioni:*

(1) $M_S(\check{Q})$ è esprimibile nella semantica della sorgente S;

(2) $M_S(\check{Q})$ sussume la $\check{Q}$;

(3) $M_S(\check{Q})$ è minima, cioè non esiste un altra espressione $\check{Q}'$ tale per cui sono soddisfatte (1) e (2) e $M_S(\check{Q})$ sussume propriamente la $\check{Q}'$.

Nel nostro contesto vogliamo prescindere da questo problema e faremo quindi l'ipotesi che ogni operatore, e quindi ogni predicato atomico, presente nella query globale, è *pienamente supportato* anche a livello locale.

Pertanto, dato un predicato atomico su un attributo A (sugli attributi A e B), tale predicato è supportato dalla sorgente S se l'attributo A (gli attributi A e B) ha un mapping non nullo. Dobbiamo osservare che la riscrittura delle interrogazioni viene effettuata in funzione dei singoli schemi locali. Ciò è reso possibile dall'impiego del modello comune dei dati grazie al quale il amediatore comunica con i Wrapper posti sulle singole sorgenti. Saranno infatti quasti ultimi moduli a tradurre ogni interrogazione formulata in linguaggi OQL$_{I^3}$ nella sintassi opportuna.

3.4.5 Individuazione dei predicati residui e generazione del Filtro Globale

Illustriamo il metodo con alcuni esempi. Sia G una classe globale, con lo schema $S(G) = (ABC)$, corrispondente alle classi loclai L_1 e L_2 con la seguente Mapping Table:

	A	**B**	**C**
L_1	A_1		C_1
L_2		B_2	C_2

Consideriamo una Global Query $\sigma_Q(FD(L_1, L_2))$ con la seguente condizione di selezione espressa in forma normale disgiuntiva (DNF):

$$Q = (A = 3 \wedge B = 7) \vee (C = 9)$$

Supponiamo che la Join Map sia già definito e che l'albero in DNF che rappresenta la query è disponibile. Nel nostro caso l'attributo sul quale eseguiamo il join è **C**, cioè riferendosi alla notazione introdotta in 3.1 $o[L_1].C$ = $o[L_2].C$ e l'albero che rappresenta la query globale in DNF (chiamato Master) è rappresentato in figura 3.3.

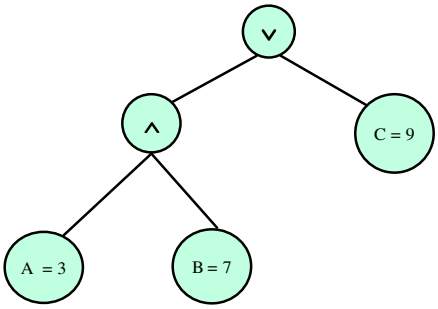

Figura 3.3: Albero Master in DNF.

Inoltre per poter chiarire meglio il concetto consideriamo anche delle istanze per le classi L_1 e L_2 che soddisfano la proprietà di omogeneità semantica, rappresentate tramite le seguenti tabelle:

Classe L_1

A_1	C_1
3	7
4	9

Classe L_2

B_2	C_2
5	7
6	9

A questo punto abbiamo a disposizione tutte le informazioni necessarie per la determinazione del Filtro Globale che è costituito dai predicati residui che andiamo a determinare tramite il seguente procedimento:

Passo 1 **(Individuazione dei predicati residui)**:
Si cerca di mappare l'albero Master su ogni classe locale coinvolta nell'interrogazione generando in questo modo il corrispondente sotto-albero in DNF e determinando quali dei predicati, che fanno parte della Query Globale, si riescono a tradurre sullo schema di questa classe utilizzando il metodo esposto nella sezione precedente. Se in una foglia c'è un predicato che non si riesce a tradurre, si setta tale predicato a "true" e nell'albero Master nella posizione che corrisponde a questo predicato si mette un flag. In questo modo proiettando l'albero Master su tutte le classi locali vengono individuati i sotto-alberi corrispondenti in cui i predicati non tradotti sono settati a "true". La trasformazione di questi predicati in "true" è necessaria per essere sicuri di non perdere nessun dato significativo nella risposta che ritornerà dalla sergente locale. Nel nostro caso le proiezioni dell'albero Master sulle classi locali L_1 e L_2 sono rappresentate in Fig. 3.4 e Fig. 3.5.

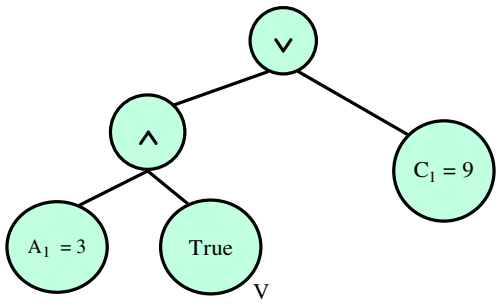

Figura 3.4: Albero in DNF per la classe L_1.

Si esegue questo procedimento per tutte le classi coinvolte. Alla fine di questo procedimento nella struttura che rappresenta l'albero Master abbiamo tutta l'informazione per poter generare il Filtro Globale. Riferendosi al nostro esempio

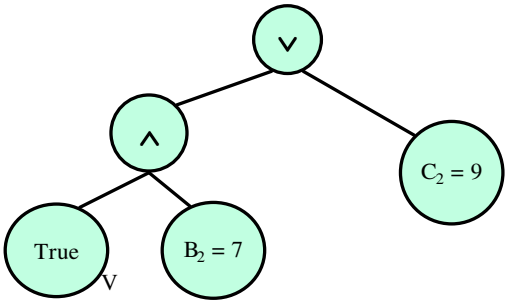

Figura 3.5: Albero in DNF per la classe L_2.

l'albero Master in DNF risulta essere quello rappresentato in figura 3.6.

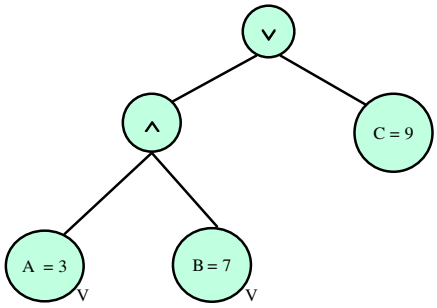

Figura 3.6: Albero risultante in DNF.

Passo 2 **(Generazione del Filtro Globale)**:
Si trascrive la query globale in CNF. La necessità di questa trasformazione deriva dal fatto che noi dobbiamo trovare tutte le condizioni necessarie che la query Q deve soddisfare. Dunque il filtro deve essere composto dalle congiunzioni di tutte le condizioni necessarie che non riusciamo a presentare a livello locale. Un modo per trovare tali condizioni necessarie è la trasformazione della query in CNF: $Q = \Pi_{i=1}^{m} D_i$, dove le $D_i = \sum_{i=1}^{n} P_j$ sono i termini disgiuntivi di qualche predicato definito in Q. Dato che le D_i sono congiunti in Q essi rappresentano le condizioni necessarie che tale query Q deve soddisfare. Ogni D_i che contiene predicati non supportati dalla corrispondente sorgente L_i rappresenter la condizione necessaria che non può essere espressa nella query locale corrispondente e dunque far parte del predicato di selezione residua (Filtro Globale). Nel nostro caso la query trascritta in CNF risulta:

$$(A = 3 \vee C = 9) \wedge (B = 7 \vee C = 9)$$

Dopo questa trasformazione si genera il corrispondente albero Master in CNF con i seguenti criteri:

- Per ogni foglia contrassegnata con il flag si contrassegnano ricorsivamente tutti i nodi che sono nel cammino che lega la foglia alla radice.
- Il predicato di selezione residua è rappresentato dalla parte dell'albero, nella forma CNF, contrassegnata con i flag.

Alla fine del procedimento la parte dell'albero Master contrassegnata rappresenta il Filtro Globale. Dunque ritornando al nostro esempio l'albero in CNF risulta essere quello di Fig.3.7 ed il predicato residuo risulta essere tutta la query. Si può notare che le tuple estrapolate dalle due classi L_1 e L_2 sono:(4,9) e (6,9) rispettivamente, mentre la tupla risultante è (4,6,9).
Da ogni sotto albero si possono generare le condizioni di selezione delle singole

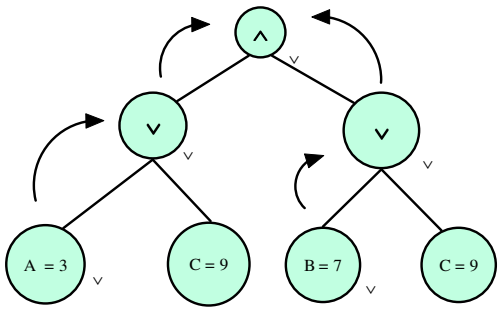

Figura 3.7: Albero Master in CNF.

query locali. Il procedimento è semplice e si basa sull'esplorazione ricorsiva del corrispondente albero binario. Per esempio riferendosi alla Figura 3.4 partendo dalla radice si usa il metodo *depth firs* con le seguenti regole di trasformazione per ogni nodo:

- $p \vee true = true$;
- $p \wedge true = p$;
- $p_1 \vee p_2$ oppure $p_1 \wedge p_2$ si mantiene tutto il predicato.

Quindi, riassumendo, la query:

$$\sigma_Q(FD(L_1, L_2))$$

dove $Q = (A = 3 \wedge B = 7) \vee (C = 9)$ è stata riscritta come:

$$\sigma_{F_G}(FD(\sigma^{\mathcal{G}}_{F_1}(L_1), \sigma^{\mathcal{G}}_{F_2}(L_2)))$$

dove
$F_G \equiv Q = (A = 3 \wedge B = 7) \vee (C = 9)$
$F_1 = (C_1 = 9 \vee A_1 = 3)$
$F_2 = (C_2 = 9 \vee B_2 = 7)$
Osserviamo che l'espressione $\sigma^{\mathcal{G}}_{F_i}(S_i)$ definisce la condizione "where" della query locale per la classe locale L_i; la select list di tale query si ottiene dalla select list dela query globale aggiungendo le condizioni di selezione degli attributi di join e gli attributi di F_G.

Introduciamo un'altro esempio considerando ancora la query dell'esempio precedente, ma questa volta effetuiamo una modifica prendendo il mapping tra gli attributi locali e quelli globali nel seguente modo:

	A	**B**	**C**
L_1	A_1	B_1	C_1
L_2		B_2	C_2

La condizione di join è sempre su **C**, cioè $o[L_1].C = o[L_2].C$. Le tuple che le classi locali contengono risultano come segue:

Classe L_1 **Classe** L_2

A_1	B_1	C_1	B_2	C_2
4	6	9	7	7
3	7	7	6	9

Si nota che per l'omogeneità semantica anche i valori degli attributi B_1 e B_2 risultano uguali. Per semplicità consideriamo la stessa query e dunque l'albero Master risulta essere sempre quello di Fig.3.3.
Si applica il secondo passo del procedimento e vengono generati i sotto-alberi in DNF per le due classi L_1 e L_2 che vengono rappresentati in Fig.3.8 e Fig.3.9.
Si nota che nella classe locale L_1 riusciamo a spingere tutta la query, mentre nella classe locale L_2 possiamo spedire solo i predicati che corrispondono agli attributi **B** e **C**.
L'albero Master corrispondente alla query in esame rappresentato nella sua forma normale disgiuntiva risulta essere quello di Fig.3.10. Facendo riferimento a tale figura possiamo notare che l'unico predicato contrassegnato è quello che non siamo riusciti spingere in nessuna delle classi locali coinvolte nell'interrogazione.

A questo punto basta effettuare la trasformazione dell'albero in forma normale congiuntiva (rappresentato in Fig.3.11) ed applicando il procedimento descritto nell'esempio precedente otterremo il predicato residuo, che risulta essere il

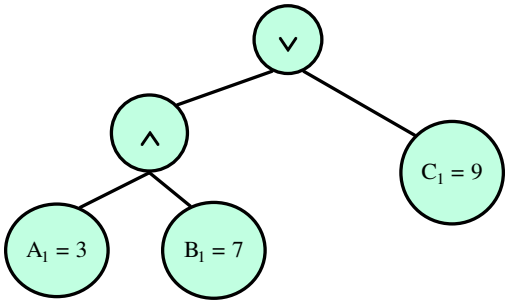

Figura 3.8: Albero in DNF per la classe L_1.

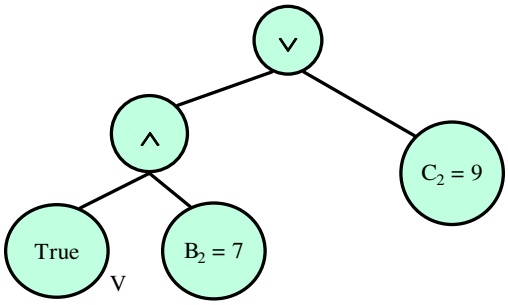

Figura 3.9: Albero in DNF per la classe L_2.

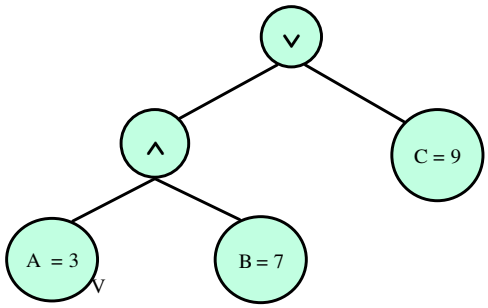

Figura 3.10: Albero in DNF.

seguente: $(A = 3 \vee C = 9)$. Invece le tuple estrapolate dalla prima classe, a cui si riesce a spedire tutta la query, sono:(4,6,9) e (3,7,7), mentre quelle estrapolate dalla seconda classe risultano essere (6,9) e (7,7). Dopo l'esecuzione del join finale e l'applicazione del filtro globale risulta che le tuple che fanno parte del risultato globale sono: (4,6,9) e (3,7,7). Dunque riprendendo la nostra simbologia otteniamo:

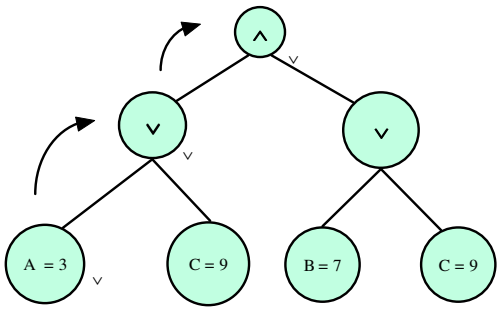

Figura 3.11: Albero in CNF.

$$\sigma_{F_G}(FD(\sigma^{\mathcal{G}}_{F_1}(L_1), \sigma^{\mathcal{G}}_{F_2}(L_2)))$$

con
$F_G = (A = 3 \vee C = 9)$
$F_1 = (A_1 = 3 \wedge B_1 = 7) \vee (C_1 = 9)$
$F_2 = (B_2 = 3 \vee C_2 = 9)$.

Riassumendo, il procedimento per l'individuazione dei predicati residui e la generazione del Filtro Globale è il seguente:

- Generazione dei sotto-alberi corrispondenti alle Local Query;
- Individuazione dei predicati non mappati nella classe locale corrispondente;
- Confronto tra l'albero Master, che rappresenta la Global Query, ed i sotto-alberi che rappresentano le Local Query, contrassegnando ogni predicato non mappato in nessuna delle classi locali;
- Trasformazione dell'albero Master in CNF;
- La parte dell'albero contrasegnata rappresenta il Filtro Globale ed i predicati che fanno parte di esso sono i predicati residui;

Dopo la determinazione del Filtro Globale la Global Query deve essere trascritta per ogni sorgente locale inerente all'interrogazione. Traduciamo tale procedimento negli algoritmi rappresentati nelle Figure 3.1 e 3.2.

In definitiva gli algoritmi presentati consentono di definire l'espressione 3.7 e quindi costruiscono il nostro metodo di riscrittura.
A questo punto considerando ogni singola sorgente si deve eseguire

$$\sigma^{G}_{S_i(F)}(l_i).$$

Generazione dei Filtri Locali

Input: * Q: query in DNF definita sullo schema globale;
* L: classe locale su cui tradurre i predicati della query Q
Output: * A_L: albero "contrasegnato" per la classe L.
* F: filtro in DNF relativo ad L.
Procedimento:
- Genera l'albero Master A corrispondente alla Q.
- Per ogni foglia f di A
  se il predicato in f non è supportato da L
  allora
  contrassegna f
- Restituisci l'albero A_L modificato.
- Restituisci F ottenuto ponendo a true tutti i predicati di foglie contrassegnate.

Tabella 3.1: Algoritmo di generazione dei Filtri Locali (ALF)

Algoritmo di generazione del Filtro Globale

Input: * Q: una query in DNF definita sullo schema globale.
* G: classe globale con l'insieme delle classi locali $\mathcal{L}$ corrispondenti.
Output: Filtro Globale
Procedimento:
- Genera l'albero Master A relativo a Q.
- per ogni classe locale L
  - applica l'algoritmo ALF e contrassegna A.
- Trasforma l'albero A cosi ottenuto in CNF.
- Costruisci il Filtro Globale come:
  $\bigwedge_i f_{i,1} \vee f_{i,2} \vee \ldots \vee f_{i,k}$
dove almeno un $f_{i,j}$ è contrassegnato.

Tabella 3.2: Algoritmo di traduzione

E dunque per ogni sorgente l'interrogazione SQL-like inviata al relativo Wrapper sarà: Select $< select - list_i >$
From L_i
Where $S_i(F)$
dove $< select\_list_i >$ è costituita da tre componenti:

1) attributi della select list originale.

2) attributi contenuti nel Filtro Globale.

3) attributi coinvolti nella Join Table relativa ad L_i.

Il risultato della singola interrogazione locale deve essere a questo punto, in base all'equazione 3.7 elaborato tramite la Full Disjunction e questo sarà l'argomento del prossimo capitolo.

Capitolo 4

Algoritmo per il calcolo della Full Disjunction

Il nostro intento è quello di calcolare la Full Disjunction usando gli operatori classici dell'Algebra Relazionale e quindi utilizzando il linguaggio SQL. Vogliamo individuare una sequenza di natural outer join che calcoli la Full Disjunction in tutti casi che soddisfano le nostre ipotesi. Come abbiamo detto nel capitolo precedente il concetto della Full Disjunction è stato già affrontato in letteratura.
In [38], utilizzando il concetto di ipergrafo γ-aciclico, è stato dimostrato che una sequenza di natural outer join (NOJ) che produce la Full Disjunction esiste se e solo se lo schema relazionale su cui viene clacolato forma un ipergrafo connesso e γ-aciclico. In praticolare in tale sequenza di NOJ ogni relazione compare esattamente una sola volta. Questi risultati saranno presentati nella sezione 4.1. Per noi questa condizione restrittiva sulla γ-aciclicità è impraticabile in quanto nel nostro caso il grafo corrispondente alla classe globale, che rappresenta l'integrazione delle classi locali, risulta generalmente completamente connesso e quindi ciclico.

Pertanto il nostro scopo è quello di individuare un metodo per calcolare la Full Disjunction tramite NOJ che funzioni in tutti i casi ed in particolare anche nel caso di un grafo ciclico. Verrà presentato un algoritmo che restituisce le operazioni di NOJ da effettuare per calcolare la FD: chiameremo tale risultato "pseudo sequenza" di NOJ in quanto, in particolare, una stessa relazione può essere considerata più volte nelle operazioni di NOJ. Sia nei casi in cui il grafo risultasse aciclico sia in quelli in cui risultasse ciclico saranno applicati algoritmi di ottimizzazione per ridurre al massimo i passi dell'algoritmo di generazione della pseudo-sequenza di outer join (PSOJ).

4.1 Calcolo della Full Disjunction tramite outer-join: caso aciclico

In [38] A. Rajaraman e J. D. Ullman hanno stato proposto un algoritmo per il calcolo della Full Disjunction tramite l'operatore di natural outer join (NOJ) nel caso in cui il grafo che rappresenta la condizione di selezione della query in esame risultasse "aciclico". Per dimostrare che tale algoritmo non è applicabile al nostro contesto nel seguito daremo la definizione formale di ciclicità e verificheremo che nel nostro caso generalmente siamo in presenza di ciclicità.

Un ipergrafo è una coppia $(\mathcal{N}, \mathcal{E})$, dove $\mathcal{N}$ è un insieme finito di nodi ed $\mathcal{E}$ è un insieme di archi che include un sottoinsieme arbitrario non vuoto di $\mathcal{N}$. Si dice che un grafo è *ordinario non orientato* se ogni suo arco contiene esattamente due nodi. Identificheremo un ipergrafo solamente menzionando i suoi archi e assumiamo che i nodi sono quelli che compaiono nei suoi tagli. Uno schema di database ovvero una collezione di schemi relazionali $\{R_1, \ldots, R_n\}$ può essere rappresentato tramite un ipergrafo, dove $\mathcal{E} = \bigcup_{i=1}^{n} schm(R_i)$. L'ipergrafo corrispondente allo schema del database R=$\{R_1, \ldots, R_n\}$ ha come nodi quegli attributi che appaiono in una o più R_i e come archi le relazioni stesse. Spesso parleremo di ipergrafo R senza menzionare il suo set di nodi $\mathcal{N}$, che comunque intenderemo come $\mathcal{N}$ = $\bigcup\{R_i : 1 \leq i \leq n\}$. Ad esempio l'ipergrafo di Figura 4.2 corrisponde agli schemi del database di Figura 4.1.

