DOTTORATO DI RICERCA IN
INGEGNERIA ELETTRONICA ED INFORMATICA
XI CICLO

Sede Amministrativa
Università degli Studi di MODENA

TESI PER IL CONSEGUIMENTO DEL TITOLO DI
DOTTORE DI RICERCA

# Utilizzo di tecniche di Intelligenza Artificiale nell'Integrazione di Sorgenti Informative Eterogenee


Supervisore

Chiar.mo Prof. Sonia Bergamaschi

Candidato

Ing. Maurizio Vincini

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

# RINGRAZIAMENTI

*Ringrazio la Professoressa Sonia Bergamaschi e l'Ing. Domenico Beneventano per l'aiuto fornito alla realizzazione della presente tesi.*
*Un ringraziamento particolare va inoltre ai tesisti Ing. Paolo Apparuti, Ing. Alberto Corni, Ing. Simone Montanari, Ing. Stefano Riccio, Ing. Alberta Rabitti, Ing. Andrea Salvarani, Ing. Ivano Benetti per la preziosa collaborazione che in questi anni hanno fornito allo sviluppo del progetto presentato nella tesi.*

# Indice


**1 Introduzione** **1**

**I Tecniche di Intelligenza artificiale** **5**

**2 Logiche Descrittive e database: il formalismo OLCD** **7**

2.1 Logiche Descrittive . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.2 La Logica Descrittiva OLCD . . . . . . . . . . . . . . . . . . . . . 8
2.2.1 Le regole OLCD e l'espansione semantica di un tipo . . 10
2.2.2 Schema e istanza del database . . . . . . . . . . . . . . 11
2.2.3 Sussunzione ed Espansione Semantica di un tipo . . . . 16
2.2.4 Algoritmi . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.3 Esempio applicativo . . . . . . . . . . . . . . . . . . . . . . . . . 20

**3 Il prototipo ODB-Tools** **23**

3.1 Il linguaggio ODL . . . . . . . . . . . . . . . . . . . . . . . . . . 24
3.1.1 Schema di esempio . . . . . . . . . . . . . . . . . . . . 24
3.1.2 ODL e il modello OLCD . . . . . . . . . . . . . . . . . . 25
3.1.3 Validazione e Sussunzione . . . . . . . . . . . . . . . . 29
3.2 Ottimizzazione semantica delle interrogazioni . . . . . . . . . . . 30
3.3 Confronti con altri lavori . . . . . . . . . . . . . . . . . . . . . . 33
3.4 Architettura di ODB-Tools . . . . . . . . . . . . . . . . . . . . . 35
3.5 Struttura del sito WWW e demo di esempio . . . . . . . . . . . 37
3.6 Risultati sperimentali . . . . . . . . . . . . . . . . . . . . . . . . 41

**4 Introduzione dei metodi in OLCD** **43**

4.0.1 Introduzione di uno schema con operazioni . . . . . . . 43
4.0.2 Operazioni e Vincoli di Integritá . . . . . . . . . . . . . 46
4.1 Principio di covarianza e controvarianza . . . . . . . . . . . . . . 50
4.2 Controllo di consistenza in schemi con operazioni . . . . . . . . 52
4.3 Utilizzo nelle applicazioni industriali . . . . . . . . . . . . . . . 54

4.3.1 Descrizione della problematica . . . . . . . . . . . . . . . 54
4.3.2 Risultati conseguiti . . . . . . . . . . . . . . . . . . . . . . . 55

**5 Ottimizzazione semantica di schemi ciclici 59**
5.1 Limiti dell'algoritmo di espansione semantica . . . . . . . . . . . 59
5.2 Algoritmo di Espansione con soglia . . . . . . . . . . . . . . . . . 62
5.3 Lavori in corso . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

**II Integrazione di Informazioni 67**

**6 L'Integrazione Intelligente di Informazioni 69**
6.1 Architettura di riferimento per sistemi $I^3$ . . . . . . . . . . . . 71
6.1.1 A cosa serve la tecnologia $I^3$ e quali problemi deve risolvere . . . . . . . . . . . . . . . . . . . . . . . . . . 72
6.1.2 Servizi di Coordinamento . . . . . . . . . . . . . . . . . . 75
6.1.3 Servizi di Amministrazione . . . . . . . . . . . . . . . . . . 75
6.1.4 Servizi di Integrazione e Trasformazione Semantica . . 76
6.1.5 Servizi di Wrapping . . . . . . . . . . . . . . . . . . . . . 77
6.1.6 Servizi Ausiliari . . . . . . . . . . . . . . . . . . . . . . . . 78

**7 Stato dell'arte nell'ambito dell'integrazione di informazioni 79**
7.1 TSIMMIS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
7.1.1 Il modello OEM . . . . . . . . . . . . . . . . . . . . . . . . 81
7.1.2 Il linguaggio MSL . . . . . . . . . . . . . . . . . . . . . . . 82
7.1.3 Il generatore di Wrapper . . . . . . . . . . . . . . . . . . . 82
7.1.4 Il generatore di Mediatori . . . . . . . . . . . . . . . . . . 84
7.1.5 Il Linguaggio LOREL . . . . . . . . . . . . . . . . . . . . . 86
7.1.6 Pregi e difetti di TSIMMIS . . . . . . . . . . . . . . . . . 87
7.2 GARLIC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
7.2.1 Il linguaggio GDL . . . . . . . . . . . . . . . . . . . . . . . 90
7.2.2 Query Planning . . . . . . . . . . . . . . . . . . . . . . . . 92
7.2.3 Pregi e difetti di GARLIC . . . . . . . . . . . . . . . . . . 92
7.3 SIMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
7.3.1 Integrazione delle sorgenti . . . . . . . . . . . . . . . . . . 95
7.3.2 Query Processing . . . . . . . . . . . . . . . . . . . . . . . 96
7.3.3 Pregi e difetti di SIMS . . . . . . . . . . . . . . . . . . . . 98

**8 Il sistema di integrazione intelligente di informazioni MOMIS 99**
8.0.4 Problematiche da affrontare . . . . . . . . . . . . . . . . . 102