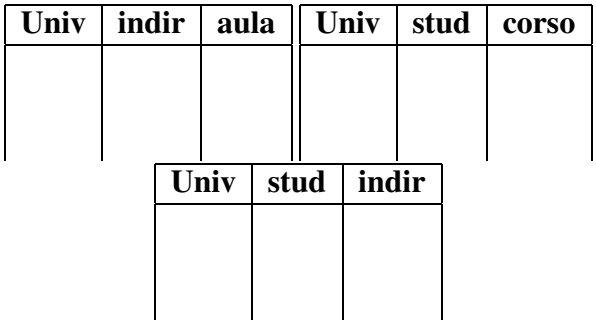

Figura 4.1: Schema di un database

Consideriamo due schemi di databse rappresentati nelle Figure 4.3 e 4.4. Si può notare che ognuna rappresenta lo schema relazionale di un database e che l'unica differenza tra di loro e che il secondo database ha un attributo D nell'ultima relazione mentre il primo database ha un attributo E. Questa piccola differenza nello schema determina però che il primo schema è aciclico mentre il secondo schema risulta essere ciclico

Come si può intuire considerando la rappresentazione degli schemi tramite

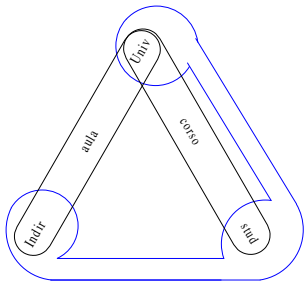

Figura 4.2: Grafo di un database

A	B	C	B	C	D	A	E

Figura 4.3:

A	B	C	B	C	D	A	D

Figura 4.4:

ipergrafi in Figura 4.5. L'ipergrafo in figura 4.5.a) è chiaramente aciclico poichè si può notare che non esiste un "cammino" che partendo da un nodo porti allo stesso nodo. Il secondo ipergrafo invece rappresenta un cilco perchè esiste il cammino A-D-C-B-A. Molti delle proprietà di base desiderabili degli schemi relazionali dei database risultano essere equivalenti alle proprietà di acilcicità. Queste proprietà sono state definite e studiate da un grande numero di ricercatori in molti diversi contesti. In questa sezione ci riferiremo ai lavori di Fagin e Galindo-Legaria.

Definizione 4.1.1 (Cammino.) *Si chiama cammino tra due nodi s e t la sequenza di $k \geq 1$ archi $E_1, \ldots, E_k$ tali che:*

1. $s \in E_1$
2. $t \in E_k$
3. $E_i \bigcap E_{i+1} \neq \emptyset$ *per* $1 \leq i < k$

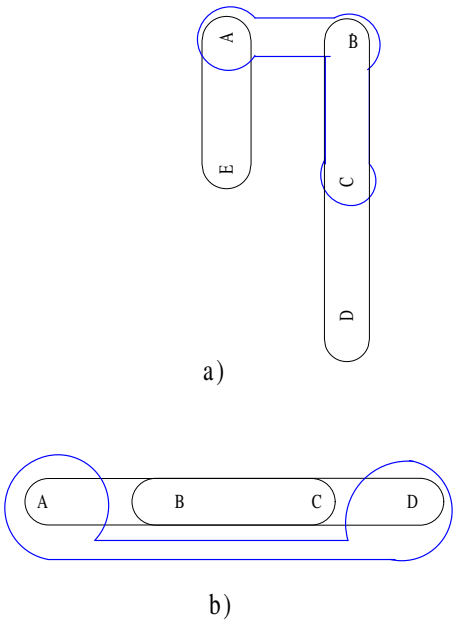

Figura 4.5:

Definizione 4.1.2 (Grafo connesso.) *Un grafo si dice connesso se presi due qualsiasi nodi che appartengono all'insieme $\mathcal{N}$, esiste un cammino di archi che appartengono ad $\mathcal{E}$ che gli collega.*

Gli ipergrafi si possono suddividere in due grandi classi: ipergrafi ciclici ed ipergrafi aciclici.

Uno schema di un database è aciclico o ciclico se l'ipergrafo corrispondente lo è. Sono stati effettuati diversi studi [43], che affermano che i database aciclici godono delle proprietà particolari e il problema per la determinazione della aciclicità di uno schema è NP-completo, ma esiste sempre un algoritmo che determina l'aciclicità di un ipergrafo, corrispondente ad uno schema, in un tempo polinominale.
Noi vogliamo dimostrare che se presa una query come espressione booleana in CNF e rappresentata tramite un grafo, possiamo ottenere il risultato desiderato applicando il nostro algoritmo PSOJ. Di seguito dimostreremo che tale l'algoritmo può essere applicato ad un qualsiasi grafo connesso. Nella letteratura esistono diverse definizioni di equivalenza tra ipergrafi e schemi di database cosi come per la definizione di ciclicità dei grafi. Noi faremo riferimento alle definizioni date da Fagin [40] e Ulman. Fagin introduce un nuovo concetto di aciclicità secondo il quale un ipergrafo può essere aciclico (denotato come α-aciclico) pur avendo al suo interno alcuni sotto-grafi ciclici. Questo fenomeno non infulenza il nostro al-

goritmo e dunque non ci soffermeremo sulle sue specifiche. Assumeremo inoltre che abbiamo a che fare con grafi connessi.

[44, 40] dimostrano che esiste una analogia tra le affermazioni di aciclicità per lo schema di un database e la forma normale corrispondente allo schema relazionale. In più è dimostrato che ogni forma normale è più restrittiva di quella che la precede. Dunque noi ci possiamo concentrare sulla ciclicità (aciclicità) di uno schema ignorando, almeno per adesso, la forma normale in cui si trova.

α-aciclicità

Sia $(\mathcal{N}, \mathcal{E})$ un ipergrafo, la sua riduzione $(\mathcal{N}, \mathcal{E}')$ si ottiene rimouvendo da $\mathcal{E}$ ogni arco che sia un sottoinsieme proprio di un altro arco. Un ipergrafo si chiama ridotto se dopo la sua riduzione nessun arco è propriamente contenuto in un altro arco dell'ipergrafo.
Sia $\mathcal{M}$ un insieme di nodi di $(\mathcal{N}, \mathcal{E})$. Gli archi parziali generati a partire da $\mathcal{M}$ possono essere ottenuti intersecando gli archi di $\mathcal{E}$ con quelli di $\mathcal{M}$, cioè $\{E \bigcap M : E \in \mathcal{E}\}$-$\{\emptyset\}$.
Sia $\mathcal{F}$ un sottografo connesso generato a partire da un insieme di archi. Siano inoltre $(E, F) \in \mathcal{F}$ e Q = $E \bigcap F$. Chiameremo Q un insieme articolato di $\mathcal{F}$ se il risultato della rimozione di Q da tutti gli archi di $\mathcal{F}$ non è un insieme connesso di archi parziali(è un insieme non connesso di archi parziali). Chiameremo un insieme triviale se contiene almeno due membri. Un blocco di un ipergrafo ridotto è un insieme di archi parziali connessi, generati a partire da un insieme di nodi, che non contiene insiemi articolati. Si dice che un ipergraffo è α-aciclico se tutti i suoi blocchi sono triviali, altrimenti si chiama α-ciclico.
Se $(\mathcal{N}, \mathcal{E})$ è un ipergrafo, $\mathcal{F} \subset \mathcal{E}$ e $\mathcal{M}$ = $\bigcup_i N_i : N_i \in \mathcal{F}$ diremo che $\mathcal{F}$ è un sottoinsieme chiuso se per ogni arco E dell'ipergrafo esiste un arco $F \in \mathcal{F}$ tale che $E \bigcap M \subseteq F$.
Per esempio se consideriamo il grafo rappresentato in Figura **??** possiamo osservare che esso è α-aciclico.

In riferimento con tale figura un insieme articolato di tagli è rappresentato da $ABC \bigcap ACE = AC$, cosicchè se rimuoviamo dall'insieme dei tagli del grafo l'insieme articolato otteniamo $\{B, DE, EF, E\}$ che non sono completamente connessi tra di loro. Notiamo inoltre che l'insieme di tagli $\{ABC, CDE, EFA\}$ è anch'esso articolato, ciò nonostante questo insieme non è generato a partire da un solo nodo e dunque non è triviale. Questo è in contraddizione con la definizione di α-ciclicità data qui di seguito.

Definizione 4.1.3 (.) *Un ipergrafo $\mathcal{H}$ = $(\mathcal{N}, \mathcal{E})$ è α-aciclico se e solo se ogni*

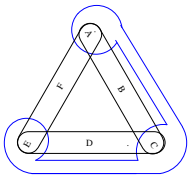

Figura 4.6: Ipergrafo α-aciclico.

sottoinsieme di archi non triviale, connesso e chiuso ha un sottoinsieme articolato.

Berge-aciclicità

Definizione 4.1.4 (Ciclo di Berge.) *Un ciclo di Berge in un ipergrafo è una sequenza* $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ *tale che:*

1. $x_1, \ldots, x_m$ *sono nodi distinti che appartengono a* $\mathcal{H} = (\mathcal{N}, \mathcal{E})$
2. $S_1, \ldots, S_{m+1}$ *sono archi di* $\mathcal{H}$*, con* $S_1 = S_{m+1}$
3. $m \geq 2$
4. *per* $x_i \in S_i$ *e* $x_i \in S_{i+1}$ *segue che* $x_i \in S_i \bigcap S_{i+1}$ *per* $1 \leq i \leq m$

Un grafo è ciclico secondo Berge(B-ciclico) se contiene almeno un ciclo di Berge altrimenti è detto B-aciclico.

β-aciclicità

In questo paragrafo daremo alcune definizioni per la β-ciclicità e affermeremo che queste sono equivalenti. La dimostrazione di ciò è stata data da [40].
Possiamo osservare che un grafo è β-aciclico se e solo se ogni suo sotto grafo è α-aciclico.

Definizione 4.1.5 (β-ciclo.) *Dato un ipergrafo* $\mathcal{H} = \mathcal{N}, \mathcal{E}$ *ed una sequenza* $(S_1, \ldots, S_m, S_{m+1})$ *di archi tali che se* $X = S_1 \bigcap \ldots \bigcap S_m$ *ed* S'_i *è la differenza di* S_i *- X per* $1 \leq i \leq m$*, allora* $(S'_1, \ldots, S'_m, S'_{m+1})$ *è un ciclo puro.*

Per esempio in riferimento con la Figura 4.6 nel paragrafo in cui abbiamo definito la α-ciclicità abbiamo detto che tale grafo è α-aciclico. Alla luce della definizione data qui sopra il grafo in tale figura non risulta β-aciclico ma ciclico perchè il sotto-grafo costituito dai tagli $\{ABC, CDE, EFA\}$ è α-ciclico. In Figura 4.7 invece è rappresentato un grafo che contiene un ciclo puro.

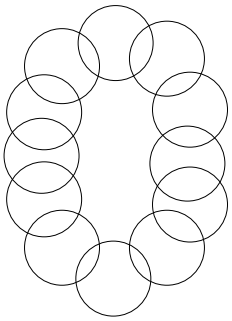

Figura 4.7: Ipergrafo di un ciclo puro.

A questo punto siamo in grado di dare le seguenti definizioni equivalenti:

Definizione 4.1.6 (β-ciclicità) .

1. *Un ipergrafo è β-ciclico se ha un β-ciclo.*
2. *Un ipergrafo è β-ciclico se qualche suo sottografo è $alpha$-ciclico.*
3. *Un ipergrafo è β-ciclico se qualche insieme, appartinente all'ipergrafo, di archi non triviale, connesso e ridotto ha un insieme articolato.*
4. *(alla Berge). Un ipergrafo è β-ciclico se:*
   - *$x_1, \ldots, x_m$ sono nodi distinti che appartengono a $\mathcal{H} = (\mathcal{N}, \mathcal{E})$.*
   - *$S_1, \ldots, S_{m+1}$ sono archi di $\mathcal{H}$, con $S_1 = S_{m+1}$.*
   - *$m \geq 3$.*
   - *$x_i \in S_i$ e $x_i \in S_{i+1}$ segue che $x_i \in S_i \bigcap S_{i+1}$ per $1 \leq i \leq m$.*
5. *Un ipergrafo è β-ciclico se ha un ciclo secondo Graham[26].*

Un esempio di β-ciclo è dato in Figura 4.8.

γ-aciclicità

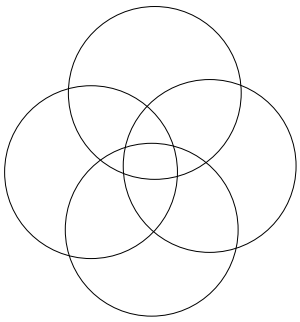

Figura 4.8: Ipergrafo di un β-ciclo.

Definizione 4.1.7 (γ-ciclo) . *Un γ-ciclo è una sequenza (S_1, x_1, S_2, x_2,- $\ldots, S_m, x_m, S_{m+1}$) tale che:*

- $x_1, \ldots, x_m$ *sono nodi distinti che appartengono a* $\mathcal{H} = (\mathcal{N}, \mathcal{E})$.
- $S_1, \ldots, S_m$ *sono archi di* $\mathcal{H}$, *con* $S_1 = S_{m+1}$.
- $m \geq 3$.
- $x_i \in S_i$ *e* $x_i \in S_{i+1}$ *segue che* $x_i \in S_i \bigcap S_{i+1}$ *per* $1 \leq i \leq m$.
- *se* $1 \leq i < m$, *allora* x_i *non appartiene ad* S_j *eccetto* S_i *e* s_{i+1}

Definizione 4.1.8 (γ-ciclicità) .

1. *Un ipergrafo è γ-ciclico se ha un γ-ciclo.*
2. *Un ipergrafo è γ-ciclico se ha un B-ciclo.*
3. *Un ipergrafo è γ-ciclico se ha un 3-γ-ciclo o un ciclo puro.*
4. *Un ipergrafo è γ-ciclico se dati due archi E ed F non disgiunti, rimuovendo $\{E \bigcap F\}$ da ogni arco di $\mathcal{H}$ si ottiene sempre un grafo connesso, cioè ciò che era connesso ad E rimane connesso a ciò che era connesso ad F.*

Anche queste definizioni sono equivalenti [40]. Le relazioni tra i diversi tipi di ciclicità sono i seguenti:
α-ciclico $\Rightarrow$ β-ciclico $\Rightarrow$ γ-ciclico $\Rightarrow$ B-ciclico;
B-aciclico $\Rightarrow$ γ-aciclico $\Rightarrow$ β-aciclico $\Rightarrow$ α-aciclico.

Data una query che contiene dei predicati di join e outer join possiamo dire che la sua condizione di selezione può essere rappresentata tramite una relazione sullo schema globale. Il nostro approccio è quello di dividere la query per ogni

sorgente locale coinvolta nell'interrogazione e risolvere le sotto-query successivamente generate sugli schemi locali. Questo ci dice che lo schema globale può essere diviso in n elementi rappresentati da n relazioni $\{R_1 \dots R_n\}$ con schemi schm(R_1), ..., schm(R_n). Allora il grafo corrispondente presenterà dei nodi per ogni relazione nominata nella query e degli archi per ogni termine disgiuntivo dei predicati di join o outer join. Un predicato P forma un'arco fra i nodi di due relazioni R_i ed R_j se P referenzia degli attributi di R_i ed R_j. Più precisamente schm(p) $\subsetneq$ schm(R_i) e schm(p) $\subsetneq$ schm(R_j), ma schm(p) $\subseteq\big($ (schm(R_i) cup schm(R_j))$\big)$. Se siamo in presenza di predicati congiuntivi ed essi riferiscono più di due relazioni, allora si parla di ipergrafi corrispondenti alla query. Il nostro algoritmo viene applicato al caso di predicati di join che referenziano due relazioni alla volta, ma può essere facilmente esteso al caso multidimensionale.
Forti delle definizioni e concetti introdotti in questo capitolo possiamo adesso presentare l'algoritmo di generazione della Full Disjunction proposto da Galindo-Legaria in [39]. Tale algoritmo, applicabile soltanto nel caso in cui il grafo corrispondente alla query sia γ-aciclico, la definizione di OuterUnion (3.3.1) e le formule di trascrizione 3.4, 3.5 e 3.6 definite nella sezione 3.3. Notiamo che l'operatore di unione minima ($\oplus$) tra due relazioni R_1 ed R_2 è definito cosi:

$$R_1 \oplus R_2 := (R_1 \biguplus R_2) \downarrow$$

cioè rimuovendo dal risultato dell'OuterUnion tutte le tuple sussunte. Questo procedimento si applica alla query in modo ricorsivo finchè non vengono riscritti tutti gli operatori di outer join tramite gli operatori di OuterUnion e join naturale. In [38] A. Rajarman e J. D. Ullman danno una rappresentazione più formale di come generare la sequenza di outer join per calcolare la Full Disjunction. Loro chiamano l'algoritmo SOJO (Sound Outer Join Ordering) che per completezza presentiamo in Figura 4.1.

Come abbiamo detto nelle defiizioni preliminari tale procedimento non è applicabile nel nostro contesto per causa dell'ipotesi di aciclicità del grafo corrispondente alla query in esame. Di seguito presenteremo il metodo di calcolo della Full Disjunction che è applicabile per qualsiasi grafo sia aciclico che ciclico.

4.2 Calcolo della Full Disjunction tramite outerjoin: caso ciclico

In questa sezione supporremo che alcune strutture che rappresentano delle informazioni ottenute durante le fasi di integrazione intensionale siano già presenti e focalizzeremo la nostra attenzione sull'algoritmo di generazione della Full Disjunction tramite una pseudo-sequenza di outer join (PSOJ). A questo proposito

Algoritmo SOJO di calcolo della Full Disjunction

Input: * Ipergrafo $\mathcal{H}$ connesso, γ-aciclico
Output: * Sequenza valida di outer join per $\mathcal{H}$
Procedimento: Cominciamo generando il diagramma di Bachman per $\mathcal{H}$.
Ricorsivamente partizioniamo $\mathcal{H}$ e BD($\mathcal{H}$) selezionando i nodi di costo minimo.
1) Definiamo il caso base: un ipertaglio rappresenta una relazione
e chiaramente definisce la Full Disjunction dei nodi che comprende.
2) trovare una decomposizione del ipergrafo $\mathcal{H}$ che contiene almeno
due ipertagli: $\mathcal{H}_1$ e $\mathcal{H}_2$
3) partizionare il diagramma di Bachman BD($\mathcal{H}$) in due parti che
corrispondono agli ipertagli determinati al passo 2)
4) ricorsivamente trovare due espressioni $espr_1$ e $espr_2$ che danno una sequenza
valida di outer join per $\mathcal{H}_\infty$ e $\mathcal{H}_\in$ rispettivamente usando il metodo di
riscrittura di Galindo-Legaria.
5) generare la $espr_1 \bowtie espr_2$.

Tabella 4.1: Algoritmo di traduzione SOJO

notiamo che una volta ottenuti i risultati delle query locali essi devono essere combinati in modo opportuno per ottenere il risultato desiderato. Man mano che i risultati delle query locali provengono dalle sorgenti locali essi vengono memorizzati dal DBMS in speciali tabelle temporanee e resi disponibili al Query Manager per la prossima elaborazione.

Algoritmo per la generazione della pseudo-sequenza di join (PSOJ)

Per illustrare il problema, consideriamo una classe globale G rappresentativa di quattro classi locali (relazionali) R_1, R_2, R_3, R_4 con i seguenti schemi: R_1 = (AEF), R_2 = (ABC), R_3 = (CDE) ed R_4 = (ACE). Per completezza consideriamo anche le istanze delle classi locali:

A	F	E
1	1	3
4	2	6
10	3	12
18	4	20

A	B	C
1	2	7
4	5	1
7	11	4
13	19	10

C	D	E
1	7	3
7	9	6
13	11	8
14	16	14

A	C	E
4	1	8
7	4	3
10	14	20
1	19	7

Consideriamo inoltre una Join Table per la classe globale G che per semplicità e per riportarci nel contesto della precedente sezione supponiamo che i join

condition siano espresse come natural join ovvero tra attributi comuni delle varie condizioni. Quindi:

1. $F_{12} \Longrightarrow R_1$.A = R_2.A
2. $F_{23} \Longrightarrow R_2$.C = R_3.C
3. $F_{13} \Longrightarrow R_1$.E = R_3.E
4. $F_{14} \Longrightarrow R_1$.AE = R_4.AE
5. $F_{24} \Longrightarrow R_2$.AC = R_4.AC
6. $F_{34} \Longrightarrow R_3$.CE = R_4.CE

Noi non ci poniamo il problema di calcolare una sequenza (in senso stretto) di outer join, ma di calcolare una rappresentazione alternativa della query che genera la Full Disjunction applicando l'algoritmo PSOJ. Dunque secondo la definizione data in 3.2 la Full Disjunction che dobbiamo implementare è:

A	**B**	**C**	**D**	**E**	**F**
1	2	7	-	3	1
4	5	1	-	6	2
10	-	-	-	12	3
18	-	-	-	20	4
7	11	4	-	3	-
13	19	10	-	-	-
-	-	13	11	8	-
10	-	14	-	20	-
1	-	19	-	7	-
4	5	1	-	8	-
1	2	7	9	6	-
4	5	1	7	3	-
7	11	4	16	14	-

Sulla base dei risultati ottenuti da Galindo-Legaria, consideriamo il grafo relativo in Figura 4.9 e notiamo che tale grafo è γ-ciclico e quindi la Full Disjunction delle relazioni (R_1, R_2, R_3, R_4) non può essere calcolata attraverso una sequenza di Outer Join. Tale grafo γ-acilcico ha come nodi gli attributi delle relazioni R_i e come archi le relazioni stesse e viene costruito nel seguente modo:

- Per ogni relazione R_i si crea un iperarco che contiene tutti gli attributi, che rappresentano i nodi, appartinenti alla relazione stessa.

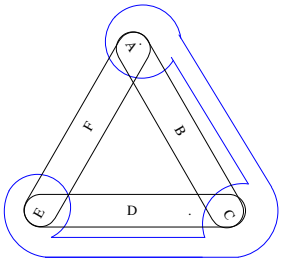

Figura 4.9: Grafo γ-ciclico

- Per ogni $F_{k,q}$ del tipo R_k.A = R_q.A viene generato un solo nodo A, incluso in tutte e due le relazioni R_k ed R_q.

- Per ogni F_k del tipo $F_k(A, B, C)$, dove A,B e C sono degli attributi globali per cui esiste una Join Table, vengono create tante relazioni quanti sono le classi coinvolte nella Join Table e per ognuna vengono definiti i predicati di outer join corrispondenti estrapolando l'informazione neccessaria dalla Join Table.