8.0.5 Problemi ontologici . . . . . . . . . . . . . . . . . . . . . 103
8.0.6 Problemi semantici . . . . . . . . . . . . . . . . . . . . . 103
8.1 La proposta di integrazione . . . . . . . . . . . . . . . . . . . . . 105
8.2 L'architettura di MOMIS . . . . . . . . . . . . . . . . . . . . . . . 108
8.2.1 Semistructured data and object patterns . . . . . . . . 111
8.3 Esempio di riferimento . . . . . . . . . . . . . . . . . . . . . . . . 115
8.4 Il linguaggio ODL$_{I^3}$ . . . . . . . . . . . . . . . . . . . . . . . . . 116
8.5 Estensioni al formalismo OLCD per l'integrazione . . . . . . . 119
8.6 Estrazione di Relazioni Terminologiche . . . . . . . . . . . . . 120
8.7 Analisi di Affinità delle classi ODL$_{I^3}$ . . . . . . . . . . . . . . . 127
8.7.1 Coefficienti di Affinità . . . . . . . . . . . . . . . . . . . . 128
8.7.2 Considerazioni sul processo di affinità . . . . . . . . . . 131
8.8 Generazione dei Cluster di classi ODL$_{I^3}$ . . . . . . . . . . . . 132
8.9 Costruzione dello Schema Globale di mediatore . . . . . . . . 134
8.10 Query Processing e Optimization . . . . . . . . . . . . . . . . . 137
8.10.1 Semantic Optimization . . . . . . . . . . . . . . . . . . . 138
8.10.2 Query Plan Formulation . . . . . . . . . . . . . . . . . . 139
8.11 Confronto con altri lavori . . . . . . . . . . . . . . . . . . . . . . 142

**9 Conclusioni 147**

**A Glossario $I^3$ 149**
A.1 Architettura . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 149
A.2 Servizi . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 151
A.3 Risorse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
A.4 Ontologia . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157

**B Il linguaggio di descrizione ODL$_{I^3}$ 159**

**C Il linguaggio di descrizione ODL dello standard ODMG-93 161**

**D Sorgenti di esempio in linguaggio ODL$_{I^3}$ 167**

# Elenco delle figure


2.1 Relazione di sussunzione dovuta alle regole . . . . . . . . . . . 17

3.1 Rappresentazione grafica dello schema Università . . . . . . . 24
3.2 Architettura di ODB-Tool . . . . . . . . . . . . . . . . . . . . 35
3.3 Validazione dello schema . . . . . . . . . . . . . . . . . . . . 37
3.4 Pagina di presentazione dello schema validato . . . . . . . . . 38
3.5 Finestra di presentazione degli attributi di una classe . . . . . 39
3.6 Ottimizzazione di una query . . . . . . . . . . . . . . . . . . . 39
3.7 Lo schema Università prima della validazione . . . . . . . . . . 40
3.8 Rapporto tra Costo Ottimizzato ed Originale in funzione della dimensione de DB . . . . . . . . . . . . . . . . . . . . . . . . . 42

4.1 Schema del database con regole . . . . . . . . . . . . . . . . . 56

6.1 Diagramma dei servizi $I^3$ . . . . . . . . . . . . . . . . . . . . 74

7.1 Architettura TSIMMIS . . . . . . . . . . . . . . . . . . . . . . 80
7.2 Oggetti esportati da **CS** in OEM . . . . . . . . . . . . . . . . 85
7.3 Oggetti esportati da **WHOIS** in OEM . . . . . . . . . . . . . 85
7.4 Oggetti esportati da **MED** . . . . . . . . . . . . . . . . . . . . 86
7.5 Architettura GARLIC . . . . . . . . . . . . . . . . . . . . . . . 89
7.6 GDL schema . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
7.7 Esempio di query SIMS . . . . . . . . . . . . . . . . . . . . . . 95
7.8 Mapping tra *domain model* e modello locale . . . . . . . . . . 96
7.9 Query Processing . . . . . . . . . . . . . . . . . . . . . . . . . 97

8.1 Servizi $I^3$ presenti nel mediatore . . . . . . . . . . . . . . . . . 101
8.2 Architettura del sistema MOMIS . . . . . . . . . . . . . . . . . 108
8.3 Cardiology Department (CD) . . . . . . . . . . . . . . . . . . . 113
8.4 Gli object pattern della sorgente semistrutturata di esempio . 114
8.5 Livelli di astrazione degli oggetti semistrutturati . . . . . . . . 114
8.6 Intensive Care Department (ID) . . . . . . . . . . . . . . . . . 115
8.7 Un esempio di oggetti fortemente eterogenei . . . . . . . . . . 118

8.8 Un esempio di tipo unione . . . . . . . . . . . . . . . . . . . . . 119
8.9 Common Thesaurus relativo ai Dipartimenti `Cardiology` e `Intensive Care` . . . . . . . . . . . . . . . . . . . . . . . . . 126
8.10 Procedura di clustering . . . . . . . . . . . . . . . . . . . . . . . . . 133
8.11 Affinity trees for the `Cardiology` and `Intensive Care` data sources . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 133
8.12 Esempio di classi globali in ODL$_{I^3}$ . . . . . . . . . . . . . . . 138
8.13 `Hospital_Patient` mapping table . . . . . . . . . . . . . . . . . 138
8.14 Algoritmo di Controllo ed Eliminazione . . . . . . . . . . . . . 140

# Elenco delle tabelle


2.1 Schema del dominio `Magazzino` in sintassi ODMG–93 . . . . . 12
2.2 Regole di integrità sul dominio `Magazzino` . . . . . . . . . . . 13
2.3 Schema con Regole di Integrità in linguaggio OCDL . . . . . . 14

3.1 Esempio ODL: database universitario . . . . . . . . . . . . . . . 26
3.2 Dimensioni delle Classi del Database . . . . . . . . . . . . . . . 41

5.1 Algoritmo di espansione con soglia . . . . . . . . . . . . . . . . . 64
5.2 Schema con regole di prova . . . . . . . . . . . . . . . . . . . . . 65

# Capitolo 1

# Introduzione

Negli ultimi decenni numerosi sistemi informativi più o meno evoluti sono stati sviluppati da aziende industriali e di servizi e da amministrazioni pubbliche seguendo approcci metodologici e tecnici tra i più disparati. La crescita disomogenea e irregolare di questi sistemi, spinta anche dalla diffusione di nuove tecnologie e nuovi strumenti, ha portato ad una evidente diseconomicità nella gestione dei dati determinata da varie ragioni, come, ad esempio, l'impossibilità di sfruttare le potenzialità informative dei sistemi disponibili e la necessità di verificare e provvedere al mantenimento dei vincoli tra dati memorizzati in sistemi diversi.

Grazie all'esplosione avvenuta in questi ultimi anni della disponibilità dei dati accessibili via rete (tanto sulla rete Internet, quanto all'interno di un'azienda) avvenuta in questi ultimi anni, ritrovare ed integrare informazioni provenienti da differenti sorgenti è divenuto un fattore cruciale per il miglioramento del patrimonio informativo aziendale. D'altra parte, molti dei compiti che devono essere portati a termine dagli utenti di sistemi complessi basati sui dati impongono l'interazione con una molteplicità di fonti di informazioni. Esempi possono essere trovati nelle aree che coinvolgono l'analisi aggregata dei dati (e.g. previsioni logistiche), come pure nella pianificazione delle risorse e nei sistemi di supporto alle decisioni.

Il reperimento delle informazioni desiderate, distribuite su una molteplicità di sorgenti, richiede generalmente familiarità con il contenuto, le strutture ed i linguaggi di interrogazione propri di queste risorse separate. L'utente deve quindi scomporre la propria interrogazione (attraverso la quale ha espresso le proprietà che devono caratterizzare i dati che sta cercando) in una sequenza di sottointerrogazioni dirette alle varie sorgenti di informazione, e deve poi *trattare* ulteriormente i risultati parziali ricevuti, al fine di ottenere una risposta unificata, considerando le possibili trasformazioni che devono subire questi dati, le relazioni che li legano, le proprietà che possono avere

in comune, le discrepanze che ancora sussistono tra loro: con un numero sempre maggiore di sorgenti, e di dati da manipolare, è quindi molto difficile individuare persone che posseggano tutte le conoscenze necessarie a portare a termine un compito di questo tipo, e un processo di automazione dell'intera fase di reperimento ed integrazione di informazioni diviene una necessità.

Le soluzioni più largamente utilizzate sono le seguenti:

1. creazione di una federazione (o una cooperazione) tra le sorgenti coinvolte;

2. costruzione di un data warehouse;

Il primo approccio consiste nella progettazione di strati di software al di sopra dei sistemi esistenti capaci di coordinarli ed integrarli preservandone la struttura ed il funzionamento interno [101, 82, 70]. Accanto a notevoli evidenti vantaggi, la soluzione presenta alcuni svantaggi: *i)* la necessità di una forte interoperabilità dei sistemi informatici che entrano a far parte della federazione, *ii)* la richiesta di accesso continuativo alle fonti di dati comporta un alto traffico tra i siti cooperanti e quindi possibili ritardi nel recupero dei dati, *iii)* la complessità di realizzazione della federazione in quanto frutto di un compromesso tra le esigenze di autonomia delle singole basi di dati e le esigenze di efficienza di gestione della federazione.

Il secondo approccio prevede la costruzione di una nuova base di dati, chiamata data warehouse [59, 103], nella quale vengono riorganizzate tutte le informazioni possedute da un'organizzazione e sparse nei vari sistemi esistenti. Il data warehouse è una sorta di deposito di dati, il cui obiettivo è quello di consentire l'estrazione e la riconciliazione dei dati delle basi di dati operative. A differenza della soluzione precedente non è richiesto accesso continuativo alle fonti dei dati, mentre è richiesta la materializzazione dei dati presenti sui sistemi operanti sul campo. Questo comporta l'utilizzo di vari algoritmi per la gestione dell'allineamento periodico dei dati che appesantiscono l'utilizzo del data warehouse.

Entrambe le soluzioni richiedono un processo di integrazione degli schemi coinvolti allo scopo di ottenere una visione univoca di tutti i dati disponibili. Questo processo è molto laborioso e generalmente viene eseguito manualmente dal progettista [12]. In presenza di un numero rilevante di basi di dati preesistenti, generalmente di grandi dimensioni e di struttura complessa, l'approccio "manuale" all'integrazione appare di difficile applicazione. Un approccio alternativo verrà presentato nel corso della tesi e consiste nell'individuare, o ancor meglio derivare automaticamente, proprietà comuni relative agli schemi coinvolti (proprietà di interschema), in modo da poter essere

utilizzate in un processo di integrazione che potrà essere, il più possibile, automatizzato [53, 23].

La presente tesi si articola su due temi fondamentali. Il primo è relativo all'estensione delle Logiche Descrittive introdotte in ambito Intelligenza Artificiale per modellare basi di dati ad oggetti e sfruttare le tecniche di inferenza di tali logiche al fine di risolvere problemi fondamentali quali: il controllo di consistenza di schemi e l'ottimizzazione di interrogazioni.

Il secondo tema è relativo all'integrazione di informazioni testuali eterogenee e distribuite. Nell'affrontare questo tema, si è potuto verificare l'efficacia di logiche descrittive estese, quali quella introdotta nella prima parte per risolvere il problema della integrazione "intelligente" e quindi effettiva.

Entrambi i temi sono stati affrontati sia dal punto di vista teorico che realizzativo, ed hanno portato allo sviluppo di due strumenti software: ODB-Tools (per il primo tema) e MOMIS (per il secondo).

La prima parte della tesi (Capitoli 2,3,4,5) è stata quindi dedicata alla presentazione e all'estensione delle tecniche di logica descrittiva per renderle adeguate alla tematica dell'integrazione. In particolare, il Capitolo 2 presenta la logica descrittiva OLCD, che è un'estensione di quella presentata in [25], le tecniche di validazione degli schemi e l'ottimizzazione semantica [78]. Il Capitolo 3 descrive il sistema ODB-Tools che realizza tali funzionalità e al quale ho lavorato sia nella fase di progettazione che durante la realizzazione nell'ambito di questa tesi. Nel Capitolo 4 la logica OLCD viene estesa per includere la definizione dei metodi. Il capitolo 5 completa la definizione dell'algoritmo di ottimizzazione semantica, estendendolo anche a query basate su schemi con regole ciclici. Inoltre verrà definita e calcolata la *forma minima* dell'interrogazione ottenuta dalla query ottimizzata attraverso l'eliminazione dei fattori ridondanti.

Nella seconda parte viene presentato il sistema MOMIS (Mediator envirOnment for Multiple Information Sources), per l'integrazione di sorgenti di dati strutturati e semistrutturati [23, 22, 24] secondo l'approccio della federazione delle sorgenti. MOMIS nasce all'interno del progetto MURST 40% *INTERDATA*, come collaborazione tra l'unitá operativa dell'Universitá di Milano e l'unitá operativa dell'Universitá di Modena di cui l'autore é membro. Inoltre il sistema prevede la definizione semi-automatica dello schema univoco integrato che utilizza le informazioni semantiche proprie di ogni schema (col termine schema si intende l'insieme di metadati che descrive un deposito di dati).