- L'ipergrafo ottenuto ripetendo i due passi precedenti è completo e spesso sovrabbondante di informazioni, perciò viene ridotto eliminando gli iperarchi che sono propriamente inclusi in un qualsiasi altro iperarco che fa parte dell'ipergrafo.

Illustriamo intuitivamente il metodo proposto per il calcolo della Full Disjunction facendo riferimento al diagramma rappresentativo di Figura 4.10. Tale diagramma viene costruito a partire dal grafo la generazione del quale è descritta sopra ha come nodi le relazioni ovvero gli iperarchi del grafo mentre i lati rappresentano le condizioni di outer join $F_{i,j}$. Nella rappresentazione supponiamo che eventuali predicati di join indiretto sono stati già trasformati durante la fase di integrazione. Ciò vuol dire che per ogni due classi S_1 e S_2 (a cui corrispondono due relazioni R_1 e R_2) a cui dobbiamo applicare un predicato di join indiretto (rappresentato da una JoinTable R_j) sono state create due relazioni indotte R_1' e R_2" che sono il risultato del join naturale tra ogni una delle relazioni interessate R_1 e R_2 e la relazione di "passaggio" R_j.

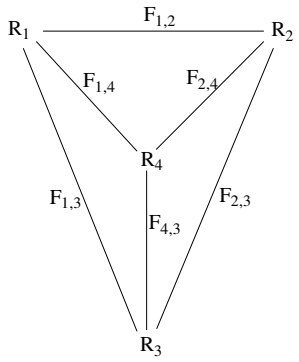

Figura 4.10: Diagramma rappresentativo

- si sceglie come nodo di partenza, per esempio, la relazione R_2 (questa operazione sarà discussa in dettaglio più avanti);

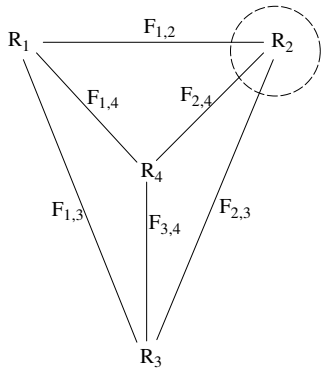

Figura 4.11: Diagramma rappresentativo alla prima iterazione

- si genera il taglio che comprende solo il nodo R_2 e si verifica quali sono i lati intersecati dal taglio. Nel nostro caso risultano $(R_1 \longleftrightarrow R_2)$,$(R_2 \longleftrightarrow R_4)$ e $(R_2 \longleftrightarrow R_3)$ (vedi Figura 4.11;
- si esegue l'operazione di inglobamento delle relazioni ottenendo il diagramma rappresentato in Figura 4.12

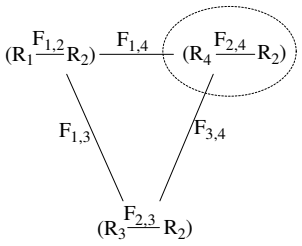

Figura 4.12: Diagramma rappresentativo alla seconda iterazione

- si genera il secondo taglio su ($R_2 \longleftrightarrow R_4$) e si ripetono le operazioni descritti sopra.

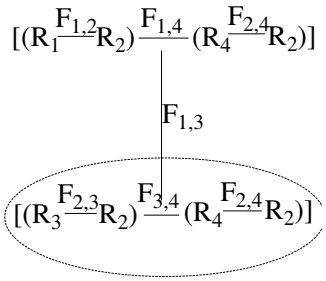

Figura 4.13: Diagramma rappresentativo alla terza iterazione

- si genera il terzo taglio su ($R_2 \longleftrightarrow R_4$)$\longleftrightarrow$($R_2 \longleftrightarrow R_3$), si esegue l'-operazione di inglobamento e alla fine si ottiene l'espressione di Figura 4.14

Riassumendo si ottiene la peudo-sequenza di join:

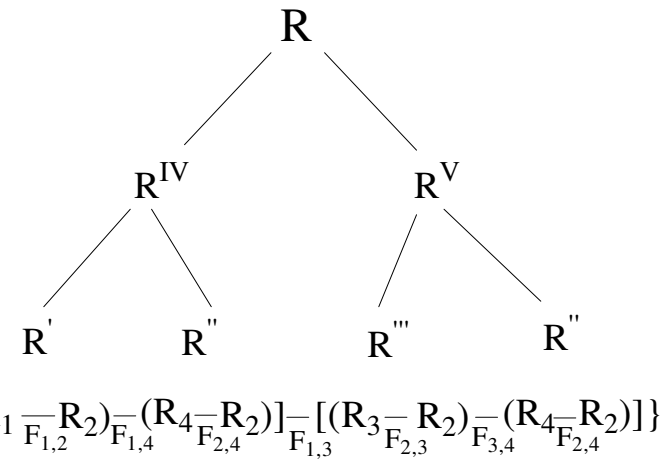

Figura 4.14: Diagramma rappresentativo della quarta iterazione

$$\{[(R_1 =\!\bowtie_{F_{1,2}}= R_2) =\!\bowtie_{F_{1,4}}= (R_4 =\!\bowtie_{F_{2,4}}= R_2)] =\!\bowtie_{F_{1,3}}= [(R_3 =\!\bowtie_{F_{2,3}}= R_2) =\!\bowtie_{F_{3,4}}= (R_4 =\!\bowtie_{F_{2,4}}= R_2)]\}$$

Tale espressione può essere implementata su un DBMS commerciale che supporta lo standard SQL92. Si può notare che alcune operazioni di outer join sono ripetute e pertanto è opportuno effettuare delle semplificazioni/ottimizzazioni dell'espressione; questo aspetto non è trattato nella presente tesi è sarà l'obbiettivo di un lavoro futuro. Notiamo che alcune tecniche sono state già proposte da Galindo-Legaria e sono basate su una rappresentazione ad albero dell'espressione, come mostrato appunto in Figura 4.14.

A questo punto presentiamo l'algoritmo in modo più formale.
Per l'applicazione dell'algoritmo PSOJ abbiamo bisogno delle seguenti informazioni:

- Schema globale **S**($\mathcal{G}$), Mapping Table, Join Table.
- L'insieme delle relazioni locali $\mathcal{R} = \{R_1, \ldots, R_n\}$ trascritti sullo schema globale **S**($\mathcal{G}$).
- $\mathcal{F} = \{F_{i,j} \mid i < j, i = 1 \ldots n-1; j = 2 \ldots n\}$

Si noti che nel nostro caso i diagrammi indotti sono sempre completamente connessi. Ricordiamo che $F_{i,j} = false$ significa che non c'è intersezione tra le calssi L_i ed L_j quindi in questo caso il full outer join si riconduce al "prodotto cartesiano" con un successivo padding delle tuple.

Calcolo della Full Disjunction

Input: * $\mathcal{R} = \{R_1, \ldots, R_n\}$: relazioni definiti sullo schema globale.
* $\mathcal{F} = \{F_{i,j}\}$: Join Table per tutte le relazioni di $\mathcal{R}$.
Output: pseudo-sequenza per calcolare la Full Disjunction
Procedimento:
• Poni $F_0 = \mathcal{F}$, $R_0 = \mathcal{R}$ ed $NP = (n-1)$ numero dei passi.
• for $(i = 0; i < NP; i++)$
  • seleziona $R_i \in \mathcal{R}_i$
    $\mathcal{R}_{i+1} = \{R_i =\bowtie_f= R'_i \mid R'_i \in \mathcal{R}_i; R'_i \neq R_i; f = OJC_i(R_i, R'_i)$ non è null$\}$
• Restituisci $\mathcal{R}_{NP}$.

Tabella 4.2: Algoritmo PSOJ

La funzione di Outer Join Condition ($OJC_i(R_i, R'_i)$) restituisce la JC da applicare tra R_i ed R'_i sulla base delle relazioni usate per definire le R_i ed R'_i ed è definita nel seguente modo:

$$OJC_i(R_i, R'_i) = \begin{cases} JC(R_0, R'_0) \in \mathcal{F} & \text{se } i = 0 \\ OJC_{i-1}(R^a_{i-1}, R^b_{i-1}) & \text{tale per cui} \\ & \exists R^c_{i-1} \in \mathcal{R}_{i-1} \mid \\ & R_i = R^c_{i-1} =\bowtie= R^a_{i-1} \\ & R'_i = R^c_{i-1} =\bowtie= R^b_{i-1} \end{cases} \quad (4.1)$$

Verifichiamo l'algoritmo sul nostro esempio di riferimento:
$\mathcal{R}_0 = \{R_1, R_2, R_3, R_4\}$
al passo $i = 0$: selezioniamo $R_2 \in \mathcal{R}_0$
otteniamo: $\mathcal{R}_1 = \{R^a_1 = R_4 =\bowtie_{F_{4,2}}= R_2$
$R^b_1 = R_3 =\bowtie_{F_{3,2}}= R_2$
$R^c_1 = R_1 =\bowtie_{F_{1,2}}= R_2\}$
al passo $i = 1$: selezionaiamo $R^a_1 \in \mathcal{R}_1$
otteniamo: $\mathcal{R}_2 = \{R^a_2 = R^c_1 =\bowtie_{F_{1,4}}= R^a_1$
$R^b_2 = R^b_1 =\bowtie_{F_{3,4}}= R^a_1\}$
al passo $i = 2$: selezionaiamo $R^b_1 \in \mathcal{R}_2$
otteniamo: $\mathcal{R}_3 = \{R^a_3 = R^b_2 =\bowtie_{F_{1,3}}= R^a_2\}$

Mostriamo anche il calcolo delle OJC. Per esempio al passo $i = 2$ abbiamo:
$OJC_2(R^b_2, R^a_2) = OJC_1(R^c_1, R^b_1) = OJC_0(R_3, R_1) = F_{1,3}$

Si noti che l'algoritmo, ad ogni iterazione opera su coppie di relazioni riducendo il numero dei lati del grafo di un numero pari al numero di lati interseccati dal arco generato. Dunque l'algoritmo si arresta quando rimane soltanto un nodo. Questo nodo in pratica rappresenta la pseudo-sequenza da noi cercata.

Si noti inoltre che dalla pseudo sequenza generata si può osservare che gli operatori di NOJ sono applicati a delle coppie di relazioni e che alcune coppie si ripetono. D'altro canto alcune operazioni di join possono essere eseguite in parallelo senza alterare il risultato finale poichè sono del tutto indipendenti. Dunque se noi sostituissimo al risultato di ogni operazione di join una relazione indotta essa andrà in join con un'altra operazione indotta e cosi via fino alla creazione di un albero, la radice del quale rappresenterà la risposta della query. L'albero risultante è un albero binario perfettamente bilanciato. Dunque le operazioni di join che si trovano sulle foglie possono essere eseguite in parallelo, i risultati delle quali diventeranno foglie nell'albero ridotto.

Infine, l'operatore di natural outer join è implementato nel SQL92 e per adesso ci appogiamo sugli algoritmi di esecuzione implementati nelle diverse DBMS commerciali.

Teorema 4.2.1 (Validità dell'algoritmo PSOJ.) *Dato un insieme* ***R***=$\{R_1, \ldots, R_n\}$ *di relazioni definite sullo schema globale* ***S(*** $\mathcal{G}$ ***)*** *e dati* ***P*** = $\{P_1, \ldots, P_m\}$ *predicati di ugualianza del tipo* ***A=B*** *oppure* $R_i.A = R_j.B$ *e/o predicati di selezzione* $\mathcal{P}$ = $\{\mathcal{P}_i, \ldots, \mathcal{P}_k\}$ *definiti da altretanti Join Table, allora: L'algoritmo PSOJ genera una pseudo-sequenza di operatori di natural outer join che produce la completa disgiunzione(FD).*

Dimostrazione. Dimostriamo il teorema per induzione.

a) Definiamo il caso base cosi:
Date due relazioni $R^{'}$ e $R^{''}$ che formano un diagramma con un solo lato(Figura 4.15), applicando l'algoritmo PSOJ il diagramma si trasforma in due relazioni collegate da un operatore di natural outer join. Per la definizione del natural outer join questo produce la full disjunction $FD(R^{'}, R^{''})$. Dunque nel caso n = 1 lato l'algoritmo è valido.

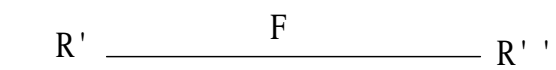

Figura 4.15: Caso base.

b) Supponiamo che date R = $\{R_1, \ldots, R_{k-1}\}$ relazioni sullo schema globale **S(**$\mathcal{G}$**)** con i rispettivi predicati di join ed il corrispondente diagrama connesso, l'algoritmo PSOJ genera la $FD(R_1, \ldots, R_{k-1})$.

Dobbiamo dimostrare che l'algoritmo è valido per R = $\{R_1, \ldots, R_{k-1}\} \bigcup \{R_k\}$, cioè per un diagramma di k lati.

c) Secondo la costruzione dell'algoritmo dati n nodi il massimo numero di lati che gli collegano(denotato con m) si hanno quando il grafo generato a partire dagli n nodi è completamente connesso ed è $m_{max} = \frac{n(n-1)}{2}$, mentre il minimo numero si ha quando il grafo è connesso, ma aciclico ed è $m_{min} = (n-1)$. In ogni caso ad ogni passo viene generato un arco che include al suo interno solo un nodo e dunque dal grafo viene eliminato solo questo nodo. Da questo segue che ad ogni passo il numero di nodi diminuisce di uno. Dunque al k-esimo passo si ha $m_{max} = \frac{(n-k)(n-(k-1))}{2}$ e $m_{min} = (n-(k-1))$. D'altro canto per costruzione ogni lato interseccato dal arco generato viene sostituito dall'-operatore di natural outer join tra due relazioni e ciò come abbiamo detto genera la FD tra questa coppia di relazioni. Dunque ad ogni passo viene applicato il caso base.
Allora supponiamo di aggiungere un'altra relazione R_n e di voler calcolare la FD($\{R_1, \ldots, R_{n-1}\} \bigcup \{R_n\}$). Per quanto detto sopra questa modifica non avrà nessuna influenza sul procedimento di applicazione dell'algoritmo, poichè ad ogni passo noi trattiamo solo due relazioni alla volta. Dunque l'algoritmo PSOJ genera la FD di un numero arbitrario di relazioni.

□

Il Teorema precedente afferma che per un numero arbitrario di nodi l'algoritmo PSOJ genera la FD a partire dal grafo connesso che rappresenta le relazioni coinvolte e le connessioni tra di loro. Se il grafo corrispondente alle relazioni risultasse non connesso l'algoritmo deve essere applicato a tutte le parti connesse.

Corollario 1 *L'algoritmo PSOJ genera la FD per un qualsiasi grafo connesso.*

Facciamo alcune considerazioni nel caso in quale il grafo corrispondente alle relazioni in esame non risulta completamente connesso. Questo può essere utile quando vogliamo applicare la nostra tecnica per il calcolo della Full Disjunction in qualche altro contesto. In particolare possiamo dire che si possono verificare due casi generali:

a) Il diagramma è completamente connesso (questo è il caso generale nel nostro contesto);

b) Il grafo non è completamente connesso.

Nel primo caso possiamo affermare che la scelta del nodo è completamente arbitraria, mentre nel secondo bisogna scegliere un nodo opportuno. Più pecisamente

il taglio che dobbiamo individuare è quello di costo minore.
Definiamo come costo di un taglio il numero di archi appartenenti al diagramma che il taglio stesso interseca. Il numero dei passi per ottenere la pseudo-sequenza di NOJ è proporzionale al numero degli archi del diagramma. Dunque se ad ogni passo scegliamo il taglio che interseca il maggior numero di archi, cioè il taglio di costo maggiore, avremo il minor numero di iterazioni. Nel caso in cui il diagramma risultasse completamente connesso tutti nodi sono collegati con tutti gli altri allora il costo di tutti i tagli è lo stesso. Proprio per questo fatto in questo caso la scelta del nodo di partenza è arbitraria. Nel secondo caso invece quando il diagramma risulta non completamente connesso non possiamo seguire lo stesso procedimento perchè si rischia di staccare parti del grafo dividendolo in due parti non connessi e dunque l'algoritmo di generazione della PSOJ si interrompe prima di determinare la pseudo-sequenza. Per ovviare questo problema basta procedere nel modo opposto del primo caso, cioè ad ogni passo si sceglie il taglio di costo minore. L'algoritmo per determinare il taglio di costo minimo è rappresentato in Tabella 4.3.

In questo caso il costo dell'algoritmo aumenta, ma comunque in tutti i due casi il costo totale di esecuzione dell'algoritmo PSOJ è minore dal costo totale del metodo di calcolo della Full Disjunction proposto da Galindo-Legaria nella sezione precedente.

Il discorso sulla scelta del nodo di partenza o più in generale sul nodo su cui effettuare il taglio può ritornare utile anche nel nostro contesto.
Infatti il nostro grafo è sempre completamente connesso, ma vi possono essere degli archi la cui condizione è "$false$". È intuibile che un full outer join su una condizione "$false$" restituisce un numero massimo di tuple pari alla cardinalità $\mid R_1 \mid \times \mid R_2 \mid$ quindi è ovvio "ritardare" queste operazioni il più possibile in modo da realizzarle su insiemi più piccoli di tuple.
Intuitivamente questo si può ottenere selezionando un nodo che ha un minor numero di OJC "$false$". Per questo scopo è applicabile l'algoritmo presentato in Tabella 4.3 con una piccola modifica: nella condizione di controllo su $F_{i,j}$ basta sostituire $null$ con $false$.

Per completare il discorso dobbiamo osservare che dopo aver trovato la pseudo-sequenza di NOJ l'esecuzione della query cosi ottenuta viene affidata ad un DBMS commerciale. Tale DBMS non presenta algoritmi di ottimizzazione per la forma della PSNOJ ottenuta e dunque come un lavoro futuro si può ipotizzare una loro implementazione.

Basandosi su questo metodo di calcolo della Full Disjunction la nostra ricerca ha portato alla definizione di un algoritmo Query Processing (RWNBE) che sarà presentato nella sezione successiva.

Input: * $\mathcal{R} = \{R_1, \ldots, R_i, \ldots, R_n\}$: relazioni definiti sullo schema globale.
* $\mathcal{F} = \{F_{i,j}\}, i = 1, \ldots, n; j = 1, \ldots, m$: Join Table per tutte le relazioni di $\mathcal{R}$.
* Diagramma D indotto.
Output: nodo di costo minimo.
Procedimento:
• Poni $old = \infty$, $R_s = \emptyset$, $i = 0$
• While $\mathcal{R} \neq \emptyset$
• seleziona $R_i \in \mathcal{R}$
poni $\mathcal{R} = \mathcal{R} - \{R_i\}$
$new = 0$
for $j = 0$ to m do
if $F_{i,j} \neq null$ then
$new = new + 1$
endif
endfor
if $new < old$ then
$R_s = R_i$
$old = new$
endif
endwhile
• Restituisci R_s che rappresenta la relazione a cui corrisponde il nodo su cui effettuare il taglio di costo minimo.

Tabella 4.3: Algoritmo per il calcolo del taglio di costo minimo.

4.3 Algoritmo di riscrittura della query

In questo paragrafo entreremo nello specifico di ogni passo che fa parte dell'algoritmo RWNBE che fa parte della fase di Query Processing. Di questa fase di gestione della query si occupa il modulo Query Manager che utilizza tutte le informazioni intensionali prodotte nelle fasi di integrazione degli schemi dal modulo Global Schema Builder. Quest'ultimo genera lo Schema Globale (**S**($\mathcal{G}$)), le Mapping Table (MT), le Join Map e le rispettive Join Table. Tutte queste strutture rappresentano le informazioni di input per il nostro algoritmo. La fase di Query Processing ha come fine ultimo l'ottenere una risposta corretta, esaustiva e minima all'interrogazione posta attraverso la Query Globale. L'intero procedimento è presentato nella Fig. 4.3 e fa parte del modulo Query Manager.

Come mostrato nella Fig. 4.3 si possono individuare tre principali attività che

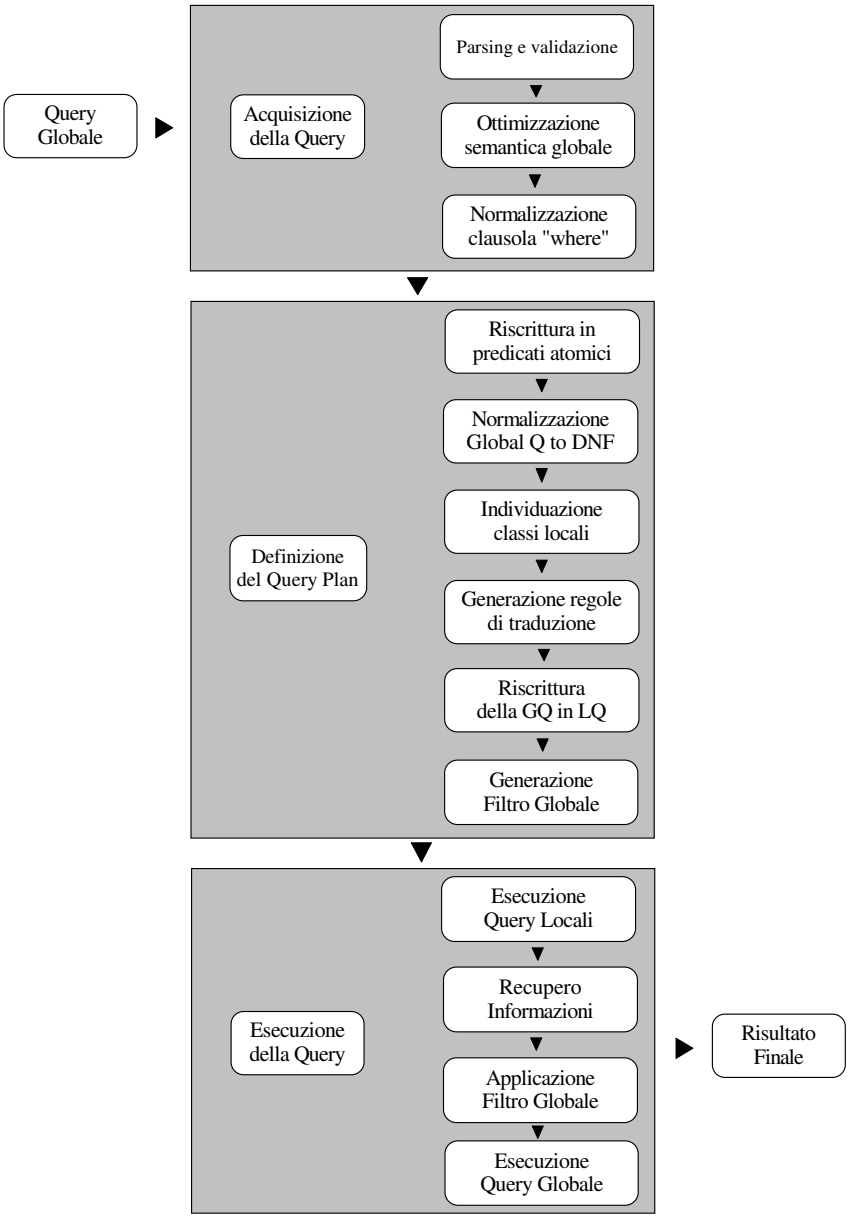

Figura 4.16: Query Processing.

vengono eseguite in sequenza:

1. Aquisizione della Query Globale;

2. Definizione del Query Plan;

3. Esecuzione della Query;

La prima attività è l'*Acquisizione della Query* e si suddivide in tre momenti distinti. Inizialmente avviene la fase di *Parsing e Validazione*, in cui si verifica la correttezza della query, sia da un punto di vista sintattico che semantico. In questa attività viene impiegato un modulo di riconoscimento grammaticale, che ha il compito di verificare la correttezza sintattica della query rispetto alla grammatica OQL_{I^3} e di produrne poi un'immagine dell'espressione stessa in memoria centrale. La struttura cosi generata sarà utilizzata per la validazione della query e prevede la verifica della correttezza semantica di questa, accertando che le classi e gli attributi coinvolti appartengono effettivamente allo schema integrato a cui la query è rivolta.
In secondo luogo avviene l'*Ottimizzazione semantica globale*. Questa é realizzabile se sono presenti regole di integritá inter-sorgenti definite sullo schema globale. Questi vincoli sono espressi con rule ODL_{I^3} sullo schema globale e vengono impiegati da ODB_Tools per riformulare la query producendone una semanticamente equivalente, ma eseguibile in modo più efficiente. Ad esempio vengono eliminati predicati ridondanti oppure vengono aggiunte nuove condizioni che possono abbreviare i tempi di risposta per la possibile presenza, nelle sorgenti, di indici sui predicati introdotti. Oppure vengono eliminate condizioni di selezione ridondanti. Quest'ultima operazione è necessaria per evitare costose navigazioni dovute al fatto che alcuni predicato potrebbero contenere join impliciti.
Infine vi è la *Normalizzazione della clausola where*. Alla fine delle due fasi precedenti, la clausola where è rappresentata da un'espressione booleana innestata ricorsivamente. Per poter proseguire, il Query Manager deve trasformare questa generica struttura booleana in forma normale congiuntiva (CNF). Avere la clausola where in questa forma, consente di ottimizzare la fase di esecuzione della query, in quanto in primo luogo possono essere valutati il maggior numero di predicati in and e quindi viene minimizzata la quantità di risultati restituiti dalle sorgenti locali, con una conseguente velocizzazione della fase di integrazione; in secondo luogo la forma CNF della query come spiegato nella Sezione 3.4.5 consente la generazione dell'albero Master.

La seconda attività svolta dal Query Manager nella fase di Query Processing, è la *Definizione del Query Plan*, in cui la query posta in termini globali viene elaborata e trascritta in query locali secondo gli schemi locali delle singole sorgenti. In questa fase viene generato il *Piano di Esecuzione*, che contiene le informazioni ed i metodi per generare le query locali e per ricostruire la risposta globale.
Il primo passo che viene svolto è la decomposizione dei predicati che contengono operatori logici in predicati atomici. Successivamente la Query Globale viene trascritta nella sua forma normale disgiuntiva (DNF), poichè questa forma consente la separabilità dei termini congiuntivi della condizione di selezione della

query globale.

Dopo l'individuazione delle classi locali coinvolte nell'interrogazione e la generazione delle regole di traduzione la Query Globale viene trascritta nelle Query Locali. Durante queste fasi parallelamente vengono individuati i predicati residui (quei predicati che non sono mappati interamente in nessuna classe) e generato il Filtro Globale da applicare in fase di Object Fusion. Questo avviene analizzando la Query Globale, per ricercare gli attributi globali contenuti nei predicati di proiezione e di selezione. La generazione del Filtro Globale e la trascrizione della GQ nelle Query Locali porta ad una notevole semplificazione del Query Plan, minimizzando il numero di query da inviare alle sorgenti. Questa semplificazione si attua nei casi in cui si verifica durante la fase di integrazione il fatto che la stessa informazione viene mappata in classi diverse. Dunque basta generare una sola Query Locale per una delle classi per ottenere l'informazione desiderata. Il Query Manager esprime le query locali in OQL, lasciando ai wrapper l'incombenza di tradurle nel linguaggio adatto in ogni sorgente.
Il piano d'accesso ottenuto fino ad adesso, può subire dei miglioramenti. Questi miglioramenti si possono ottenere analizzando eventuali predicati di selezione nelle Local Query, che contengono valori di default. Un'altra semplificazione è ottenibile nel caso in cui sia possibile raggruppare più Local Query da inviare ad una stessa sorgente, effettuando localmente i join.
Inoltre è possibile realizzare l'*ottimizzazione semantica locale* (che consente di realizzare query meno onerose), sfruttando, tramite l'impiego di ODB-Tools, le regole di integrità definite sulle singole sorgenti. Quest'ultima fase di semplificazione è posta a livello di mediatore, in quanto non tutte le sorgenti potrebbero essere in grado di realizzarla.

La terza attività che costituisce il *Query Processing* è l'*Esecuzione della Query*, dove dopo l'esecuzione delle Query Locali, il Query Manager utilizza le informazioni del piano, generato nelle fasi precedenti, per ricomporre i risultati ottenuti dalle sottoquery, presentando all'utente una risposta integrata. Questa fase prevede quattro livelli: Esecuzione delle Local Query, Recupero delle Informazioni, Applicazione del Filtro Globale ed esecuzione della Global Query.
Per realizzare quest'ultima attività il Query Manager sfrutta un DBMS, che gli permette di creare delle tabelle in grado di memorizzare i risultati temporanei degli stadi intermedi della ricostruzione della risposta. Così il Query Manager può eseguire varie operazioni di ricostruzione e fusione attraverso query (join e outer join) poste su queste tabelle temporanee.
Nella fase di Applicazione del Filtro Globale vine applicato l'algoritmo di generazione di una pseudo-sequenza di join che produce la completa disgiunzione (Full Disjunction) tra i vari risultati delle Query Locali. Come definito nella sezione precedente la generezione della Full Disjunction ci garantisce che il risul-

tato contenga tutta l'informazione di cui abbiamo bisogno. Nel prossimo paragrafo daremo la definizione dell'algoritmo che calcola la Full Disjunction (PSOJ) e dimostreremo la sua validità.

Capitolo 5

Analisi e ottimizzazione semantica in MOMIS

Il Query Manager, come già visto nel Capitolo 1, è il modulo che gestisce la fase di *Query Processing*, in cui viene eseguita la query posta dall'utente sulla vista globale. Per svolgere il suo compito il Query Manager utilizza le strutture dati create nella fase di integrazione, descritta nel capitolo precedente, strutture che rappresentano la conoscenza intensionale ed estensionale necessaria per ottenere una risposta corretta e quanto più possibile completa e minima. Nel capitolo precedente abbiamo presentato un metodo di Query Processing che non fa uso dell'ottimizzazione semantica. Tale metodo alternativo è ancora in fase di sviluppo. In questo capitolo presenteremo le techniche di analisi e ottimizzazione semantica presenti attualmente in MOMIS.

Il processo di gestione delle interrogazioni può essere suddiviso in tre fasi principali che vengono eseguite in sequenza dal Query Manager:

1. *Acquisizione della Query*:
   in questa fase la query viene caricata nelle apposite strutture dati che ne rappresentano il contenuto. Il Query Manager ne verifica quindi la correttezza sintattica e semantica e ne esegue l'ottimizzazione. A questo punto la clausola where, acquisita inizialmente come una generica espressione booleana, viene posta in forma normale congiuntiva.

2. *Definizione del Query Plan*:
   la query acquisita al passo precedente è espressa in termini globali, in quanto posta dall'utente sulla vista integrata, è quindi necessario ricondurla agli schemi locali delle singole sorgenti. In questa fase viene generato il *Piano di Esecuzione* da associare alla query, che contiene non solo le informazioni relative alla generazione delle *Local Query*, ma anche quelle che consentiranno la successiva ricomposizione dei risultati.

3. *Esecuzione della Query*:
   in questa fase vengono eseguite dai wrapper le *Local Query* generate in precedenza ed i risultati restituiti vengono rielaborati seguendo le indicazioni contenute nel Piano di Esecuzione.

Le tre fasi appena citate verranno analizzate nelle sezioni seguenti, ponendo particolare attenzione alla gestione della clausola where della query (sia per quanto riguarda la normalizzazione che per la traduzione), all'individuazione delle Base Extension e all'esecuzione della query, che costituiscono uno degli argomenti approfonditi da questa tesi.

5.1 Acquisizione della Query

Come già detto in precedenza, la query posta dall'utente è sottoposta ad alcune trasformazioni che la rendono eseguibile dal Query Manager.
Questa fase iniziale è suddivisa in tre momenti distinti:

- *parsing e validazione*;
- *ottimizzazione semantica globale*;
- *normalizzazione della clausola where*.

L'acquisizione della query comporta la verifica di correttezza sintattica e semantica e la successiva ottimizzazione, anche se in realtà all'interno di MOMIS le fasi di ottimizzazione e parsing sono invertite per sfruttare al meglio il software esistente.

5.1.1 Parsing e validazione

A livello teorico, prima di effettuare il processo di ottimizzazione deve essere verificata la correttezza della query, sia da un punto di vista sintattico che semantico. Per questo viene utilizzato un modulo di riconoscimento grammaticale, il modulo di parsing, che ha il compito di verificare la correttezza sintattica della query, rispetto alla grammatica OQL (riportata in Appendice D) e ne produce un'immagine in memoria centale. A questo punto la query viene validata, cioè ne viene accertata la correttezza semantica verificando l'effettiva appartenenza delle classi e degli attributi coinvolti allo schema integrato a cui è rivolta la query.

5.1.2 Ottimizzazione semantica globale

L'ottimizzazione è realizzabile se sono presenti regole di integrità inter-sorgenti definite sullo schema globale. Questi vincoli sono espressi con rule ODL_{I^3} e vengono impiegati da ODB_Tools per riformulare la query producendone una semanticamente equivalente, ma eseguibile in modo più efficiente.
Ad esempio vengono eliminati predicati ridondanti oppure vengono aggiunte nuove condizioni che possono abbreviare i tempi di risposta per la possibile presenza, nelle sorgenti, di indici sui predicati introdotti. In riferimento all'esempio introdotto in 2.1.1, supponiamo che esista una relazione tra il numero di impiegati (`employee_nr`) e i fondi a disposizione del dipartimento (`budget`). Questo legame viene espresso dal progettista per mezzo di una rule ODL_{I^3} che definisce la seguente regola di integrità sulla classe globale `Workplace`:

```
rule RG1  for all X in Workplace: (X.employee_nr > 100)
             then X.budget > 20000
```

Consideriamo la query seguente: *"Selezionare i nomi dei dipartimenti appartenenti al settore 'Engineering', con numero di impiegati maggiore di 150"*

```
Q:  select name
    from Workplace
    where area = 'Engineering'
    and employee_nr > 150
```

ODB-Tools espande semanticamente la query `Q` sfruttando la regola `RG1` ed ottiene la seguente query:

```
Q': select name
    from Workplace
    where area = 'Engineering'
    and employee_nr > 150
    and budget > 20000
```

Le query `Q` e `Q'` sono semanticamente equivalenti, ma in `Q'` è stato aggiunto un predicato nelle clausola *where*, che potrebbe velocizzare la fase di Query Processing delle singole sorgenti se sull'attributo aggiunto *budget* fossero definiti degli indici. Il prezzo da pagare è un aumento della complessità della fase di *Definizione del Query Plan* in quanto deve essere riformulata una query più complessa.

5.1.3 Normalizzazione della clausola where

La clausola where della query, dopo le fasi descritte in 5.1.1 e 5.1.2, è rappresentata da un'espressione booleana innestata ricorsivamente.
Il Query Manager, prima di iniziare la *Definizione del Query Plan*, trasforma questa generica struttura booleana in **forma normale congiuntiva**:

Definizione 5.1.1 (Forma Normale Congiuntiva) *Date n variabili di ingresso, una formula booleana viene detta in* forma normale congiuntiva *o CNF quando è 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.*

Avere la clausola where in questa forma, consente di ottimizzare la fase di esecuzione della query, in quanto vengono valutati il maggior numero di predicati in and e quindi viene minimizzata la quantità di risultati restituiti dalle sorgenti locali, con una conseguente velocizzazione della fase di integrazione.

Per ottenere la forma normale, la clausola where viene analizzata ricorsivamente e trasformata utilizzando i **teoremi dell'algebra di commutazione** presentate nel capitolo 3.

5.2 Definizione del query plan

La query posta dall'utente viene validata e ottimizzata durante la fase descritta nel Capitolo 5.1, la struttura così generata prende il nome di Global Query. In Figura 5.1 sono rappresentate le operazioni effettuate dal Query Manager che, partendo dalla Global Query, portano alla produzione del risultato finale da restituire all'utente.

In questa sezione verrà descritta la fase di *Definizione del Query Plan*, durante la quale vengono generate le Basic Query, individuate le classi locali coinvolte e preparate le Local Query da inviare alle sorgenti. Parallelamente a queste attività il Query Manager raccoglie le informazioni necessarie per la fase successiva di *Esecuzione della Query*.

L'approccio consiste nel mettere a disposizione al Query Manageruna *esplicita rappresentazione* della conoscenza estensionale disponibile sovraponendo o combinando l'informazione diponibile nelle classi locali coinvolte nell'interrogazione. Questa conoscenza estensionale è generata durante la fase di integrazione come un set di asserzioni di: contenimento, equivalenza e disgiunzione fra le classi locali delle diverse sorgenti coinvolte nell'integrazione. La rappresentazione esplicita è il risultato della computazione di tutte le possibili estensioni effettuato con l'intervento diretto del designer e delle asserzini estensionali. Più sono le asserzioni

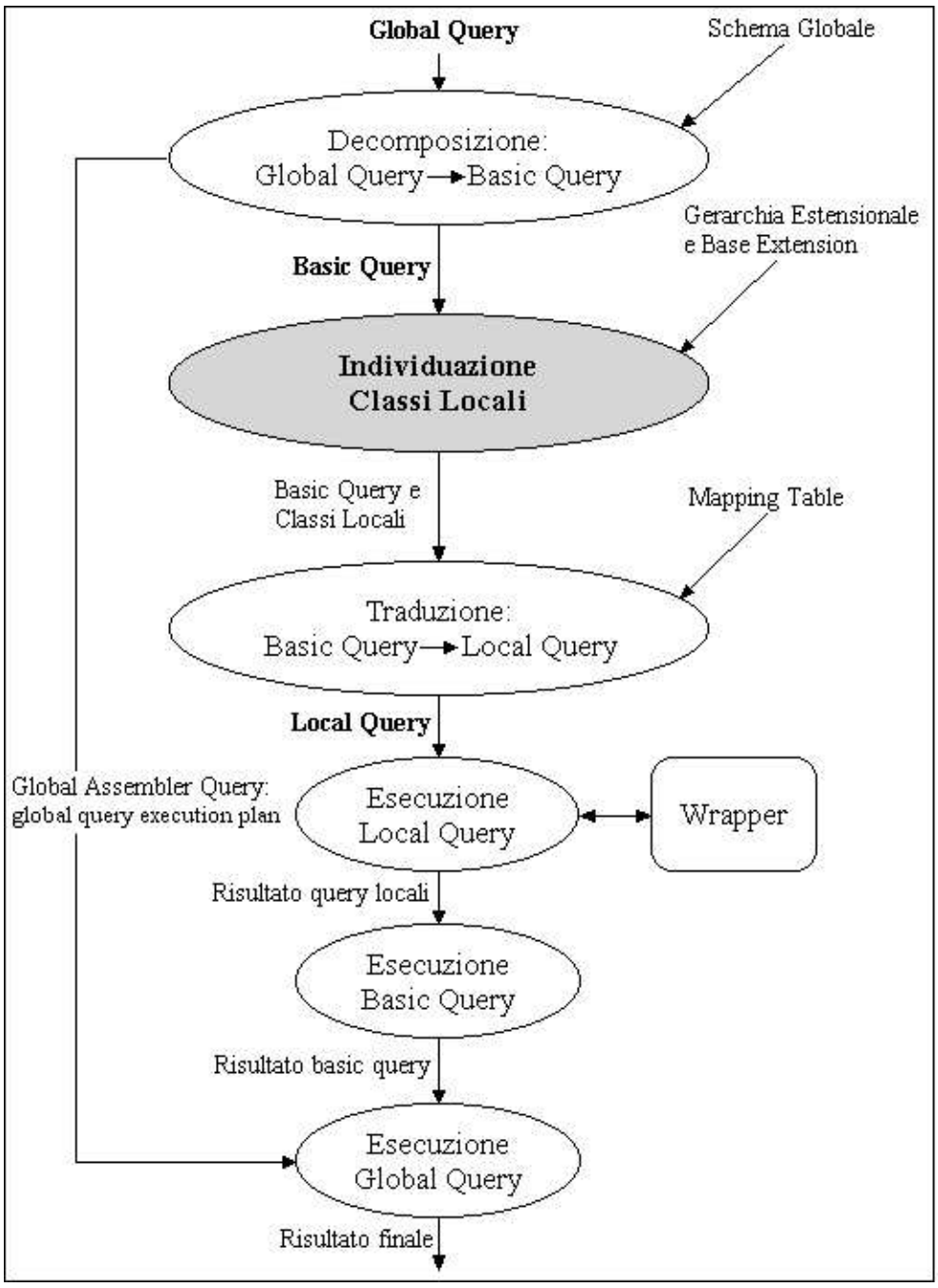

Figura 5.1: Operazioni del Query Manager

estensionali e meno sono le Base Extension. Tutto questo procedimento viene effettuato nella fase di integrazione e per questo non grava sui costi di esecuzione

della query.

5.2.1 Decomposizione della Global Query in Basic Query

Una query, in generale, conterrà richieste rivolte a più classi globali appartenenti allo stesso schema, di conseguenza il primo passo nella *Definizione del Query Plan* è la decomposizione della Global Query in Basic Query, la cui definizione è riportata di seguito:

Definizione 5.2.1 (Basic Query) *Una Basic Query è una query contenente tutte le richieste rivolte ad una singola classe globale.*

Le Basic Query costituiscono quindi gli elementi di base per le fasi successive, senza contare che di fatto questa tipologia di interrogazioni costituisce la percentuale maggiore di query rivolte ad un Mediatore.

Le Basic Query sono espresse mediante un sottoinsieme del linguaggio OQL, riportato in Appendice E, in particolare non sono ammessi:

- join espliciti fra classi diverse, ma è possibile la navigazione attraverso aggregazioni ed associazioni per ricostruire oggetti complessi;
- subquery;
- operatori di ordinamento (order by) o di conversione (listtoset, element, flatten);
- restituzione di strutture complesse (list, array, struct).

Questi operatori devono essere implementati ad un livello superiore, in quanto agiscono su insiemi di informazioni già integrate.

A livello Global Query, il Query Manager si deve occupare di due attività:

1. determinazione delle Basic Query;
2. ricostruzione della Global Query a partire dai dati restituiti dalle Basic Query.

Per rappresentare un esempio di decomposizione da Global Query a Basic Query, facciamo per il momento una variazione alla Mapping Table riportata in Figura 2.3, e immaginiamo che non esista la navigazione implicita consentita dal mapping complesso dell'attributo globale *dept*, ma che il collegamento fra le classi globali `University_Person e Workplace`, sia fornito dal join esplicito con l'attributo globale *code*.
Con le modifiche apportate, cosideriamo la seguente Global Query `Q`: *"selezionare lo stipendio medio dei professori che lavorano in dipartimenti con budget maggiore di 10.000 dollari"*.

```
Q: select avg(University_Person.pay)
   from University_Person,Workplace
   where University_Person.dept = Workplace.code
   and University_Person.rank = 'professor'
   and Workplace.budget > 10000
```

Questa Global Query viene scomposta nelle due Basic Query `Qa` e `Qb`, rivolte rispettivamente alle classi globali `University_Person` e `Workplace`.

```
Qa: select pay, dept
    from University_Person
    where rank = 'professor'
```

```
Qb: select code
    from Workplace
    where budget > 10000
```

Viene generata una terza query `Qc`, chiamata *Global Query Assembler*, che rappresenta l'operazione di join che il Query Manager dovrà compiere per ricostruire il risultato.

```
Qc: select avg(Qa.pay)
    from Qa,Qb
    where Qa.dept = Qb.code
```

Le informazione da utilizzare per la fase di integrazione dei risultati, e le Basic Query generate vengono inserite in *execution plan evaluator*, consultato dal Query Manager durante la fase di esecuzione della Global Query.

5.2.2 Individuazione delle classi locali

Per ogni Basic Query generata al passo precedente, bisogna determinare l'insieme ottimo di classi locali a cui inviare le Local Query, per pervenire ad una risposta corretta e il più possibile completa e minima. Per *correttezza* si intende la possibilità di reperire le sole istanze che godono delle proprietà richieste e soddisfano tutte le condizioni imposte, la *completezza* indica la possibilità di individuare tutte le sorgenti che possono contribuire alla risposta ed infine per *minimalità* si intende l'individuazione del set minimo di classi locali che possono fornire una risposta corretta e completa senza aggiungere informazioni inutili e non richieste dall'utente.