Vengono dapprima presentate l'introduzione alle problematiche dell'Integrazione di Informazioni (Capitolo 5) ed una rassegna delle soluzioni più interessanti proposte in letteratura (Capitolo 6). Successivamente (Capitolo 7) viene presentato il progetto del mediatore di informazioni MOMIS,

analizzando l'intero processo che porta, a partire da un insieme di schemi di sorgenti locali, alla definizione di una schema *globale*, che include tutti i concetti che le diverse fonti di informazioni vogliono condividere, e che sarà l'unico col quale l'utente dovrà interagire.

# Parte I

# Tecniche di Intelligenza artificiale

# Capitolo 2

# Logiche Descrittive e database: il formalismo OLCD

## 2.1 Logiche Descrittive

Per poter comprendere le potenzialità offerte da questi formalismi è vantaggioso fare riferimento ad una famiglia più generale di strumenti cui si ispirano tutte le ricerche sui sistemi intelligenti. Di seguito sono introdotti i concetti fondamentali che stanno alla base di qualsiasi Sistema a Base di Conoscenza (KBRS). La trattazione é volutamente semplificata, poiché l'obiettivo é solo quello di un'inquadramento dell'ambito di ricerca.

### La Logica del Primo Ordine

La necessità di rappresentare la conoscenza riconduce un problema di intelligenza ad uno di ricerca. Ad esempio, scrivere un programma finalizzato alla risoluzione di un certo problema richiede di compiere i seguenti passi:

1. scelta dell'algoritmo
2. scelta del linguaggio di programmazione
3. codifica del programma
4. esecuzione del programma

Un problema *di intelligenza*[1], per essere risolto da un calcolatore richiede quattro passi analoghi:

[1] Tipici problemi, legati alla storia iniziale dell'Intelligenza Artificiale, sono ad esempio quelli del gioco degli scacchi, del backgammon, oppure quelli di determinazione del percorso per raggiungere un certo scopo, come ad esempio la storiella del pastore della capra del cavolo e del lupo, o dei tre missionari e dei tre cannibali in riva al fiume.

1. identificazione della conoscenza necessaria a risolvere il problema
2. scelta del linguaggio adatto a rappresentare tale conoscenza
3. espressione della conoscenza attraverso il linguaggio identificato
4. soluzione del problema usando le **conseguenze** della conoscenza

L'importanza della Logica del Primo Ordine in relazione a problemi di intelligenza deriva sostanzialmente dalla relazione di *Implicazione Logica* ($\models$, entailment relation). Questo dipende dal fatto che , se qualcuno dispone di un certo stato di conoscenza, esprimibile con n proposizioni[2] $p_1, p_2, \ldots, p_n$, allora può concludere il fatto espresso dalla proposizione q. Espressa questa conoscenza come *statement* logici, l'ultimo passo del processo di soluzione di un problema di intelligenza consiste nello stabilire se, dati i fatti p e q, sia possibile asserire che $p \models q$. Per rispondere a questa domanda si applicano *Regole di Inferenza*. Queste regole, come ad esempio Modus Ponens, affermano che dato un certo insieme di conoscenze in una qualche forma specifica, sia legittimo aggiungere conoscenza (in una qualche altra forma). Questo, ovviamente, consente di stabilire se, in un certo dominio conoscitivo, una certa affermazione sia vera o falsa.

La Logica del Primo Ordine è semidecidibile, ovvero una procedura di inferenza termina se il fatto è effettivamente una conseguenza vera, altrimenti il processo non termina. Per portare questi problemi alla decidibilità è necessario ridurre le potenzialità espressive della logica. Per comprendere meglio quest'ultima affermazione, introduciamo la sintassi utilizzata per esprimere gli statement della $1^{st}OPL$. La Logica consente di parlare di *oggetti*, di loro *proprietà* e di reciproche *relazioni*. Un *atomo* è la base di ogni proposizione conoscitiva e si ottiene applicando una relazione ad uno o più oggetti del dominio di interesse. Combinando gli atomi attraverso opportuni operatori si originano le proposizioni della logica. Gli operatori sono: l'*and*, l'*or*, il *not* e l'*implicazione*, necessaria ad esprimere relazioni *if then*. Questi elementi determinano la *Predicate Logic*, mentre per parlare di $1^{st}OPL$ si introducono le *variabili* ed i *quantificatori* esistenziale ed universale.

## 2.2 La Logica Descrittiva OLCD

Le Logiche Descritive (DLs) [37, 86] costituiscono una restrizione delle $1^{st}OPL$ [36]: esse consentono di esprimere i *concept*[3] sottoforma di formule logiche, usando solamente predicati unari e binari, contenenti solo una

[2]In particolare si definisce Base di Conoscenza (KB) un insieme di proposizioni

[3]Un concept è una struttura del tutto simile ad una classe

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

Il formalismo OLCD deriva dal precedente **ODL**, proposto in [25], che estende l'espressività dei linguaggi sviluppati nell'area delle DLs. La caratteristica principale di OLCD consiste nell'assumere una ricca struttura per il sistema dei tipi di base: oltre ai classici tipi atomici *integer*, *boolean*, *string*, *real*, e tipi *mono-valore*, viene ora considerata anche la possibilità di utilizzare dei sottoinsiemi di questi (come potrebbero essere, ad esempio, intervalli di interi). A partire da questi tipi di base si possono definire i *tipi valore*, attraverso gli usuali costruttori di tipo definiti nei *CODMs*, quali *tuple*, *insiemi*, *liste* e *tipi classe*, che denotano insiemi di oggetti con una identità ed un valore associato. Ai tipi può essere assegnato un nome, mantenendo la distinzione tra nomi di *tipi valore* e nomi di *tipi classe*, che d'ora in poi denomineremo semplicemente *classi*: ciò equivale a dire che i nomi dei tipi vengono partizionati in nomi che indicano insiemi di oggetti (*tipi classe*) e nomi che rappresentano insiemi di valori (*tipi valore*). L'ereditarietà, sia semplice che multipla, è espressa direttamente nella descrizione di una classe tramite l'operatore di *intersezione*.
OLCD introduce inoltre la distinzione tra nomi di tipi *virtuali*, che descrivono condizioni necessarie e sufficienti per l'appartenenza di un oggetto del dominio ad un tipo (concetto che si può quindi collegare al concetto di *vista*), e nomi di tipi *primitivi*, che descrivono condizioni necessarie di appartenenza (e che quindi si ricollegano alle classi di oggetti). In [9], OLCD è stato esteso per permettere la formulazione dichiarativa di un insieme rilevante di vincoli di integrità definiti sulla base di dati. Attraverso questo logica descrittiva, è possibile descrivere, oltre alle classi, anche le *regole di integrità*: permettono la formulazione dichiarativa di un insieme rilevante di vincoli di integrità sottoforma di regole *if-then* i cui antecedenti e conseguenti sono espressioni di tipo OLCD. In tale modo, è possibile descrivere correlazioni tra proprietà strutturali della stessa classe, o condizioni sufficienti per il popolamento di sottoclassi di una classe data. In altre parole le rule costi-

tuiscono uno strumento dichiarativo per descrivere gli oggetti che popolano il sistema.