Per poter eseguire correttamente questo passaggio è necessario utilizzare entrambi i tipi di conoscenza individuati in fase di integrazione degli schemi. L'uso della sola Mapping Table infatti consente di individuare quali classi locali contengono le proprietà richieste dalla query, ma non consente la ricostruzione degli oggetti virtuali che rappresentano le entità descritte in termini globali.
L'analisi effettuata esclusivamente in funzione della conoscenza intensionale porta all'ottenimento di una risposta più o meno corretta, ma sicuramente non completa, in quanto le sorgenti vengono considerate singolarmente perdendo la possibilità di ricostruire le entità proprie del contesto applicativo. Queste entità sono caratterizzate dall'avere proprietà distribuite su più classi locali appartenenti a sorgenti differenti, di conseguenza la completezza è garantita solo utilizzando anche le informazioni estensionali rappresentate dalle base extension e dalla gerarchia virtuale.
Inoltre l'impiego di tale conoscenza consente di introdurre elementi di ottimizzazione che permettono di soddisfare il requisito di minimalità: sfruttando le informazioni estensionali è possibile evitare le duplicazioni, sia eliminando le base extension dominate, sia semplificando l'insieme di classi da interrogare, come vedremo nel seguito.
Si noti che la correttezza è effettivamente garantita, mentre per quanto riguarda la completezza e la minimalità non si può assicurare il loro assoluto ottenimento. Il grado di completezza e minimalità di una risposta difatti sono legati alla scelta dei cluster e delle classi globali, quello che si può garantire è quindi che una risposta sia completa e minima rispetto alla scelta compiuta per i cluster, ma non in termini assoluti.

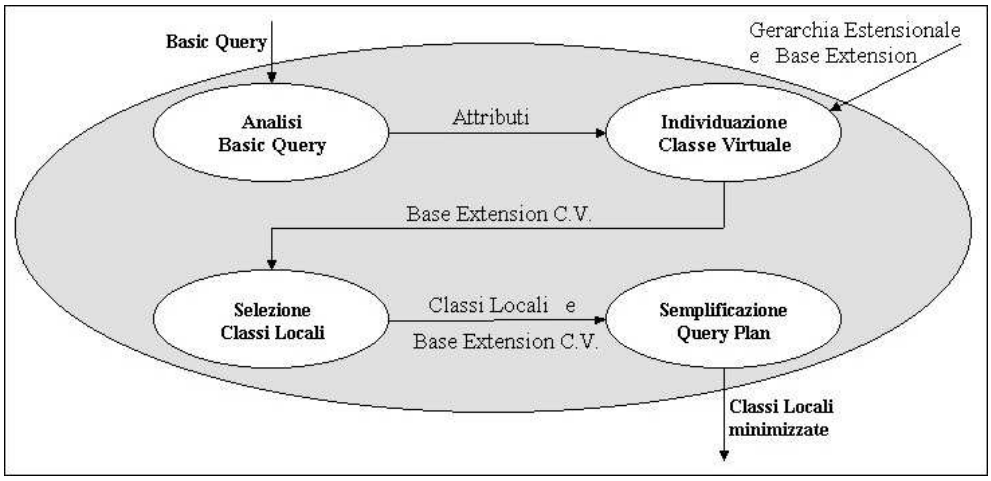

Figura 5.2: Passi della fase di individuazione delle classi locali

I passi costituenti questa fase di individuazione delle classi locali, cioè dell'insieme minimo ed ottimale di classi locali a cui accedere per ottenere una risposta corretta, completa e minima, sono schematizzati in Figura 5.2 e spiegati qui di seguito, utilizzando la simbologia introdotta in 2.2.

1. **Analisi della Basic Query**
   Il testo della Basic Query viene analizzato per individuare tutti gli attributi globali in esso contenuti.
   Una generica Basic Query può essere schematizzata come segue:

```
SELECT <select-list>
FROM G
WHERE <where-clause>
```

   Dove `<select-list>` e' un insieme di attributi globali: $f_1(A_G(G))$, mentre `<where-clause>` e' un'espressione booleana, sempre funzione degli attributi globali, già posta in forma normale congiuntiva $f_2(A_G(G))$. Viene individuato l'insieme AG_Q di attributi globali contenuti nella `<select-list>` e nella `<where-clause>`.

   Consideriamo ad esempio la query seguente:

```
Q_1: select name,dept
     from University_Person
     where faculty = 'CS'
     and (pay > 20000)
```

   Per la query `Q_1` viene determinato l'insieme:
   $AG_Q = \{$`name, dept, faculty, pay`$\}$, che costituisce il punto di partenza per i passi successivi.

2. **Individuazione delle Classi Virtuali Target**
   A questo punto bisogna utilizzare la gerachia estensionale ISA_{EXT} dello schema virtuale $(\mathbf{V}, INT, EST)$, associata alla classe globale G, per individuare le *Classi Virtuali Target*, cioe' le classi virtuali più generali che dispongono di tutte le proprietà richieste.

$$VC = \{V \in \mathbf{V} \mid AG_Q \subseteq INT(V), \text{not } \exists\, V' \neq V \mid AG_Q \subseteq INT(V) \wedge V\ ISA_{EXT}\ V'\}$$

Le classi virtuali vengono organizzate in una gerarchia di ereditarietà proprio per minimizzare i tempi di ricerca, infatti il Query Manager deve esaminare la gerarchia partendo dalla classe padre e passare al livello inferiore solo se le classe virtuale presa in considerazione non contiene le proprietà richieste.

In realtà, allo stato attuale, le classi Virtuali sono organizzate in una lista che non consente di effettuare il processo di ricerca appena descritto.

Per ora vengono eseguiti due passi successivi: in primo luogo si determina un insieme iniziale di classi virtuali VC_{in}, la cui intensione contiene tutti gli attributi AG_Q della query:

$$VC_{in} = \{V \in \mathbf{V} \mid AG_Q \subseteq INT(V)\}$$

Nell'esempio vengono selezionate: $VC_{in} = \{C7, C8, C9, C12, C13\}$. Da questo insieme iniziale, sono individuate le classi virtuali con estensione massima, quelle cioè che contengono il maggior numero di base extension.

$$VC_{opt} = \{V \in VC_{in} \mid EST(V) \supseteq EST(V_h), \forall V_h \in VC_{in}\}$$

Nell'esempio si ottiene: $VC_{opt} = \{C7\}$.

3. **Individuazione delle Base Extension**
Nel caso in cui le classi virtuali siano più di una, bisogna considerare l'unione delle base extension, in caso contrario l'insieme BE_{in} di base extension iniziali è determinato automaticamente dall'estensione della classe virtuale. Nell'esempio $VC_{opt} = \{C7\}$ e quindi $BE_{in} = EST(C11) = \{4, 5, 6, 8, 10, 12\}$. In generale vale la formula seguente:

$$BE_{in} = \bigcup_{V \in VC_{opt}} EST(V)$$

Vediamo un altro esempio:

```
Q:  select pay, rank
    from University_Person
```

L'analisi di questa semplicissima query ci fornisce l'insieme di attributi {`pay`, `rank`}; le classi virtuali più generali che li contengono entrambi nella loro intensione sono C8 e C10, la prima ha estensione formata dalle base extension $\{4, 5, 6\}$, la seconda dalle $\{10, 11, 12\}$. Nessuna delle classi virtuali candidate ad essere quella target ha estensione maggiore dell'altra, in questo caso nella fase seguente bisognerà considerare l'insieme delle base extension delle due classi.

4. **Selezione delle Classi Locali**
   Individuate le base extension sono note automaticamente anche l'insieme CL_{in} di classi da interrogare, definito come:

   $$CL_{in} = \{L \in SG(G) \mid \exists B \in BE_{in}, L \in F(B)\}$$

   Relativamente alla nostra query di esempio `Q_1`, inizialmente le base extension selezionate sono: $BE_{in} = \{4, 5, 6, 8, 10, 12\}$, esaminando la tabella riportata in Figura 2.3, dalle colonne delle base extension in questione si evince l'insieme CL_{in} delle classi locali alle quali accedere.

5. **Semplificazione del Query Plan**
   Gli insiemi BE_{in} e CL_{in} determinati devono essere ulteriormente semplificati per minimizzare il numero di query da inviare alle sorgenti. Questa semplificazione avviene in due momenti distinti:

   - **Eliminazione di base extension**:
     Può succedere che non tutte le base extension contenute in BE_{in} debbano essere ricostruite per ottenere la risposta minima, per valuare le eventuali ridondanze viene introdotto il concetto di *dominazione*:

     Definizione 5.2.2 (Dominazione) *Dato un insieme di Base Extension $\mathcal{B}_\mathcal{A}$ allora diciamo che una base extension $B_1 \in \mathcal{B}_\mathcal{A}$ domina un'altra base extension $B_2 \in \mathcal{B}_\mathcal{A}$ se le funzioni di aserzine estensionale godono della seguente proprietà:*

     $$\mathcal{E}(B_1) \supset \mathcal{E}(B_2)$$
     Per ogni $\mathcal{E}$ legale rispetto ad $\mathcal{A}$.

     Le dominazioni tra le Base Extension possono essere esplorate per effettuare l'ottimizzazine del query plan. Il concetto è stato già definito nel Capitolo **??**.

     Per ricostruire una base extension si deve effettuare il join tra tutte le classi locali che la compongono, ma le base extension rappresentano insiemi disgiunti di entità, quindi dal join tra le classi si ottiene in realtà un soprainsieme della base extension desiderata. Facciamo riferimento alla Figura 5.3 nelle quale sono rappresentate 3 classi locali (A, B, C) e le base extension che esse vanno a formare (1, . . . , 7). La base extension 2 è composta dalle classi A e B, ma il join (intersezione, evidenziata in grigio) tra queste due classi produce oltre alle entità della b_2[1] (in grigio chiaro) anche una rappresentazione parziale

[1] In questo caso le base extension vengono indicate con la "*b*" minuscola per non creare confusione con i nomi delle classi locali.

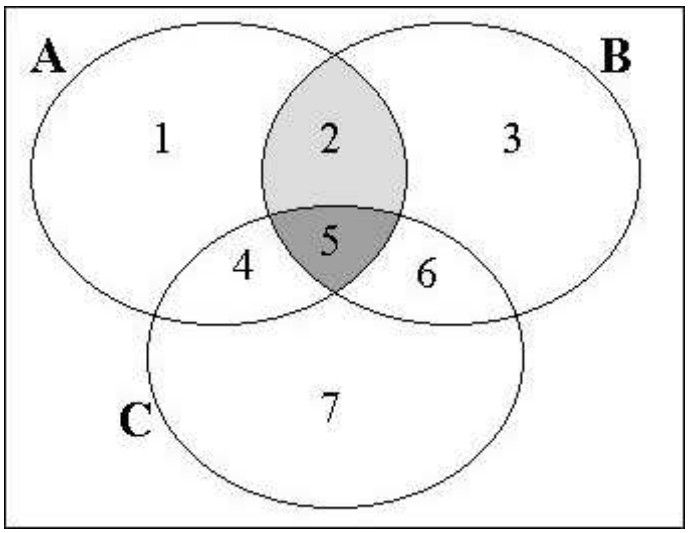

Figura 5.3: Esempio di dominazione tra Base Extension

delle entità della base extension 5 (in grigio scuro), parziale perchè mancano alcuni attributi: quelli di C. La b_5 è data da: $A \bigcap B \bigcap C$, quindi $b_2 = (A \bigcap B) - b_5$. Ricostruendo b_2 dunque inevitabilmente si ricostruisce anche una parte di b_5, quella limitata agli attributi di A e B. La definizione su riportata ci dice che, rispetto a questo insieme di attributi, b_2 domina b_5.

Quindi si possono, anzi si devono, non prendere in considerazione le base extension dominate, ma si devono ricostruire solo quelle dominanti per evitare duplicazioni.
Di conseguenza sfruttando la definizione precedente, si ottiene l'insieme di base extension dominate:

$$BE_{dom} = \{B_2 \in BE_{in} \mid \exists B_1 \in BE_{in} \wedge B_1 \text{dom } B_2 \text{ rispetto a } AG_Q\}$$

Con questo primo passo di semplificazione si ottiene l'insieme ottimo di base extension:

$$BE_{opt} = BE_{in} - BE_{dom}$$

Per la query di esempio `Q_1`, l'insieme di base extension iniziali $BE_{in} = \{4, 5, 6, 8, 10, 12\}$ viene semplificato in $BE_{opt} = \{6, 8\}$, infatti, rispetto all'insieme di attributi della query, si vede che la 6 domina sia la 4 che la 5, mentre la 8 domina l'insieme $\{4, 5, 10, 11\}$. Devono quindi essere ricostruite solo le base extension 6 e 8.

Scartando le base extension dominate da altre abbiamo notevolmente limitato le possibilità di duplicazioni, ma non le abbiamo eliminate del tutto. Nel caso infatti che due base extension ne dominino una terza, questa viene scartata, ma le altre due una volta ricostruite generano insiemi di entità parzialmente sovrapposti: la sovrapposizione è rappresentata proprio dall'estensione della base extension eliminata. Per ovviare a ciò è necessario tenere traccia, nel query plan, di come unire le base extension per una determinata Basic Query: di default bisognerà effettuare una unione delle entità, ma se si presenta un caso come quello appena discusso bisognerà effettuare un outer join tra le base extension al fine di presentare una sola volta la porzione comune. Nel nostro esempio `Q_1` entrambe le base extension selezionate dominano sia la 4 che la 5, di conseguenza in fase di esecuzione devono essere fuse tramite un outer join , come verrà approfondito in 5.3.2.

- **Eliminazione di classi locali**:
  una volta individuato l'insieme ottimo di base extension, ognuna di queste viene esaminata per ottenere il numero minimo di classi a cui accedere. Questa semplificazione del Query Plan consente di diminuire il numero di Local Query da generare e potrebbe anche evitare l'accesso ad una sorgente.

  In una base extension potrebbe accadere che due classi locali siano estensionalmente equivalenti, se questo accade e nessuna delle due classi aggiunge informazioni rispetto all'altra, puo' essere eliminata una delle due in modo indifferente. Se invece una classe contiene attributi della query non presenti nell'altra deve essere eliminata la seconda.

  Un altro caso in cui è possibile minimizzare l'insieme di classi locali, si verifica quando due classi appartengono allo stesso database ed esiste fra loro una relazione di specializzazione: se la sorgente è 'object' o se in ogni caso la classe specializzata contiene tutte le proprietà richieste, è possibile scartare la superclasse.

  Nell'esempio introdotto, la base extension 8 prevede l'interrogazione delle seguenti classi locali $CL_8 = \{$ `University_Worker,CS_Person` $\}$, come si puo' vedere questo insieme non può essere ulteriormente ridotto in quanto non ricorre uno dei due casi visti in precedenza. La base extension 6 prevede inizialmente $CL_6 = \{$ `University_Worker, School_Member, University_Student` $\}$, in questo caso `School_Member` e `University_Student` risultano estensionalmente equivalenti e nessuna delle due aggiunge informazioni

rispetto all'altra, quindi una delle due può essere eliminata indifferentemente, ma eliminanado `University_Student` si evita di accedere alla terza sorgente, `TAX_POSITION`. L'insieme ottimo di classi a cui inviare le Local Query risulta essere: CL_{opt} = {`University_Worker, CS_Person, School_Member`}.

Prima di eseguire la fase di semplificazione delle classi locali, devono essere eventualmente aggiunti alla `<select-list>` della query, gli attributi aggiuntivi per la fusione delle base extension. Come verrà descritto nel dettaglio in 5.3.2, per due base extension da fondere in outer join, è necessario individuare una chiave comune. Per questo è necessario considerare, per determinare l'insieme ottimo delle classi locali, non solo gli attributi originari della query, ma anche eventuali attributi aggiuntivi necessari alla fusione delle base extension.

5.2.3 Generazione delle Local Query

Conclusa la fase di individuazione delle classi locali, il Query Manager ha a disposizione l'insieme minimo di classi da interrogare, gli attributi della Basic Query ed eventualmente gli attributi aggiunti per ricostruire le base extension. Questi elementi costituiscono l'input per l'ultimo passo della fase di *Definizione del Query Plan* che consiste nella generazione delle Local Query da inviare alle varie sorgenti. Con l'ausilio della Mapping Table, contenente tutte le informazioni necessarie per risolvere i conflitti intensionali, la Basic Query, posta sulla schema globale, viene *''riscritta''* in funzione degli schemi locali.

Grazie al modello comune di dati ODL$_{I^3}$, con il quale il Mediatore comunica con i Wrapper, il Query Manager non si deve preoccupare dei linguaggi e dei formalismi utilizzati dalle singole sorgenti, ma saranno i Wrapper stessi a tradurre la query, formulata in OQL, in un linguaggio comprensibile alla sorgente. Prima di effettuare la traduzione vera e propria in funzione degli schemi locali, il Query Manager si occupa della scomposizione della clausola where e dell'individuazione di eventuali attributi aggiuntivi necessari per la successiva fase di *Esecuzione della Query*.

Introduciamo la Basic Query `Q_2`, che ci servirà da esempio per descrivere le diverse fasi della generazione delle Local Query:

```
Q_2: select name, email
     from University_Person
     where faculty = 'cs'
     and rank = 'professor'
     and ( pay > 10000 or title = 'full professor')
```

- **Scomposizione della clausola where e generazione della Basic Query Assembler**
  La clausola WHERE di uno statement OQL è una espressione booleana composta da attributi, tipi base, operatori logici e relazionali, che deve essere soddisfatta dalle tuple di una relazione. Conclusa la fase di normalizzazione descritta in 5.1.3, la clausola è posta in forma normale congiuntiva.

  Ai fini della traduzione in funzione delle sorgenti locali, una *congiunzione* puo' essere scomposta ed i suoi due termini analizzati in modo indipendente, mentre una *disgiunzione* deve essere analizzata nel suo complesso.
  Consideriamo la query `Q_2` riguardante la classe globale `University_Person`: la fase di *Definizione del Query Plan* individua una sola base extension da ricostruire $BE_{opt} = \{12\}$, e l'insieme ottimo di classi locali da interrogare [2]:

  - `Research_Staff(name, dept, pay, rank, email, relation, takes)`
  - `Professor(name, dept, faculty, rank, title)`

  In questo caso il predicato di selezione:

  ```
  (pay > 10000 or title = 'full professor')
  ```

  non è compreso interamente in nessuna delle due classi locali, la scomposizione della clausola nei suoi due fattori porterebbe all'ottenimento di una risultato incompleto e deve quindi essere valutata nel suo insieme.
  Nel caso in cui la Basic Query precedente avesse avuto la seguente clausola in *and*:

  ```
  (pay > 10000 and title = 'full professor')
  ```

  si sarebbe potuta scomporre per valutare i suoi due termini in modo indipendente. Le due Local Query ottenute sarebbero[3]:

[2] Vengono indicati, per ogni classe locale, gli attributi globali con mapping non nullo.

[3] Allo stato attuale il Query Manager non si occupa della gestione dei predicati contenenti valori di default, che vengono semplicemente tradotti col valore predefinito. Nelle rappresentazioni successive questi predicati non vengono riportati se hanno valore *true*, ad esempio le query `Q_21` e `Q_22` conterrebbero rispettivamente `('professor' = 'professor')` e `('cs' ='cs')` che non sono riportati.

```
Q_21: select first_ name, last_name, e_mail
      from Research_Staff
      where pay > 10000

Q_22: select name
      from Professor
      where rank = 'professor'
      and title = 'full professor'
```

Il Query Manager ha a disposizione la clausola where in forma normale congiuntiva, rappresentabile da un albero di predicati che evidenzia i fattori booleani da considerarsi in modo indipendente in fase di traduzione.

Nella figura 5.4 è rappresentato l'albero associato alla query di esempio `Q_2`; viene evidenziato il modo in cui il Query Manager analizza la clausola ricorsiva e i predicati che vengono valutati nel loro complesso. Ognuno di

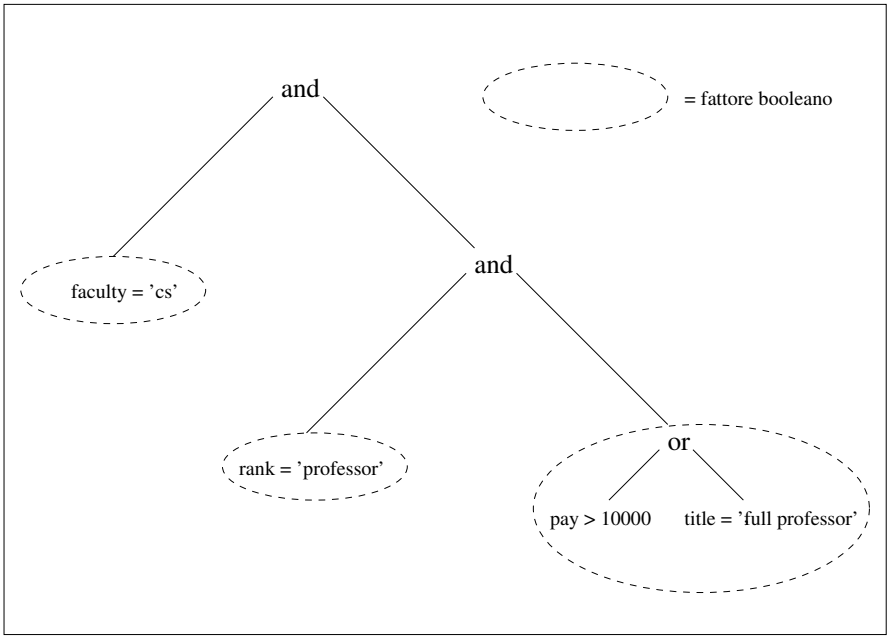

Figura 5.4: Albero dei predicati di selezione

questi fattori viene inserito nei *predicati da tradurre*, se gli attributi globali in esso contenuti hanno un mapping non nullo in almeno una delle classi locali considerate.

È necessario però considerare anche quei fattori che rimangono esclusi in fase di generazione delle query locali, quelli cioè che non sono mappati

interamente da nessuna classe. Per questo motivo ad ogni Basic Query viene associata una struttura detta *Basic Query Assembler* che contiene all'interno della propria clausola where i predicati "esclusi". Il Query Manager, dopo aver concluso la fase di *Esecuzione della Query* ed aver quindi integrato i risultati provenienti dalle sorgenti locali, dovrà eseguire su di essi la Basic Query Assembler per ottenere la risposta desiderata.

Per poter svolgere correttamente questa query finale, devono essere recuperati dalle sorgenti locali anche gli attributi presenti nella clausola where della Query Assembler, aggiungendoli a quelli della Basic Query originale.

Valutando la query precedente, si nota che l'ultimo fattore booleano non è interamente mappato da nessuna delle due classi locali, la Basic Query Assembler diventa quindi:

```
Q_2A: select name, email
      from University_Person
      where ( pay > 10000 or title = 'full professor')
```

Per poter eseguire questa interrogazione sui risultati finali, la Basic Query viene completata aggiungendo agli attributi di selezione **pay** e **title**, dando origine a `Q_2'`:

```
Q_2': select name, email, pay, title
      from University_Person
      where faculty = 'cs'
      and rank = 'professor'
      and ( pay > 10000 or title = 'full professor')
```

Nella Tabella 5.1 è rappresentato schematicamente l'algoritmo che implementa i passi appena descritti a livello teorico, in cui con $AG(P_i)$ è indicato l'insieme di attributi globali contenuti nel predicato P_i, mentre $AG(L_j)$ rappresenta l'insieme di attributi globali con mapping non nullo sulla classe locale considerata, vale a dire: $AG(L_j) = \{ag \in A_G(G) \mid MT[L_j][ag] \text{ è un mapping non nullo}\}$.

- **Traduzione da Basic Query a Local Query**
  La fase di traduzione riguarda gli attributi di proiezione e i predicati di selezione della Basic Query.

  Attributi di proiezione
  Il Query Manager analizza gli attributi presenti nella <select-clause> della Basic Query e quelli aggiunti per la fusione delle base extension, sostituendoli con i corrispondenti attributi

♢ *input*: <where-clause>$= P_1 \wedge P2 \wedge \ldots \wedge P_m$, con $P_j = f_1(A_G(G))$;
MT Mapping Table;
$CL_{opt} = \{L_1, L_2, \cdots, L_k\}$ insieme ottimo di classi da interrogare;
<select-clause> della Basic Query iniziale;
♢ *output*: <where-clause>$_{ass} = f_2(A_G(G))$ clausola where della Basic Query Assembler;
$P = f_3(A_G(G))$ insieme dei predicati da tradurre.

```
    begin
<where-clause>_ass := true;
    P := {};
       for i := 1 to m do begin
           ok:= false;
            for j := 1 to k do
               if (AG(P_i) ⊆ AG(L_j))
               then
               begin
                   ok := true;
                   break;
               end
            if ( ok= true )
            then P := P ⋃ {P_i};
            else
              begin
                  <where-clause>_ass =<where-clause>_ass ∧ P_i;
                  <select-clause>=<select-clause> ⋃ AG(P_i);
              end
        end
    end
```

Tabella 5.1: Generazione della Basic Query Assembler

originali della classe locale. Come visto in 2.2.1, la Mapping Table contiene le informazioni riguardanti il tipo di mapping fra attributo globale e attributo locale, informazioni definite in fase di integrazione degli schemi dal Global Schema Builder.

Per ognuna delle tipologie di mapping sono definite le apposite re-

gole di traduzione che consentono di riscrivere gli attributi presenti nella clausola di selezione della Basic Query in funzione degli schemi locali.

È necessario per ogni elemento tradotto, tener traccia dell'attributo globale originale, con il rispettivo mapping; queste informazioni vengono inserite nel *plan* per essere utilizzate nella fase di esecuzione delle Local Query descritta in 5.3.1.

Quando si incontra un *null mapping* o *default mapping* non viene eseguita nessuna traduzione, ma nell'ultimo caso viene comunque registrata l'informazione nel piano per consentire la corretta ricostruzione della risposta.

I metodi relativi alla *union mapping* non sono ancora stati implementati, saranno elementi necessari il vettore contenente gli attributi tra cui effettuare la scelta e una condizione che permetta di discriminare fra le diverse alternative.

Il Query Manager deve verificare la possibilità di ricostruire ogni base extension partendo dai risultati delle classi locali interrogate, per questo motivo devono essere eventualmente aggiunti alla `<select-clause>` della Local Query gli attributi mancanti, necessari per effettuare i join durante la fase di esecuzione delle Basic Query, come descritto in 5.3.2.

Predicati di selezione

Per ogni classe locale appartenente all'insieme ottimo di classi da interrogare, vengono analizzati i *predicati* individuati in precedenza.

Ognuno di questi fattori viene riportato nella Local Query se tutti gli attributi globali in esso contenuti hanno un mapping non nullo nella classe locale considerata. L'algoritmo riportato nella Tabella 5.2 rappresenta, in modo schematico, questo processo di traduzione per una generica Basic Query posta sulla classe globale G in funzione di ogni classe locale definita in 5.2.2 ($\forall L \in CL_{opt}$). `<where-clause>`$_L$ rappresenta la clausola where della generica classe locale L, P_j è il predicato espresso in termini globali, mentre P'_j è la sua traduzione ottenuta sfruttando le opportune regole definite in funzione della tipologia di mapping riscontrato.

Devono quindi essere definiti metodi di traduzione oltre che per i singoli attributi anche per i predicati di confronto nel loro complesso.

Per quanto riguarda la query di esempio `Q_2`, le due query locali generate in fase di traduzione sono le seguenti:

$\diamondsuit$*input*: $\{P_1, P2, \ldots, P_n\}$, con $P_j = f_1(A_G(G))$
$\diamondsuit$*output*: <where-clause>$_L = f_2(A_L(L))$

```
    begin
<where-clause>_L := true;
for j := 1 to n do
 if (AG(P_j) ⊆ AG(L))
 then <where-clause>_L := <where-clause>_L ∧ P'_j;
    end
```

Tabella 5.2: Algoritmo di traduzione della clausola where

```
Q_21:  select first_name, last_name, e_mail, pay
       from Research_Staff

Q_22:  select name, title
       from Professor
       and rank = 'professor'
```

5.2.4 Semplificazioni del piano di accesso

Un miglioramento al piano di accesso si può ottenere analizzando eventuali predicati di selezione della Basic Query, contenenti valori di default.
Se ad esempio fosse formulata un'interrogazione contenente la clausola `where faculty = 'biology'`, prima di affrontare la fase di traduzione delle query locali, si potrebbero eliminare dal piano tutte le base extension che coinvolgono le classi locali in cui è predefinito un valore di *faculty* diverso, riducendo così il numero di query da inviare alle sorgenti.
Nel nostro esempio intodotto in 2.1.1 sarebbe possibile eliminare immediatamente le classi locali della sorgente `COMPUTER_SCIENCE`, in cui è predefinito il valore di default `faculty = 'cs'`.

Un'altra semplificazione del piano di accesso è ottenibile nel caso in cui sia possibile raggruppare più Local Query da inviare ad una stessa sorgente, effettuando localmente i join. Per applicare questa semplificazione è necessario che i risultati provenienti dalle classi locali coinvolte siano utilizzati per la ricostruzione delle stesse base extension. Ad esempio, considerando la query seguente:

```
Q_3:  select name, year
      from University_Person
```

```
where faculty = 'CS'
and pay > 30000
```

La fase di *Definizione del Query Plan* individua la classe virtuale C8, l'insieme finale $BE_{opt} = \{6\}$ e quindi le classi locali `University_Worker` e `School_Member`[4]. Queste classi appartengono alla stessa sorgente `UNIVERSITY` ed inoltre vengono utilizzate entrambe per la ricostruzione della base extension 6. Si può quindi migliorare il plan raggruppando le Local Query `Q_31` e `Q_32`, generate dal normale processo di traduzione, nell'unica query `Q_31'`:

```
Q_31:  select firt_name, last_name
       from University_Worker
       where pay > 30000

Q_32:  select firt_name, last_name, year
       from School_Member
       where faculty = 'CS'

Q_31':  select a.first_name, a.last_name, b.year
        from University_Worker as a
             School_Member as b
        where a.first_name = b.first_name
        and a.last_name = b.last_name
        and a.pay > 30000
        and b.faculty = 'CS'
```

Al momento queste due semplificazioni al piano di accesso sono due ipotesi di sviluppi futuri.

5.2.5 Ottimizzazione semantica locale

Un'altra tipologia di semplificazione, diversa da quella vista in fase di traduzione da Basic Query a Local Query, e' l'*ottimizzazione semantica locale*. Come per l'*ottimizzazione semantica globale* vista in 5.1.2, vengono sfruttate, tramite l'impiego di ODB_Tools, le regole di integrità che in questo caso sono definite sulle singole sorgenti.
Questa fase di ottimizzazione è posta a livello Mediatore, in quanto non tutte le sorgenti potrebbero essere in grado di realizzarla.

[4]Difatti le classi `School_Member` e `University_Student` risultano estensionalmente equivalenti e quindi quest'ultima va scartata.

L'ottimizzazione semantica consente di creare query meno onerose, cioè query la cui esecuzione su di un database locale è migliorata da:

- aumenta la possibilità di utilizzare degli indici:

  aggiungendo un predicato implicato da una rule si può introdurre nella query un attributo che potrebbe essere indicizzato;

- consente di eliminare o modificare dei join impliciti:

  una rule consente di riconoscere se due condizioni sono ridondanti: in questo caso, se quella implicata comporta l'esecuzione di un join, viene eliminata;

- permette di evitare o ridurre l'accesso a dati inutili:

  questo caso è generato da una query che possa essere trasformata in una equivalente su di una sottoclasse;

- può determinare l'accesso a dati senza necessità di valutazione di predicati:

  questa possibilità è realizzata qualora si riconosca che una query sussuma una intera classe dello schema.

Ad esempio supponiamo che sulla classe locale `University_Student`, sia definito un vincolo di integrità espresso dalla suguente rule ODL_{I^3}:

```
rule RG2: forall x in University_Student : x.faculty = 'cs'
            then x.tax_fee > 12000
```

porta all'espansione della query `Q` in `Q'`:

```
Q: select name
   from University_Student
   where faculty_name = 'cs'

Q': select name
    from University_Student
    where faculty_name = 'cs'
    and tax_fee > 12000
```

L'aggiunta di questo predicato può essere conveniente purchè sia presente nella sorgente un indice sull'attributo `tax_fee`. L'espansione ottenuta porta ad un aumento del costo di analisi dell'interrogazione, ma risultati sperimentali [45] hanno dimostrato che il costo complessivo di esecuzione della query ottimizzata decresce rapidamente all'aumentare del numero di istanze nel database e, mediamente, all'aumentare del numero di query fatte.

5.3 Esecuzione della query

L'ultimo passo del *Query Processing* è l'*Esecuzione della Query*: il Query Manager utilizza le informazioni del *plan*, generate in precedenza, per ricomporre i risultati ottenuti dalle sottoquery, presentando all'utente una risposta integrata.
Questa fase si svolge a tre livelli differenti, rappresentati in Figura 5.5: inizialmente vengono eseguite le Local Query, successivamente le Basic Query ed infine la ricomposizione della Global Query.

Con l'ausilio di un DBMS, il Query Manager crea delle tabelle in grado di memorizzare i risultati temporanei degli stadi intermedi della ricostruzione della risposta. In questo modo il Query Manager potrà eseguire le varie operazioni di ricostruzione e fusione attraverso query (join e outer join) poste su queste tabelle temporanee.
Poichè il database "interno" sarà a disposizione, in uno stesso istante, di varie istanze del Query Manager, ognuna delle quali eseguirà delle Global Query, per limitare i problemi dovuti allo spazio disponibile sul database, ogni tabella viene immediatamente eliminata non appena i dati temporanei in essa contenuti sono stati elaborati.

5.3.1 Esecuzione delle Local Query

Le Local Query generate in 5.2.3 sono espresse in OQL, sarà poi compito del Wrapper tradurre la sintassi OQL in un linguaggio di interrogazione comprensibile alla relativa sorgente.

Le Local Query vengono inviate parallelamente ai Wrapper che le eseguono e restituiscono i risultati momentaneamente memorizzati in relazioni temporanee. Il Query Manager accede a questi dati e, sfruttando le informazioni contenute nel *plan*, deve restituirne una rappresentazione globale, traducendo i valori locali con un processo inverso rispetto a quello visto in precedenza. In questo momento vengono inseriti eventuali valori di default, non presenti all'interno dei risultati locali, ma di cui si era tenuto traccia nel piano. Il Query Manager crea quindi, per ogni Local Query, la corrispondente tabella sul database interno, in cui inserisce tutti i valori ottenuti dall'elaborazione precedente, valori che sono espressi in

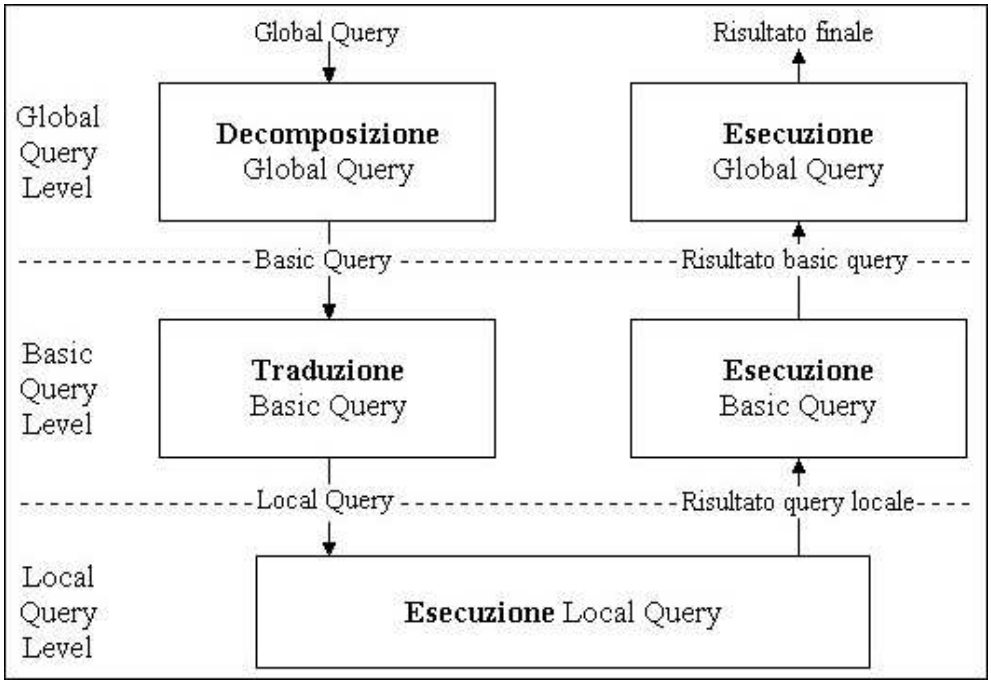

Figura 5.5: I diversi livelli di query processing

termini globali.

Abbiamo visto che per `Q_1`, intodotta in precedenza, le classi locali da interrogare sono:

- `University_Worker`
- `School_Member`
- `CS_Person`

Le Local Query generate dal processo di traduzione descritto in 5.2.3 sono[5]:

```
Q_11: select first_name, last_name, dept_code
      from Universty_Worker
      where  pay > 20000
```

[5]Le prime due classi locali appartengono alla stessa sorgente `UNIVERSITY`, ma in questo caso non si può effettuare la semplificazione vista in 5.2.4 in quanto i risultati della Local Query `Q_11` vengono utilizzati per la ricostruzione, oltre che della base extension 6 insieme a `Q_12`, anche della base extension 8.

```
Q_12: select first_name, last_name, dept_code
      from School_Member
      where  faculty = 'cs'


Q_13: select name
      from CS_Person
```

Il Query Manager esegue, tramite i Wrapper, le tre Local Query e crea sul database le tre tabelle corrispondenti[6]:

- TABLE University_Worker (name, dept)
- TABLE School_Member (name, dept)
- TABLE CS_Person (name)

All'interno di queste tabelle vengono inseriti i risultati restituiti dalle sorgenti, ma espressi in termini globali, quindi ad esempio la tupla `('Maria', 'Rossi', 'd1')`, restituita dalla classe locale `University_Worker`, viene inserita nella tabella corrispondente nella seguente forma: `('Maria Rossi', 'd1')`.

5.3.2 Esecuzione delle Basic Query

Come è evidenziato in figura 5.6, l'esecuzione della Basic Query è composta da tre passi temporalmente successivi:

1. **Ricostruzione di ogni base extension**
   Questo passo viene effettuato attraverso l'unione dei risultati provenienti dalle classi locali. Vengono utilizzate le *regole di join*, che stabiliscono per ogni coppia di classi locali gli attributi su cui effettuare il join. Questi attributi identificano in modo univoco gli elementi all'interno di entrambe le classi e costituiscono quindi una chiave comune utilizzabile per fondere i risultati provenienti dalle classi locali.
   Durante la fase di *Definizione del Query Plan*, vengono inserite queste informazioni nel piano di accesso e vengono eventualmente aggiunti alla clausola di selezione delle Local Query gli attributi necessari per la fusione di tutte le classi della base extension.

[6] In realtà, come si vedrà nel Capitolo 3, il nome delle tabelle è costruito aggiungendo al nome della classe locale un prefisso che rappresenta l'iteratore univoco associato alla Global Query.

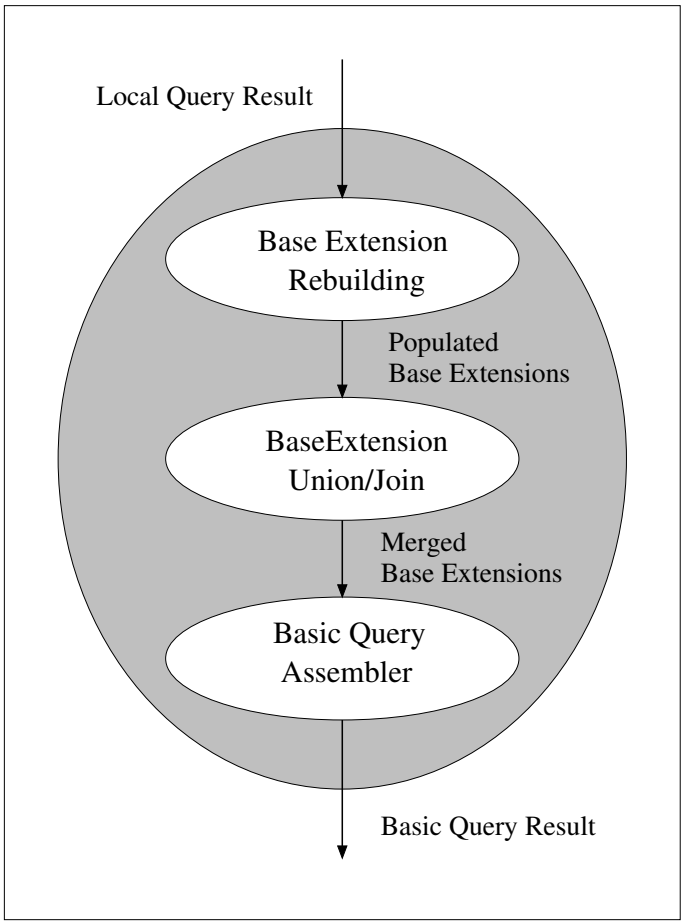

Figura 5.6: Esecuzione delle Basic Query

A livello pratico il Query Manager genera, per ogni base extension da ricostruire, una query che riporta i join da effettuare fra le tabelle contenenti i risultati dell'esecuzione delle Local Query e pone i risultati ottenti in altre tabelle temporanee.
Nella nostra query di esempio `Q_1` l'insieme di base extension ottimo da interrogare è $BE_{opt} = \{6, 8\}$, supponendo che l'attributo globale *name* costituisca la chiave semantica comune alle classi coinvolte, le regole di join individuate nel piano sono le seguenti:

- `R1 (University_Worker,School_Member)` $\Rightarrow k =$ `name`
- `R2 (University_Worker,CS_Person)` $\Rightarrow k =$ `name`

Quindi le query generate dal Query Manager sono:

```
B6: select a.name, a.dept
    from University_Worker as a,
         School_Member as b
    where a.name = b.name

B8: select a.name, a.dept
    from University_Worker as a,
         CS_Person as b
    where a.name = b.name
```

Il Query Manager, dopo aver eseguito queste due query, crea le due tabelle corrispondenti in cui inserire i dati ottenuti.

2. **Fusione delle base extension**
   Si sono quindi ottenute base extension popolate che a questo punto devono essere fuse per ricostruire il risultato della Basic Query. I criteri di fusione sono stati definiti durante la fase di *Definizione del Query Plan*: per ogni coppia di base extension dell'insieme ottimo viene indicato, all'interno del piano, il tipo di fusione da effettuare. Di default viene effettuata l'*unione*, ma nel caso in cui le due base extension ne dominino una terza, è necessario un *outer join* per evitare duplicazioni. In figura 5.7 viene rappresentato un caso di *outer join*: le base extension iniziali sono: $BE_{in} = \{4, 5, 6\}$. Durante la fase di semplificazione, la base extension 5 viene scartata in quanto è dominata sia dalla 4 che dalla 6. Questo però non è sufficiente per eliminare completamente le duplicazioni, infatti la ricostruzione delle base extension 4 e 6 genera insiemi di entità parzialmente sovrapposti e la sovrapposizione è rappresentata proprio dall'estensione della base extension eliminata. In questo caso quindi è necessario un *outer join* fra i risultati ottenuti dalla restituzione delle due base extension, ma per fare ciò bisogna poter identificare gli elementi comuni, quelli cioè appartenenti alla base extension 5.