### 2.2.1 Le regole OLCD e l'espansione semantica di un tipo

La nozione di ottimizzazione semantica di una query è stata introdotta, per le basi di dati relazionali, da King [78, 77] e da Hammer e Zdonik [74]. L'idea di base di queste proposte è che i vincoli di integrità, espressi per forzare la consistenza di una base di dati, possano essere utilizzati anche per ottimizzare le interrogazioni fatte dall'utente, trasformando la query in una *equivalente*, ovvero con lo stesso insieme di oggetti di risposta, ma che può essere elaborata in maniera più efficiente.
Sia il processo di consistenza e classificazione delle classi dello schema, che quello di ottimizzazione semantica di una interrogazione, sono basati in ODB-Tools sulla nozione di *espansione* semantica di un tipo: l'espansione semantica permette di incorporare ogni possibile restrizione che non è presente nel tipo originale, ma che è logicamente implicata dallo schema (inteso come l'insieme delle classi, dei tipi, e delle regole di integrità). L'espansione dei tipi si basa sull'iterazione di questa trasformazione: se un tipo *implica* l'antecedente di una regola di integrità, allora il conseguente di quella regola può essere aggiunto alla descrizione del tipo stesso. Le *implicazioni* logiche fra i tipi (in questo caso il tipo da espandere e l'antecedente di una regola) sono determinate a loro volta utilizzando l'algoritmo di *sussunzione*, che calcola relazioni di sussunzione, simili alle relazioni di raffinamento dei tipi definite in [48].
Il calcolo dell'espansione semantica di una classe permette di rilevare nuove relazioni *isa*, cioé relazioni di specializzazione che non sono esplicitamente definite dal progettista, ma che comunque sono logicamente implicate dalla descrizione della classe e dello schema a cui questa appartiene. In questo modo, una classe può essere automaticamente classificata all'interno di una gerarchia di ereditarietà. Oltre che a determinare nuove relazioni tra classi virtuali, il meccanismo, sfruttando la conoscenza fornita dalle regole di integrità, è in grado di riclassificare pure le classi base (generalmente gli schemi sono forniti in termini di classi base).
Analogamente, rappresentando a run-time l'interrogazione dell'utente come una classe virtuale (l'interrogazione non è altro che una classe di oggetti di cui si definiscono le condizioni necessarie e sufficienti per l'appartenenza), questa viene classificata all'interno dello schema, in modo da ottenere l'interrogazione più specializzata tra tutte quelle semanticamente equivalenti alla

iniziale. In questo modo l'interrogazione viene spostata verso il basso nella gerarchia e le classi a cui si riferisce vengono eventualmente sostituite con classi più specializzate: diminuendo l'insieme degli oggetti da controllare per dare risposta all'interrogazione, ne viene effettuata una vera ottimizzazione indipendente da qualsiasi modello di costo fisico.

**Esempio di riferimento**

Nella prossima sezione viene descritto il formalismo OLCD: nel corso della trattazione, si farà riferimento ad un esempio esplicativo, di un dominio `Magazzino`, di cui riportiamo la descrizione in sintassi ODMG-93 nella tabella 2.1, e le regole di integrità su di esso definite in tabella 2.2.

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

### 2.2.2 Schema e istanza del database

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

Sia **A** un insieme numerabile di *attributi* (denotati da $a_1, a_2, \dots$) e $\mathcal{O}$ un insieme numerabile di *identificatori di oggetti* (denotati da $o, o', \dots$) disgiunti

**Classi dello Schema**

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

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

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

Tabella 2.2: Regole di integrità sul dominio `Magazzino`

da **D**. Si definisce l'insieme $\mathcal{V}(\mathcal{O})$ dei *valori su* $\mathcal{O}$ (denotati da $v, v'$) come segue (assumendo $p \geq 0$ e $a_i \neq a_j$ per $i \neq j$):

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

Gli identificatori di oggetti sono associati a valori tramite una *funzione totale* $\delta$ da $\mathcal{O}$ a $\mathcal{V}(\mathcal{O})$; in genere si dice che il valore $\delta(o)$ è lo *stato* dell'oggetto identificato dall'oid $o$.
Sia **N** l'insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che **A**, **B**, e **N** siano a due a due disgiunti. **N** è partizionato in tre insiemi **C**, **V** e **T**, dove **C** consiste di nomi per *tipi-classe base* ($C, C' \ldots$), **V** consiste di nomi per *tipi-classe virtuali* ($V, V' \ldots$), e **T** consiste di nomi per *tipi-valori* ($t, t', \ldots$). Un *path* $p$ è una sequenza di elementi $p = e_1 . e_2 . \ldots . e_n$, con $e_i \in \mathbf{A} \cup \{\triangle, \forall, \exists\}$. Con $\epsilon$ si indica il path vuoto.
$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$[4] indica l'insieme di tutte le *descrizioni di tipo finite* ($S$, $S'$, ...), dette brevemente *tipi*, su di un dato $\mathbf{A}, \mathbf{B}, \mathbf{N}$, ottenuto in accordo con la seguente regola sintattica:

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

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

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

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

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

$$\mathbf{R} \begin{cases} R_1: & \texttt{Material} \sqcap (\triangle.\texttt{risk}: > 10) \rightarrow \texttt{SMaterial} \\ R_2: & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\forall.\texttt{item}: \texttt{SMaterial}) \rightarrow \texttt{SStorage} \\ R_3: & \texttt{Storage} \sqcap (\triangle.\texttt{managed_by}.\triangle\texttt{level}: 6 \div 12) \rightarrow (\triangle.\texttt{category}: \texttt{``A2''}) \\ R_4: & \texttt{Storage} \sqcap (\triangle.\texttt{category}: \texttt{``A2''}) \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk}: > 10) \\ & \qquad \rightarrow \triangle[\texttt{managed_by}: \texttt{TManager}] \end{cases}$$

Tabella 2.3: Schema con Regole di Integrità in linguaggio OCDL

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

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

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

La *funzione interpretazione* $\mathcal{I}$ è una funzione da $\mathbf{S}$ a $2^{\mathcal{V}(\mathcal{O})}$ tale che: $\mathcal{I}[\top] = \mathcal{V}(\mathcal{O})$, $\mathcal{I}[B] = \mathcal{I}_{\mathbf{B}}[B]$[5] , $\mathcal{I}[C] \subseteq \mathcal{O}$, $\mathcal{I}[V] \subseteq \mathcal{O}$, $\mathcal{I}[t] \subseteq \mathcal{V}(\mathcal{O}) - \mathcal{O}$.

[5]Assumendo $\mathcal{I}_{\mathbf{B}}$ funzione di *interpretazione standard* da $\mathbf{B}$ a $2^{\mathbf{B}}$ tale che per ogni

L'interpretazione è estesa agli altri tipi come segue:

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

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

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

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

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

Si introduce ora la nozione di istanza legale di uno schema con regole come una interpretazione nella quale le interpretazioni di classi e tipi vengono vincolate alla loro descrizione e sono valide le relazioni di inclusioni stabilite tramite le regole.

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

Si noti come, in una istanza legale $\mathcal{I}$, l'interpretazione di un nome di classe base (C) *è contenuta* nell'interpretazione della sua descrizione, mentre per un nome di classe virtuale (V), come per un nome di tipo-valore (T), l'interpretazione *coincide* con l'interpretazione della sua descrizione. In altri termini, mentre l'interpretazione di una classe base è fornita dall'utente, l'interpretazione di una classe virtuale è calcolata sulla base della sua descrizione. Pertanto, in particolare, le classi virtuali possono essere utilizzate per rappresentare sia viste che query effettuate sulla base di dati. Se lo schema è aciclico, l'istanza di classi virtuali e tipi può essere univocamente calcolata

---

$d\in\mathbf{D} : \mathcal{I}_{\mathbf{B}}[d]=\{d\}$.

sulla base della sua descrizione, in presenza di classi virtuali cicliche questa computazione non è univoca: fissata un'interpretazione per le classi base, una classe virtuale ciclica può avere più di una interpretazione possibile. Tra tutte queste interpretazioni, in OLCD viene selezionata come istanza legale quella *più grande*, cioè viene adottata una semantica di *massimo punto fisso* (gfp). Le definizioni formali di tale semantica e le motivazioni della scelta sono discusse in [25] e verranno considerate nel Capitolo 5.

Nel proseguio del capitolo si farà riferimento a schemi e query acicliche.

### 2.2.3 Sussunzione ed Espansione Semantica di un tipo

Introduciamo la definizione di relazione di sussunzione in uno schema con regole.

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

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

È importante notare che la relazione di sussunzione rispetto al solo schema $\sigma$, cioè considerando $\mathbf{R} = \emptyset$, denotata con $\sqsubseteq_\sigma$, è simile alle relazioni di *subtyping* o *refinement* tra tipi definite nei *CODMs* [17, 80]. Questa relazione può essere calcolata attraverso una comparazione sintattica sui tipi; per il nostro modello tale l'algoritmo è stato presentato in [25].

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

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

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

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

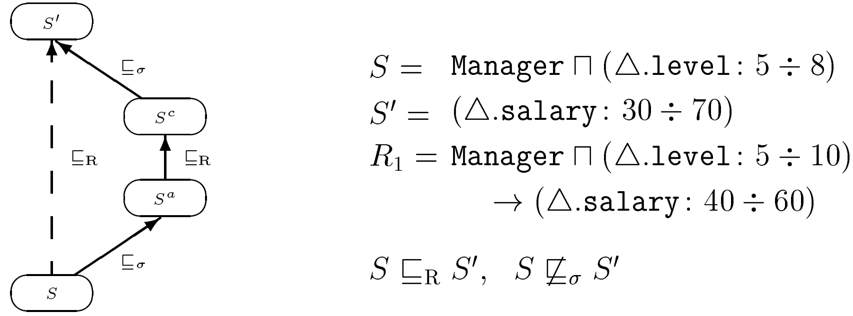



Figura 2.1: Relazione di sussunzione dovuta alle regole

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

L'espressione $EXP(S)$ permette di esprimere la relazione esistente tra $\sqsubseteq_{\mathrm{R}}$ e $\sqsubseteq_\sigma$: per ogni $S, S' \in \mathbf{S}$ si ha $S \sqsubseteq_{\mathrm{R}} S'$ se e solo se $EXP(S) \sqsubseteq_\sigma S'$. Questo significa che, dopo aver determinato l'espansione semantica, anche la relazione di sussunzione nello schema con regole può essere calcolata tramite l'algoritmo presentato in [25].

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

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

e poniamo $\tilde{\Gamma} = \Gamma^i$, dove $i$ è il più piccolo intero tale che $\Gamma^i = \Gamma^{i+1}$. L'esistenza di $i$ è garantita dal fatto che il numero di regole è finito e una regola non può essere applicata più di una volta con lo stesso cammino ($S \not\sqsubseteq_\sigma (p\colon S^c)$). Si può dimostrare che, per ogni $S \in \mathbf{S}$, EXP(S) è effettivamente calcolabile tramite $\tilde{\Gamma}(S)$. L'algoritmo, presentato nella sezione successiva ed pubblicato in [14, 28, 15], rappresenta uno dei contributi originali della presente tesi ed è basato sulle tecniche DL applicate al formalismo OLCD: trasformazione in forma canonica, controllo di consistenza e calcolo della relazione di sussunzione.

### 2.2.4 Algoritmi

In questa sezione, presentiamo brevemente, gli algoritmi che ci permettono di calcolare in forma polinomiale sia la *sussunzione* che l'*espansione semantica* di uno schema. Per raggiungere tale obiettivo, introduciamo la nozione di *forma canonica* di uno schema.