   Una proposta di risoluzione del problema è basata, anche in questo caso, sull'uso delle *regole di join*, che individuano per ogni coppia di classi locali una chiave comune, utilizzabile, oltre che per la ricostruzione della singola base extension anche per effettuare eventuali outer join.
   È possibile fondere due base extension se si determina una regola di join che indica un identificatore comune ad una classe locale di una base extension e ad una classe locale dell'altra, o eventualmente usando la chiave semantica di una classe comune ad entrambe.

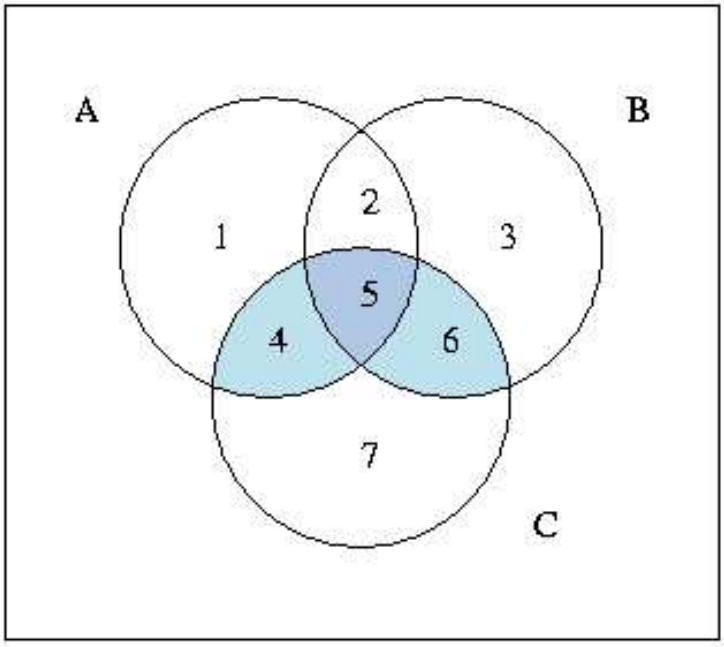

Figura 5.7: Fusione tramite outer join

Nell'esempio riportato in 5.7, il risultato della base extension 4 è fornito dalla fusione delle classi A e C, mentre la base extension 6 è costituita da B e C. Supponiamo che esistano le seguenti possibili regole di join:

- R1 (classe A, classe B) ⇒ joineable = false
- R2 (classe A, classe C) ⇒ k1
- R3 (classe C, classe B) ⇒ k2

Inoltre avendo la classe C in comune :

- (classe C) ⇒ k1,k2

Il Query Manager analizza quelle che in definitiva rappresentano le chiavi alternative per le due base extension, con lo scopo di trovare un possibile identificatore comune con cui effettuare l'outer join.

Come già visto in 5.2.2, questa analisi deve essere effettuata prima della *semplificazione dell'insieme di classi locali*, infatti, nel caso in cui fra

le possibili chiavi, non ne esista una già compresa fra gli attributi della query, bisogna considerare anche gli attributi aggiuntivi prima di procedere all'eliminazione di classi superflue.

Per la nostra query di esempio le due base extension 6 e 8 dominavano entrambe sia la 4 che la 5, di conseguenza una semplice unione dei loro risultati genererebbe insiemi di entità parzialmente sovrapposti, quindi il Query Manager deve creare una query che esegua l'*outer join* fra i risultati della ricostruzione delle due base extension, contenuti nelle due tabelle temporanee B6 e B7.
Il piano di accesso indica l'attributo *name*, come chiave utilizzabile per la fusione, in quanto la classe `University_Worker` appartiene ad entrambe le base extension e la sua chiave, *name* appunto, è compresa fra gli attributi della query.
Si ottiene quindi:

```
Q_1outer: select B6.name, B6.dept
          from B6 full join  B8 on (B6.name = B8.name)
```

In tutti i casi in cui non si ha dominazione di due base extension su di una terza, si può procedere all'*unione* dei risultati senza rischi di ridondanze, avendo eliminato le base extension dominate.

3. **Basic Query Assembler**
   L'ultimo passo che rimane per l'ottenimento del risultato della Basic Query è l'esecuzione della Basic Query Assembler, contenente i predicati non verificati dalle Local Query e la `<select-clause>` della query originaria. Infatti durante la fase di *Definizione del Query Plan*, la query è stata aggiornata aggiungendo eventuali attributi necessari per la fusione delle base extension, per la valutazione dei predicati "esclusi" ed inoltre i risultati delle Local Query possono contenere campi aggiunti per effettuare la ricostruzione delle base extension. Con l'esecuzione della Basic Query Assembler si ottiene quindi il risultato finale della Basic Query, comprendente sia la valutazione completa dei predicati, sia il reperimento delle informazioni effettivamente richieste inizialmente.

   Vediamo la fase di esecuzione della query `Q_2`:
   in fase di *Definizione del Query Plan* viene selezionata la sola base extension 1, quindi il Query Manager genera la query seguente per effettuare

la ricostruzione dei risultati ottenuti dalle due Local Query `Q_21, Q_22` descritte in 5.2.3:

```
B1: select a.name, a.email, a.pay, b.title
    from Research_Staff as a
         Professor as b
    where a.name = b.name
```

La Basic Query Assembler `Q_2A`, riportata in 5.2.3, restituisce il risultato finale della Basic Query:

```
Q_2A: select name, email
      from B1
      where ( pay > 10000 or title = 'full professor')
```

5.3.3 Esecuzione della Global Query

L'esecuzione di una Global Query consiste nell'esecuzione della *Global Query Assembler*, introdotta in 5.2.1, sui risultati ottenuti dall'esecuzione delle Basic Query coinvolte.
Oltre ai join fra le classi globali, vengono eseguite tutte le operazioni complesse che, dovendo agire su insiemi di informazioni già integrate, non sono trattabili a livello Basic Query.

Capitolo 6

Stato dell'arte

MOMIS, nonostante l'integrazione di informazioni sia un campo di ricerca relativamente nuovo, non è l'unico sistema che cerca di realizzare un modulo integratore. Tra i sistemi mediatori più conosciuti si possono citare GARLIC [15, 16], SIMS [17, 18], HERMES [13], TSIMMIS [5],ecc.
In particolare quest'ultimo sviluppa in maniera approfondita ed interessante le problematiche, riguardo all'Object Fusion [7, 46], al centro della trattazione di questa tesi. Per questo motivo, in questo capitolo, si esporrà come TSIMMIS realizza il processo di fusione e lo si metterà a confronto con l'approccio adottato da MOMIS.
Inoltre, nel Capitolo 2, si è vista l'importanza del concetto di *omogeneità semantica* tra i campi chiave, nel processo che porta al riconoscimento delle istanze facenti riferimento alla medesima entità del mondo reale.
Le grandi organizzazioni necessitano di scambiare ed integrare informazioni, tra molti sistemi separati tra di loro. Perchè questa attività produca dei risultati utili e corretti, è fondamentale che vi sia accordo sui significati dei dati gestiti. Insomma, quello che bisogna garantire è che vi sia *interoperabilità semantica*. In questo capitolo verranno esposti alcuni studi riguardo questa problematica, e si cercherà di capire come questi possano influenzare gli sviluppi futuri del sistema MOMIS.

6.1 TSIMMIS

TSIMMIS (The Stanford_IBM Manager of Multiple Information Sources), si pone come obiettivo lo sviluppo di strumenti che facilitino la rapida integrazione di sorgenti testuali eterogenee, includendo sia sorgenti di dati strutturati che **semistrutturati**. Questo obiettivo è raggiunto attraverso un'architettura comune a molti altri sistemi (ed a MOMIS in particolare): i *Wrapper* convertono i dati in un modello comune, mentre i *Mediator* combinano ed integrano i dati ricevuti dai Wrapper.

La peculiarità di TSIMMIS si manifesta nel modello comune dei dati utilizzato per rappresentare le informazioni. OEM (Object Exchange Model) è un modello ad *etichette* basato sui concetti di identità di oggetto e di annidamento, particolarmente adatti a descrivere dati la cui struttura non è nota o è variabile nel tempo. Usando OEM, non è necessario che oggetti che si descrivono tramite la stessa etichetta abbiano pure lo stesso schema. In aggiunta a questo modello sono disponibili ed utilizzati dal sistema due linguaggi di interrogazione OEM-QL e MSL (Mediator Specification Language), per ottenere i dati dalle fonti ed integrarli opportunamente. In particolare quest'ultimo linguaggio viene sfruttato per definire in modo dichiarativo i mediatori: attraverso l'uso di *rules* si può specificare il punto di vista del mediatore ed in questo modo definire lo "schema globale" in modo manuale. Ogni rule è costituita da una *testa*, che definisce la struttura che l'oggetto dovrà avere una volta estratto e ricostruito; e da una *coda*, che descrive dove andare a recuperare l'oggetto che si vuole ricevere.
Per un'analisi più approfondita dell'architettura di TSIMMIS e dei linguaggi da esso utilizzati si rimanda alla bibliografia. Quello che interessa analizzare in questa sezione è come TSIMMIS affronti le problematiche inerenti al processo di Object Fusion.

6.1.1 Object Fusion basata su oid semantici

In TSIMMIS il processo di fusione delle istanze è basato sull'uso di *oid* semantici: il mediatore è specificato da una serie di rule logiche e non procedurali, che permettono di mappare oggetti relativi a determinate entità del mondo reale in determinate sorgenti, in un oggetto "virtuale" presso il mediatore. A questi oggetti virtuali viene assegnato un *oid* semanticamente significativo, dimodochè oggetti che presso il mediatore hanno lo stesso *oid* sono fusi assieme, perchè riferiti alla medesima entità del mondo reale. Ovviamente questa fusione avviene solo quando arriva una query al mediatore, le cui specifiche possono essere equiparate a delle viste.

Per mostrare come TSIMMIS realizzi l'Object Fusion, si supponga di voler integrare sorgenti che presentano delle informazioni bibliografiche. Mediante OEM un generico oggetto di queste sorgenti (esportato da un wrapper) può essere così rappresentato in un generico mediatore chiamato `simple`:

```
<&r1, report, set, {&r1n, &r1a, &r1t}>
      <&r1n, report_num, string, "AB-123-456">
      <&r1a, author, string, "John Patriot">
      <&r1t, title, string, "UN Conspirancies">
              ...
```

Per facilitare l'operazione di integrazione, quando nel mediatore si rappresentano gli oggetti OEM, si usa un identificatore semanticamente significativo: ad esempio se `report_num` è una *chiave*, può essere usata per riconoscere altri oggetti che modellano la medesima entità del mondo reale. Quindi `&AB-123-456` è l'*oid* semantico che si cercava.
Si supponga che `simple` esporti oggetti che presentano l'etichetta `techreport`: questi oggetti fondono informazioni di natura bibliografica (`report`) che presentano lo stesso `report_num`, e sono nelle generiche sorgenti `s1` e `s2`.
Allora sul mediatore possono essere effettuate tramite MSL le seguenti specifiche/rule, che definiscono la *vista* esportata:

(R1):

```
<trep(RN) thecreport {<title T>}>@simple :-
          <report {<report_num RN> <title T>}>@s1
```

(R2):

```
<trep(RN) thecreport {<postscript P>}>@simple :-
          <report {<report_num RN> <postscript P>}>@s2
```

La rule R1 specifica che se esiste una coppia di valori (*t,r*) per le variabili `T` e `RN`, tali che la sorgente `s1` contiene un oggetto `report` che prevede i sottoggetti `report_num` e `title` che assumono rispettivamente i valori *r* e *t*; allora il mediatore `simple` esporta un oggetto `techreport` con *oid* `trep(`*r*`)`, che ha un sottoggetto `title` con valore *t* ed un unico *oid* generato dal sistema.
Analogamente si possono leggere le specifiche presenti nella rule R2. La funzione `trep()` non è altro che una funzione di skolemizzazione che viene applicata alla chiave semantica comune.
Si può osservare che R1 non impedisce agli oggetti `techreport` con *oid* `trep(`*r*`)` di avere sottoggetti diversi da `title`, quindi permette alla rule R2 di aggiungere più sottoggetti (`postscript` in questo caso) allo stesso oggetto `techreport`. In generale è così che viene realizzata l'Object Fusion in TSIMMIS: MSL fornisce delle rules che permettono di inserire incrementalmente ed indipendentemente informazioni in un oggetto del mediatore, identificato tramite un *oid* semantico.
I passi eseguiti dal Mediatore, nel processo che va dall'acquisizione della query alla generazione degli oggetti che costituiscono la risposta finale, sono descritti nella prossima sezione.

6.1.2 Query Processing

Nella fase di Query Processing viene definito un *datamerge program*, sulla base della query posta dall'utente e delle rule sul mediatore, che descrivono la vista esportata su cui è posta la query. Questo datamerge program consiste in una collezione di rule, la cui coda si riferisce alla struttura degli oggetti presso le rispettive sorgenti, e la cui testa descrive la struttura degli oggetti costituenti la risposta.

Si consideri il mediatore `simple`, descritto nella sezione precedente, che integra informazioni provenienti dalle sorgenti `s1` e `s2`.
Si formuli su `simple` la seguente query, che richiede tutti i sottoggetti degli oggetti `techreport` che prevedono titolo `'abc'`.

(Q1):

```
<X techreport V> :- <X techreport:{<title 'abc'>}@simple
```

La prima cosa da fare è convertire la query Q1 e le rule R1 e R2 sul mediatore, in forma normale MSL:

(Q1a):

```
<X techreport {<Void Vl Vv>}> :- <X report {<T2 title
'abc'>}>@simple AND <X report {<Void Vl Vv>}>@simple
```

(R1a):

```
<trep(RN1) techreport {<T1 title T>}>@simple :- <Ro1 report {<RNo1
rn RN1> <T1 title T>}>@s1
```

(R2a):

```
<trep(RN2) techreport {<Poid postscript P>}>@simple :- <Ro2 report
{<RNo2 rn RN2> <Poid postscript P>}>@s2
```

In particolare, nella query Q1a è stata spezzata la coda, per far sì che ogni insieme $\{\ldots\}$ contenga esattamente un solo oggetto $<\ldots>$.
Inoltre, per quanto riguarda le rule R1a e R2a, vengono rinominate le variabili, in modo che non ci siano due rule con variabili comuni. Questo viene fatto per evitare confusione quando le due rule sono fuse in una singola rule nel datamerge program.
Arrivati a questo punto, si procede all'individuazione del matching tra le condizioni espresse nella coda della query e la testa delle rule. In partcolare una

condizione *c* realizza il matching con una rule *r*, se la rule può produrre oggetti che soddisfano la condizione. Ogni matching che ha successo, produce un *unifier*, che è una struttura dati che descrive il matching tra *c* e *r*. Per ogni unifier, la condizione *c* viene sostituita dalle condizioni sulle sorgenti specificate nella rule *r*.
Quindi, nel nostro caso, si producono i seguenti unifier.

θ_1 = [(R1a): `X→trep(RN1), T1→T2, T→'abc'`]

Siccome la rule R2a realizza due matching, è necessario introdurre una seconda istanza della rule in questione:

(R2a.bis):

```
<trep(RNb) techreport {<T1b title Tb>}>@simple :- <Ro1b report
{<RNo1b rn RNb> <T1b title Tb>}>@s1
```

e quindi gli unifier:

θ_2 = [(R2a): `RN2→RN1, Void→Poid, Vl→postscript, Vv→P`]

θ_3 = [(R2a.bis): `RNb→RN1, Void→T1b, Vl→title, Vv→T1b`]

Di conseguenza, per la query Q1, il datamerge program (DP) che si ottiene è:

(DP1):

```
<trep(RN1) techreport {<Poid postscript P>}:- <Ro1 report {<RNo1
rn RN1> <T2 title 'abc'>}>@s1 AND <Ro2 report {<RNo2 rn RN1> <Poid
postscript P>}>@s2
```

(DP2):

```
<trep(RN1) techreport {<T1b title Tb>}:- <Ro1 report {<RNo1 rn
RN1> <T2 title 'abc'>}>@s1 AND <Ro1b report {<RNo1b rn RN1> <T1b
title Tb>}>@s1
```

Come è evidente, il datamerge program definito, permette di inserire incrementalmente ed indipendentemente informazioni, nel medesimo oggetto nel mediatore, identificato univocamente dall'*oid* semantico `trep(RN1)`.

6.1.3 Considerazioni

Il vantaggio maggiore offerto da questa soluzione risiede nel creare la possibilità di raggruppare/fondere oggetti a prescindere dalla sorgente in cui si trovano, sulla base dell'identificatore prescelto per la definizione dell'*oid* specifico. Si realizza un modello molto vicino ad un'approccio ad oggetti, realizzando una maggiore modularità della soluzione, in quanto l'introduzione di una nuova sorgente determina esclusivamente la scrittura di una nuova rule. Una condizione che può essere vincolante risiede nella necessità che esista una chiave semantica comune a tutte le sorgenti: la sua presenza è indispensabile per poter fondere gli oggetti. Qualora in una sorgente non fosse presente l'attributo che per le altre sorgenti è stato identificato come significativo, si è costretti ad esportare oggetti senza poterli fondere, perchè gli *oid* generati per oggetti provenienti da sorgenti diverse non sono compatibili.
La soluzione al problema dell'Object Fusion che è stata proposta per MOMIS, vuole mantenere sia la capacità di fondere gli oggetti che di creare oggetti complessi. Per assicurare la fusione delle istanze facenti riferimento alla medesima entità del mondo reale si definiscono, per ogni coppia di classi locali di una data classe globale, i campi semanticamente identificativi (le *chiavi* quando dichiarate) sulla base dei quali effettuare il join in fase di *Esecuzione della Basic Query*. Perchè il join sia effettivamente realizzabile è necessario che questi campi risultino semanticamente omogenei; oppure qualora questo non si possa ottenere, il *matching* tra i valori assunti dai campi chiave viene esplicitato dal progettista tramite una tabella, che richiede il passaggio attraverso un join intermedio per realizzare una corretta fusione.
Il vantaggio di questa soluzione consiste nel superare la forte restrizione che in TSIMMIS richiede che tutte le classi aventi estensione parzialmente sovrapposta presentino la medesima chiave semantica.
Il nostro sistema, come dimostrato, è invece in grado di realizzare la fusione in ogni caso, purchè le classi coinvolte presentino a due a due chiavi semanticamente omogenee, o comunque una tabella comune di *matching*.

6.2 Interoperabilità semantica

Un importante ostacolo al raggiungimento di una corretta e completa integrazione delle informazioni è l'*eterogeneità semantica*.
Ad esempio, si consideri il caso di dati sull'altitudine. Il concetto di "altitudine", in un contesto orbitale viene considerato come distanza dal centro della terra, mentre, nel contesto dell'aviazione, definisce la distanza verticale dalla superficie terrestre.

La risoluzione di questi problemi non si può limitare ad un approccio sintattico o strutturale. L'eterogeneità sintattica riguarda differenze nella rappresentazione dei dati (ad esempio, l'altitudine può essere espressa in metri o kilometri). L'eterogeneità strutturale riguarda differenze nella struttura dei dati (ad esempio i siti web possono strutturare contenuti similari in modi diversi).
È evidente che vi è la necessità di una *riconciliazione semantica*, per ottenere il vero significato di questi dati e per evitare errori in fase di integrazione.
In questa sezione verranno esposti alcuni approcci, finalizzati alla risoluzione del problema della riconciliazione semantica.

6.2.1 Riconciliazione semantica

Come primo approccio alla riconciliazione semantica, può essere interessante fare riferimento, come esposto in [47], al concetto di *semantic value*. Alla base di questi studi, vi è la convinzione che le informazioni sul contesto di una dato, devono essere un componente attivo di un dato sistema di informazioni. Per *semantic value* si intende una porzione di dato con associato il corrispondente *contesto*. Con *contesto* di un dato si fa riferimento al metadato che raccoglie le informazioni riguardo il suo significato, le sue proprietà e la sua struttura. La conseguenza immediata di utilizzare esplicitamente il contesto di un dato, è la possibilità di rendersi conto se due dati sintatticamente diversi hanno il medesimo significato. Un esempio di semantic value può essere:

Prezzo = 1.25(Periodicità = 'quadrimestre', Valuta = 'dollaro USA')

In questo caso, il termine *Prezzo*, che assume il valore 1.25, viene definito nel suo significato dalle due proprietà *Periodicità* e *Valuta*, che ne individuano il contesto.
A questo punto, per realizzare l'operazione di riconciliazione semantica, è però necessario avere a disposizione delle *funzioni di conversione*, che permettano di ricondurre due dati al medesimo contesto, e quindi di confrontarli tra di loro (col fine di capire se rappresentano la stessa cosa).
Data una proprietà P, che individua il contesto di un termine, si definisce funzione di conversione per P una funzione $f()$, che converte il valore di P in un contesto, in quello assunto (sempre da P) in un altro contesto. Queste funzioni di conversione (che possono essere definite o dal sistema mediatore o dalle applicazioni che fanno parte del sistema mediatore) necessitano di essere raccolte in librerie, che possono essere o del sistema oppure anche trovarsi on-line. È chiaro quindi, che l'utilizzo di ogni funzione di conversione ha un costo associato: conversioni che implicano la consultazione di librerie on-line avranno un costo maggiore di conversioni che necessitano solo di un calcolo matematico.

Un problema che rimane irrisolto però, è il fatto che non sempre è possibile riconciliare (tramite funzioni di conversione) due termini al medesimo contesto. Ad esempio, non sempre è possibile trovare funzioni di conversione che riportino al medesimo contesto due valute, presenti in due semantic value rappresentanti un prezzo.

Analoghi studi sulla riconciliazione semantica (che sfruttano nuovamente il concetto di contesto) sono stati sviluppati dalla MITRE Corporation per conto del Dipartimento della Difesa degli Stati Uniti, col fine di realizzare un *Integrated Information Space (IIS)* [48].
In questi studi, si è visto come il fatto di associare ad un termine, il contesto che ne definisce il significato, possa risultare, col tempo e con l'aumentare del numero di informazioni da gestire, molto dispendioso. Questo per tre ordini di motivi:

- alcuni contesti sono molto complessi da rappresentare;
- in genere si ha a che fare con grandi numeri di attributi, a cui associare un contesto;
- anche le informazioni sul contesto necessitano di essere mantenute ed aggiornate (si hanno di conseguenza elevati costi di amministrazione).

Si rende evidente la necessità di individuare informazioni semanticamente efficienti, che permettano di descrivere correttamente il contesto di grosse classi di attributi, e di conseguenza di realizzare il processo di riconciliazione semantica. La proposta fatta dalla MITRE Corporation prevede la definizione di tre tipi di informazione, che permettono di inferire il contesto degli attributi (costituenti le informazioni da integrare), rendendo espliciti conflitti semantici che altrimenti rimarrebbero nascosti, e di conseguenza, rendendo possibile la riconciliazione semantica.
Queste informazioni sono:

1. **Source-descriptors**: ad ogni sorgente viene associato un source-descriptor, ovvero un record formato da nove campi, che permettono di delineare il contesto in cui sono inserite le sorgenti da cui provengono le informazioni.
2. **Usage-descriptors**: uno usage-descriptor è un record di sei campi, che permette di individuare il contesto in cui si inseriscono gli obiettivi di una determinata comunità di utenti, che consulta le informazioni integrate.
3. **Canonical attributes**: un canonical attribute, rappresenta una classe di attributi affini e contiene informazioni su come il significato di questi attributi cambi, al cambiare del contesto. Tra i campi che costituiscono un canonical attribute, vi è una libreria di funzioni di trasformazione semantica.

6.2.2 Considerazioni

Come si è visto nelle sezioni precedenti il processo che porta alla formulazione della Join Table di una data classe globale, non è un processo completamente automatico, bensì necessita di un significativo intervento da parte del progettista. Facendo riferimento alla [22], tale intervento può limitarsi ad una validazione dei campi di join, oppure può essere più imponente e consistere nell'iserimento di una tabella contenente il *matching* tra i valori dei campi di join. Questa tabella realizza il concetto di *vista materializzata*.
Siccome i database che MOMIS gestisce sono in continua evoluzione e possono subire notevoli modifiche, è necessario che queste tabelle di esplicitazione del matching vengano tenute continuamente aggiornate, per garantire la correttezza dei risultati delle operazioni di fusione.
La possibilità di stabilire il contesto dei termini da integrare può essere molto importante. È immediato rendersi conto che ciò verrebbe in grosso aiuto al sistema mediatore, qualora questo si trovasse a dover gestire oggetti identificati da chiavi disomogenee o che, comunque, presentano contesti diversi.
Ad esempio, si supponga che due classi presentino istanze sovrapposte identificate da un codice alfanumerico, calcolato secondo criteri diversi. Allo stato attuale per realizzare la fusione viene usata una tabella intermedia. In futuro però, la presenza di una funzione di conversione da un codice ad all'altro renderebbe immediata l'operazione di individuazione delle istanze facenti riferimento alla medesima entità del mondo reale e, di conseguenza, faciliterebbe l'operazione di fusione.

Conclusioni

Come è stato illustrato nei primi due capitoli di questa tesi, l'obiettivo del sistema MOMIS è quello di realizzare un mediatore in grado di attuare l'integrazione di un insieme di sorgenti eterogenee ed autonome, generando una vista globale che l'utente può interrogare senza possedere un'effettiva conoscenza delle diverse sorgenti. Infatti questo compito è lasciato al sistema che formula le interrogazioni in modo automatico per ogni sorgente locale nel linguaggio specifico eseguendo anche le necessarie ottimizzazioni.

Proseguendo il lavoro di ricerca ed implementazione svolta [22, 49, 36] questa tesi si è concentrata sul componente Query Manager. In particolare si è posta come obbiettivo la definizione e la formulazione di algoritmi atti a migliorare i tempi di risposta, la completezza e la correttezza dell'interrogazione. Il Query Manager dopo le fasi di integrazione delle informazioni, utilizzando la rappresentazione della clausola where tramite un albero binario, genera le query locali ed il filtro globale da applicare in fase di Object Fusion. In particolare è stata discussa la definizione ed il calcolo di una particolare risposta chiamata Full Disjunction in due casi diversi: ciclico ed aciclico. Nel primo caso è stato presentato un metodo algebrico di riscrittura delle query per il calcolo della Full Disjunction. Nel secondo caso è stato formulato il procedimento di trattamento della Query Globale tramite strutture dati che non usano le Base Extension in cui durante la ricomposizione del risultato finale si è deciso di utilizzare l'operatore di outer join per ricomporre i risultati delle query locali. Come è noto tale operatore non gode la proprietà asociativa e dunque la sequenza di esecuzione diventa molto importante. A questo proposito viene applicato l'algoritmo RWOBE che genera una pseudo-sequenza di outer join che, combinando i risultati delle query locali, ottiene la risposta desiderata.

I prossimi sviluppi dovranno essere rivolti alla ricerca di metodi più semplici e veloci di individuazione delle classi locali coinvolte nell'interrogazione ed allo sviluppo del software che implementi l'algoritmi ed i metodi trattati.

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 organizazzione 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 delle 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 autonomamente, 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

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

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

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

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

Appendice C

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

Quella che segue è la descrizione in linguaggio ODL$_{I^3}$ dell'esempio a cui si è fatto riferimento in questa tesi, quello dell'Università.

```
UNIVERSITY source:
interface University_Worker
( source relational University
  extent University_Worker
  key first_name, last_name
  foreign_key dept_code )
{ attribute string first_name;
  attribute string last_name;
  attribute integer dept_code;
  attribute integer pay; };

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;
  attribute integer pay; };

interface School_Member
( source relational University
  extent School_Members
  keys first_name, last_name )
{ attribute string  first_name;
  attribute string  last_name;
  attribute string  faculty;
  attribute integer year; }

interface Department
( source relational University
  extent Departments

interface Section
( source relational University
  extent Sections
```

```
  key dept_code )
{ attribute string dept_name;
  attribute integer dept_code;
  attribute integer budget;
  attribute string dept_area; };
```

```
  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; };
```

COMPUTER_SCIENCE source:

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

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

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

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

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

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

```
  attribute string county;
  attribute integer number; };


Tax_Position source:

interface University_Student
( source file Tax_Position
  extent University_Students
  key student_code )
{ attribute string name;
  attribute integer student_code;
  attribute string faculty_name;
  attribute integer tax_fee; };
```

Appendice D

Grammatica OQL

La grammatica OQL viene descritta usando la notazione BNF e utilizzando la seguente simbologia:

- `symbol` indica una espressione OQL che non viene tradotta ma che è necessario specificare in quanto è prevista dalla sintassi.

- **symbol** indica un elemento terminale del linguaggio.

- *symbol_name* indica che deve essere specificato un identificatore il cui significato semantico è indicato dalla prima parte del nome.

- *symbol_literal* indica che deve essere specificato un simbolo di tipo literal. Ad es. “una stringa” viene indicato come *string_literal*

⟨Query⟩ ::= **(** ⟨Query⟩ **)**|
⟨SelectExpr⟩
⟨SelectPreamble⟩ ::= **select distinct** |
select
⟨SelectExpr⟩ ::= ⟨SelectPreamble⟩ ⟨ProjectionList⟩
⟨FromClause⟩
⟨WhereClause⟩
⟨ProjectionList⟩ ::= ⟨ProjectionAttributes⟩ |
*
⟨ProjectionAttributes⟩ ::= ⟨Attribute⟩ |
⟨ProjectionAttributes⟩ **,** ⟨Attribute⟩
⟨Attribute⟩ ::= ⟨Projection⟩ |
⟨Property⟩
⟨Projection⟩ ::= **(** ⟨Projection⟩ **)** |
⟨Identifier⟩ **,** ⟨Property⟩ |
⟨Property⟩ **as** ⟨Identifier⟩ |
⟨Property⟩ ::= **(** ⟨Property⟩ **)** |
⟨Basic⟩ |
⟨Identifier⟩ |
⟨Accesor⟩
⟨Basic⟩ ::= **nil** |
true |
false |
⟨FloatLiteral⟩ |
⟨IntegerLiteral⟩ |
⟨StringLiteral⟩
⟨StringLiteral⟩ ::= " ⟨String⟩ "
⟨IntegerLiteral⟩ ::= ⟨UnsignedLong⟩ |
⟨Sign⟩ ⟨UnsignedLong⟩
⟨FloatLiteral⟩ ::= **.** ⟨UnsignedLong⟩ |
⟨Sign⟩ **.** ⟨UnsignedLong⟩ |
⟨IntegerLiteral⟩ **.** ⟨UnsignedLong⟩
⟨Sign⟩ ::= + | −
⟨Digit⟩ ::= **0** | **1** | ... | **9**
⟨Char⟩ ::= **a** | **b** | ... | **z** | **A** | **B** | ... | **Z**
⟨UnsignedLong⟩ ::= ⟨Digit⟩ |
⟨Digit⟩ ⟨UnsignedLong⟩
⟨String⟩ ::= ⟨Char⟩ | ⟨Digit⟩ | _ |
⟨Char⟩ ⟨String⟩ |
⟨Digit⟩ ⟨String⟩ |
_ ⟨String⟩
⟨Identifier⟩ ::= ⟨Char⟩ |
⟨Char⟩ ⟨String⟩
⟨Accessor⟩ ::= ⟨Identifier⟩ ⟨Path⟩ ⟨Accessor⟩ |
⟨Identifier⟩ ⟨Path⟩ ⟨Identifier⟩
⟨Path⟩ ::= . | − >

⟨FromClause⟩ ::= **from** ⟨VariableDeclaration⟩
⟨VariableDeclaration⟩ ::= ⟨Identifier⟩ |
⟨Identifier⟩ **as** ⟨Identifier⟩ |
⟨Identifier⟩ ⟨Identifier⟩
⟨WhereClause⟩ ::= ϵ |
where ⟨Predicates⟩
⟨Predicates⟩ ::= **(** ⟨Predicates⟩ **)** |
⟨Comparison⟩ |
⟨BooleanExpr⟩ |
⟨CollectionExpr⟩
⟨Comparison⟩ ::= ⟨Operand⟩ ⟨ComparisonOperator⟩
⟨Quantifier⟩ ⟨Operand⟩ |
⟨Property⟩ **like** ⟨StringLiteral⟩
⟨ComparisonOperator⟩ ::= $>$ | $>=$ | $<$ | $<=$ | $=$ | **!** $=$
⟨Quantifier⟩ ::= ϵ |
some |
any |
all
⟨Operand⟩ ::= ⟨Basic⟩ |
⟨SimpleExpr⟩ |
⟨Property⟩
⟨SimpleExpr⟩ ::= ⟨Property⟩ + ⟨Property⟩ |
⟨Property⟩ - ⟨Property⟩ |
⟨Property⟩ / ⟨Property⟩ |
⟨Property⟩ * ⟨Property⟩ |
- ⟨Property⟩ |
+ ⟨Property⟩ |
⟨Property⟩ **mod** ⟨Property⟩ |
abs(⟨Property⟩ **)** |
⟨Property⟩ || ⟨Property⟩
⟨BooleanExpr⟩ ::= **not** ⟨Predicates⟩ |
⟨Predicates⟩ **and** ⟨Predicates⟩ |
⟨Predicates⟩ **or** ⟨Predicates⟩
⟨CollectionExpr⟩ ::= **for all** ⟨Identifier⟩ **in** ⟨Property⟩ **:** ⟨Condition⟩ |
exists ⟨Identifier⟩ **in** ⟨Property⟩ **:** ⟨Condition⟩ |
exists(⟨Accessor⟩ **)** |
unique(⟨Accessor⟩ **)** |
⟨Property⟩ **in** ⟨Condition⟩ |
count(⟨Property⟩ **)** |
sum(⟨Property⟩ **)** |
min(⟨Property⟩ **)** |
max(⟨Property⟩ **)** |
avg(⟨Property⟩ **)**

Appendice E

Restrizione dell' OQL per le BasicQuery

Le Basic Query costituiscono di fatto una restrizione del linguaggio OQL. Di seguito viene riportata la descrizione in forma BNF di tale restrizione.

⟨Query⟩ ::= **(** ⟨Query⟩ **)**|
⟨SelectExpr⟩
⟨SelectPreamble⟩ ::= **select distinct** |
select
⟨SelectExpr⟩ ::= ⟨SelectPreamble⟩ ⟨ProjectionList⟩
⟨FromClause⟩
⟨WhereClause⟩
⟨ProjectionList⟩ ::= ⟨ProjectionAttributes⟩ |
*
⟨ProjectionAttributes⟩ ::= ⟨Attribute⟩ |
⟨ProjectionAttributes⟩ **,** ⟨Attribute⟩
⟨Attribute⟩ ::= ⟨Projection⟩ |
⟨Property⟩
⟨Projection⟩ ::= **(** ⟨Projection⟩ **)** |
⟨Identifier⟩ **,** ⟨Property⟩ |
⟨Property⟩ **as** ⟨Identifier⟩ |
⟨Property⟩ ::= **(** ⟨Property⟩ **)** |
⟨Basic⟩ |
⟨Identifier⟩ |
⟨Accesor⟩
⟨Basic⟩ ::= **nil** |
true |
false |
⟨FloatLiteral⟩ |
⟨IntegerLiteral⟩ |
⟨StringLiteral⟩
⟨StringLiteral⟩ ::= " ⟨String⟩ "
⟨IntegerLiteral⟩ ::= ⟨UnsignedLong⟩ |
⟨Sign⟩ ⟨UnsignedLong⟩
⟨FloatLiteral⟩ ::= **.** ⟨UnsignedLong⟩ |
⟨Sign⟩ **.** ⟨UnsignedLong⟩ |
⟨IntegerLiteral⟩ **.** ⟨UnsignedLong⟩
⟨Sign⟩ ::= $+$ | $-$
⟨Digit⟩ ::= **0** | **1** | ... | **9**
⟨Char⟩ ::= **a** | **b** | ... | **z** | **A** | **B** | ... | **Z**
⟨UnsignedLong⟩ ::= ⟨Digit⟩ |
⟨Digit⟩ ⟨UnsignedLong⟩
⟨String⟩ ::= ⟨Char⟩ | ⟨Digit⟩ | _ |
⟨Char⟩ ⟨String⟩ |
⟨Digit⟩ ⟨String⟩ |
_ ⟨String⟩
⟨Identifier⟩ ::= ⟨Char⟩ |
⟨Char⟩ ⟨String⟩
⟨Accessor⟩ ::= ⟨Identifier⟩ ⟨Path⟩ ⟨Accessor⟩ |
⟨Identifier⟩ ⟨Path⟩ ⟨Identifier⟩
⟨Path⟩ ::= . | $->$

⟨FromClause⟩ ::= **from** ⟨VariableDeclaration⟩
⟨VariableDeclaration⟩ ::= ⟨Identifier⟩ |
⟨Identifier⟩ **as** ⟨Identifier⟩ |
⟨Identifier⟩ ⟨Identifier⟩
⟨WhereClause⟩ ::= ϵ |
where ⟨Predicates⟩
⟨Predicates⟩ ::= **(** ⟨Predicates⟩ **)** |
⟨Comparison⟩ |
⟨BooleanExpr⟩ |
⟨CollectionExpr⟩
⟨Comparison⟩ ::= ⟨Operand⟩ ⟨ComparisonOperator⟩ ⟨Quantifier⟩
⟨Operand⟩ |
⟨Property⟩ **like** ⟨StringLiteral⟩
⟨ComparisonOperator⟩ ::= $>$ | $>=$ | $<$ | $<=$ | $=$ | **!** $=$
⟨Quantifier⟩ ::= ϵ |
some |
any |
all
⟨Operand⟩ ::= ⟨Basic⟩ |
⟨SimpleExpr⟩ |
⟨Property⟩
⟨SimpleExpr⟩ ::= ⟨Property⟩ + ⟨Property⟩ |
⟨Property⟩ - ⟨Property⟩ |
⟨Property⟩ / ⟨Property⟩ |
⟨Property⟩ * ⟨Property⟩ |
- ⟨Property⟩ |
+ ⟨Property⟩ |
⟨Property⟩ **mod** ⟨Property⟩ |
abs(⟨Property⟩ **)** |
⟨Property⟩ || ⟨Property⟩
⟨BooleanExpr⟩ ::= **not** ⟨Predicates⟩ |
⟨Predicates⟩ **and** ⟨Predicates⟩ |
⟨Predicates⟩ **or** ⟨Predicates⟩
⟨CollectionExpr⟩ ::= **for all** ⟨Identifier⟩ **in** ⟨Property⟩ **:** ⟨Condition⟩ |
exists ⟨Identifier⟩ **in** ⟨Property⟩ **:** ⟨Condition⟩ |
exists(⟨Accessor⟩ **)** |
unique(⟨Accessor⟩ **)** |
⟨Property⟩ **in** ⟨Condition⟩ |
count(⟨Property⟩ **)** |
sum(⟨Property⟩ **)** |
min(⟨Property⟩ **)** |
max(⟨Property⟩ **)** |
avg(⟨Property⟩ **)**

Bibliografia

[1] Gio Wiederhold. Mediators in the architecture of Future Information Systems. *IEEE Computer*, 25:38–49, 1992.

[2] A. Zanoli. SI-Designer, un tool di ausilio all'integrazione di sorgenti di dati eterogenee distribuite: progetto e realizzazione. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[3] G. P. Grifa. Analisi di Affinità Strutturali fra classi ODL_{I^3} nel Sistema MOMIS. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1998-1999.

[4] S. 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, 1997-1998.

[5] H. Garcia-Molina et al. The TSIMMIS Approach to Mediation: Data Models and Languages. In *NGITS workshop*, 1995. Available ftp://db.stanford.edu/pub/garcia/1995/tsimmis-models-languages.ps.

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

[7] Y.Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *VLDB Int. Conf.*, Bombay, India, September 1996.

[8] N.Guarino. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. Technical report, Summer School on Information Extraction, Frascati, Italy, July 1997.

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

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

[11] 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.

[12] Oliver M.Duschka and Micheal R.Genesereth. Infomaster - An Information Integration Toolkit. Technical report, Department of Computer Science. Stanford University, 1996.

[13] 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.

[14] Alon Levy, Dana Florescu, Jaewoo Kang, Anand Rajaraman, and Joanne J. Ordille. The Information Manifold Project. Available at http://www.research.att.com/ levy/imhome.html.

[15] 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.

[16] 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.

[17] 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.

[18] Y. Arens, C. A. Knoblock, and C. Hsu. Query Processing in the SIMS Information Mediator. *Advanced Planning Technology*, 1996.

[19] A. Zaccaria. MOMIS: Il componente Query Manager. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[20] 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, 1999-2000.

[21] A. Rabitti. Architettura di un Mediatore per un Sistema di Sorgenti Distribuite ed Autonome. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[22] Silvia Zanni. Il componente Query Manager di MOMIS: esecuzione di interrogazioni. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1999-2000.

[23] R. Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. Technical report, Bell Laboratories, Bell Laboratories, 1996.

[24] AA. VV. The common object request broker: Architecture and specification. Technical report, Object Request Broker Task Force, 1993. Revision 1.2, Draft 29, December.

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

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

[27] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. A Description Logics Based Tool for Schema Validation and Semantic Query Optimization in Object Oriented Databases. Technical report, sesto convegno AIIA, 1997.

[28] A.G. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[29] 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.

[30] 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.

[31] I. Schmitt and G. Saake. Merging Inheritance Hierarchies for Database Integration. *IEEE Trans. on Knowledge and Data Engineering*, 1999.

[32] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. An Intelligent Approach to Information Integration. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, june 1998.

[33] S. Castano and V. De Antonellis. Semantic Dictionary Design for Database Interoperability. In *Proc. of Int. Conf. on Data Engineering, ICDE'97*, Birmingham, UK, 1997.

[34] S. Castano, V. De Antonellis, and S. De Capitani Di Vimercat. Global viewing of heterogeneous data sources. Technical Report 98-08, Dip. Elettronica e Informazione, Politecnico di Milano, Milano, Italy, 1998.

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

[36] Francesco Venuta. Trattamento della conoscenza estensionale nel sistema MOMIS. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1999-2000.

[37] S. Abiteboul and P. Kanellakis. Object identity as query language primitive. *Journal of the ACM*, (45)(5):798–842, 1998. A first version appeared in SIGMOD'89.

[38] Anand Rajaraman and Jeffrey D. Ullman. Integration information by outerjoins and full disjunction. *ACME*, 1997.

[39] C. Glaindo-Legaria. Outerjoins as disjunctions. *CWI*, 1993.

[40] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM, 30:3, pp 514-550*, 1983.

[41] Chen-Chuan K. Chang and Hector Garcia-Molina. Query mapping across heterogeneous information sources (extended version). *JVLDB*, 2000.

[42] Chen-Chuan K. Chang and Hector Garcia-Molina. Approximate query tranlation across heterogeneous information sources (extended version). *JVLDB*, 1999.

[43] M. Yannakakis. Algorithms for acyclic database schemes. *7th International Conf. VLDB , ACM*, pages 82–94, 1982.

[44] E. F. Codd. Further normalization of the database relational model. *In database Systems, Courant Computer Science Symposia 6.*, 6:65–98, 1971.

[45] M. Vincini. ODB-QOptimizer: un ottimizzatore semantico di interrogazioni. Tesi di Laurea, Univeristà di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1994.

[46] Serge Abiteboul Y.Papakonstantinou, H. Garcia-Molina. Object Fusion in Mediator Systems. In *VLDB Int. Conf.*, 1996.

[47] Aron Rosenthal Edward Sciore, Michael Siegel. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database System*, 19(2):254–290, June 1994.

[48] Ken Smith and Leo Orbst. Unpacking The Semantics of Source and Usage To Perform Semantic Reconciliation In Large-Scale Information Systems. *SIGMOD Records*, 28(1):26–31, 1999.

[49] Micol Ferrari. Progetto e realizzazione di tecniche di object fusion nel sistema momis. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1999-2000.