$\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$ denota l'insieme delle *descrizioni canoniche di tipo* ($\overline{S}$, $\overline{S}'$, ...) su $\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}}$, con $\tilde{\mathbf{N}} = \overline{\mathbf{C}} \cup \tilde{\mathbf{V}} \cup \tilde{\mathbf{T}}$, che sono costruite rispettando la seguente regola sintattica[6] (con $k \geq 0$, $i \geq 0$ e $\tilde{S} \in \{\top\} \cup \mathbf{B} \cup \tilde{\mathbf{V}} \cup \tilde{\mathbf{T}}$ e $\overline{C}_i \in \overline{\mathbf{C}}$. )

$$
\begin{array}{rcl}
\overline{S} & \rightarrow & \tilde{S} \\
& | & \forall\{\tilde{S}\} \\
& | & \exists\{\tilde{S}\} \\
& | & \exists\{S_1\} \sqcap \exists\{S_2\} \ldots \sqcap \forall\{S\} \\
& | & [a_1 \colon \tilde{S}_1, \ldots, a_k \colon \tilde{S}_k] \\
& | & \bigsqcap_i \overline{C}_i \sqcap \triangle \tilde{S}.
\end{array}
$$

Uno *schema canonico* $\nu$ è uno schema su $\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$. Ne segue immediatamente la definizione di Schema Canonico Equivalente.

**Definizione 4 (Schema Canonico Equivalente)** *Dato uno schema $\sigma$ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ si dice che uno schema canonico $\nu$ su $\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$ con $\mathbf{N} \subseteq \tilde{\mathbf{N}}$ è* equivalente *a $\sigma$ se solo se per ogni istanza legale $\mathcal{I}$ di $\sigma$ esiste un'istanza legale $\mathcal{I}'$ di $\nu$ e* viceversa *tale che $\mathcal{I}[N] = \mathcal{I}'[N], \forall N \in \mathbf{N}$.*

Da questa definizione segue che le relazioni semantiche in due schemi equivalenti sono le stesse: di conseguenza, il calcolo della sussunzione e il controllo di coerenza in uno schema arbitrario possono essere effettuati considerando lo schema canonico equivalente.

Ogni schema può essere effettivamente trasformato in uno schema canonico equivalente. I punti principali sono i seguenti: ogni classe base $C$ è trasformata in una classe virtuale (espressa come la congiunzione della descrizione di $C$, $\sigma(C)$, e di un nuovo nome di classe base $\overline{C}$, detto *atomo fittizio*, appartenente a $\overline{\mathbf{C}}$); la descrizione di ogni nome di tipo $N$ è trasformata sostituendo i nomi dai quali $N$ eredita con le rispettive descrizioni, espandendo i nomi di tipi valore e rinominando i nuovi tipi che si ottengono risolvendo le intersezioni. Quindi, all'inizio si ha $\tilde{\mathbf{V}} = \mathbf{C} \cup \mathbf{V}$ e

[6] Si assume che per $i = 0$ la regola $\bigsqcap_i \overline{C}_i \sqcap \triangle \tilde{S}$ produca il tipo $\triangle \tilde{S}$. Inoltre, per semplicità, i nomi di tipo non vengono segnati, cioè gli elementi di $\tilde{\mathbf{N}}$ vengono indicati ancora con $N$.

$\tilde{\mathbf{T}} = \mathbf{T}$ e questi insiemi crescono man mano che si effettuano le trasformazioni.

Per calcolare la sussunzione, si introduce una relazione $\leq$ tra i nomi di tipi ($\leq \subseteq \tilde{\mathbf{N}} \times \tilde{\mathbf{N}}$) e si definisce una relazione di sussunzione basata su $\leq$, denotata con $\overline{S} \sqsubseteq_{\leq} \overline{S}'$, come segue:

$$\begin{array}{rcl}
B \sqsubseteq_{\leq} B' & \text{sse} & B \sqcap B' = B \\
\forall\{N\} \sqsubseteq_{\leq} \forall\{N'\} & \text{sse} & N \leq N' \\
\sqcap_i \exists\{N_i\} \sqcap \forall\{N\} \sqsubseteq_{\leq} \forall\{N'\} & \text{sse} & N \leq N' \\
\sqcap_i \exists\{N_i\} \sqcap \forall\{N\} \sqsubseteq_{\leq} \sqcap_j \exists\{N_j'\} \sqcap \forall\{N'\} & \text{sse} & N \leq N' \wedge \forall j \exists i\colon N_i \leq N_j' \\
[\ldots, a_i\colon N_i, \ldots] \sqsubseteq_{\leq} [\ldots, a_j'\colon N_j', \ldots] & \text{sse} & \forall j \exists i\colon (a_i = a_j' \wedge N_i \leq N_j') \\
\sqcap_i \tilde{C}_i \sqcap \triangle N \sqsubseteq_{\leq} \sqcap_j \tilde{C}_j' \sqcap \triangle N' & \text{sse} & N \leq N' \wedge \forall j \exists i\colon \tilde{C}_i = \tilde{C}_j'.
\end{array}$$

Abbiamo ora tutti gli strumenti per definire l'aloritmo di *sussunzione*:

**Algoritmo 1 (Sussunzione)**
*Sia* $\leq^0 = \tilde{\mathbf{N}} \times \tilde{\mathbf{N}}$ *e definisci* $\leq^{i+1}$ *nel seguente modo:*

$$\leq^{i+1} = \left\{ (N, N') \in \tilde{\mathbf{N}} \times \tilde{\mathbf{N}} \,\middle|\, \nu(N) \sqsubseteq_{\leq^i} \nu(N') \vee N = \bot \vee N' = \top \right\}.$$

*Calcola* $\leq^0, \ldots, \leq^i, \ldots$ *finchè* $\leq^i = \leq^{i+1}$ *e poni* $\lesssim_\nu = \leq^i$.

Poichè $\tilde{\mathbf{N}}$ è finito, esiste sempre un numero naturale $n$ tale che $\leq^n = \leq^{n+1}$; il massimo valore di $n$ è $|\tilde{\mathbf{N}}|^2$, quindi l'algoritmo di sussunzione è polinomiale in $\nu$. L'algoritmo è corretto e completo: per ogni $N, N' \in \tilde{\mathbf{N}}$: $N \sqsubseteq_\sigma N'$ sse $N \lesssim_\nu N'$.

Calcolata la *sussunzione* dello schema, possiamo introdurre l'algoritmo per il calcolo dell'espansione semantica, per la descrizione del quale servono le seguenti definizioni. Dati $N$ e $N' \in \mathbf{N}$, diciamo che $N$ *dipende direttamente* da $N'$ sse $N'$ è contenuto nella descrizione canonica di $N, \nu(N)$. La chiusura transitiva, denotata con $\mathbf{CT}^+(N)$ è definita come segue:

**Definizione 5 (Chiusura di un tipo)** *Dato lo schema canonico* $\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$*:*
$\mathbf{CT}^0(N) = \{N\} \cup \{N_1 \in \tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}}) \setminus (\mathbf{B} \cup \overline{\mathbf{C}}) \mid N$ *dipende direttamente da* $N_1\}$
$\mathbf{CT}^{i+1}(N) = \mathbf{CT}^i(N) \cup \{N_1 \in \tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}}) \setminus (\mathbf{B} \cup \overline{\mathbf{C}}) \mid N_2 \in \mathbf{CT}^i(N) \wedge N_2$ dipende direttamente da $N_1\}$
$\mathbf{CT}^+(N) = \mathbf{CT}^{\bar{i}}(N)$, *dove* $\bar{i}$ *è il piú piccolo intero tale che* $\mathbf{CT}^{\bar{i}}(N) = \mathbf{CT}^{\bar{i}+1}(N)$.

Infine, vediamo l'algoritmo:

**Algoritmo 2 (Espansione Semantica)**

- **Acquisizione del tipo** $N$**:** $((\nu(N_1), GS(N_1)^7), \forall\ N_1 \in \mathbf{CT}^+(N))$
- **Iterazione**: $\forall\ N_1 \in \mathbf{CT}^+(N)$, *calcolo dell'espansione semantica di* $N_1$, $\tilde{\Gamma}(N_1)$:

$$\Gamma^i(N_1) \quad = \quad \begin{cases} N_1 \sqcap \bigsqcap_k\ R_k^c & \forall R_k:\ R_k^a \in GS(N_1), R_k^c \notin GS(N_1) \\ N_1 & altrimenti \end{cases}$$

*itera finchè* $\Gamma^{i+1}(N_1) = \Gamma^i(N_1)$, *poni* $\tilde{\Gamma}(N_1) = \Gamma^i(N_1)$.

- **Restituzione**: $(N',\ \sigma(N') = \tilde{\Gamma}(N))$

Complessità: il calcolo della funzione $\Gamma^i(N_1)$ ha complessità pari a $|\tilde{\mathbf{N}}|$ + $|\tilde{\mathbf{N}}_R^2|$ (con $|\tilde{\mathbf{N}}_R|$ numero di regole dello schema), poichè occorre ispezionare dapprima GS($\mathrm{N}_1$) che contiene al più $|\tilde{\mathbf{N}}|$ elementi e poi controllare per ciascun antecedente (al più $|\tilde{\mathbf{N}}_R|$) la condizione di applicabilità (al più $|\tilde{\mathbf{N}}_R|$ confronti). Il numero di iterazioni compiute su $\Gamma^i(N_1)$ sono pari, nel caso peggiore, al numero di regole $|\tilde{\mathbf{N}}_R|$, poichè una regola può essere applicata una sola volta sullo stesso *path*; mentre il ciclo esterno sugli elementi di $\mathbf{CT}^+(N)$ viene ripetuto al più $|\tilde{\mathbf{N}}|$ volte. Ne consegue una complessità di caso peggiore di tipo polinomiale pari a $|\tilde{\mathbf{N}}^2|$ $|\tilde{\mathbf{N}}_R|$ + $|\tilde{\mathbf{N}}|$ $|\tilde{\mathbf{N}}_R^3|$.

## 2.3 Esempio applicativo

Riprendiamo l'esempio descritto all'inizio di questo capitolo e vediamo a quali vantaggi può portare l'uso della logica OLCD.
L'attività di inferenza iniziale consiste nel determinare l'espansione semantica di ogni classe dello schema. Questo permette di rilevare relazioni *isa* non esplicitate nella definizione delle classi stesse. Ad esempio, nel calcolo dell'espansione semantica della classe `DMaterial` si rileva che tale classe è sussunta (implica) l'antecedente della regola R1: congiungendone quindi il conseguente si determina che `DMaterial` è una specializzazione di `SMaterial`. Nel caso in cui vengano applicate più regole durante l'espansione semantica di una

[7] GS denota Generalization Set, cioè l'insieme dei tipi generalizzazione di un dato tipo, calcolato dall'algoritmo di sussunzione.

classe, le relazioni *isa* ricavate sono meno ovvie di quella appena descritta. Ad esempio, nel calcolo dell'espansione semantica della classe `DStorage` viene applicata prima la regola R1 e successivamente la regola R2, concludendo che `DStorage` è una specializzazione di `SStorage`.

L'altra applicazione della OLCD riguarda l'ottimizzazione semantica delle interrogazioni, anch'essa basata sull'espansione semantica, in modo da specializzare le classi interessate dall'interrogazione. Prendiamo ad esempio la seguente query, espressa in *OQL*, che vuole selezionare dallo schema tutti i magazzini che contengono materiali che hanno tutti rischio maggiore o uguale a 15.

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

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

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

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

# Capitolo 3

# Il prototipo ODB-Tools

In questo capitolo viene presentato ODB-Tools, uno strumento software per la validazione di schemi e l'ottimizzazione semantica di interrogazioni per le Basi di Dati Orientate agli Oggetti (OODB), che sfrutta le potenzialità di Internet e del linguaggio JAVA per la rappresentazione grafica *on-line* dei risultati [19].

Gli algoritmi operanti in ODB-Tools sono basati su tecniche di inferenza che sfruttano il calcolo della *sussunzione* e la nozione di *espansione semantica* di interrogazioni per la trasformazione delle query al fine di ottenere tempi di risposta inferiori. Il primo concetto è stato introdotto nell'area dell'Intelligenza Artificiale, più precisamente nell'ambito delle Logiche Descrittive, il secondo nell'ambito delle Basi di Dati. Questi concetti sono stati studiati e formalizzati in OLCD (*Object Languages with Complements allowing Descriptive cycles*), una logica descrittiva per basi di dati introdotta nel Cap. 2.

Come interfaccia verso l'utente esterno è stata scelta la proposta ODMG-93 [57], utilizzando il linguaggio ODL (Object Definition Language) per la definizione degli schemi ed il linguaggio OQL (Object Query Language) per la scrittura di query. Entrambi i linguaggi sono stati estesi per la piena compatibilità al formalismo OLCD.

Il capitolo è organizzato nel seguente modo: la sezione 3.1 presenta il formalismo ODL tramite un esempio di riferimento, le estensioni al linguaggio ODL, e le funzioni inferenziali di validazione e calcolo della sussunzione. Nella sezione 3.4 viene descritta l'architettura del prototipo e nella sezione 3.5 vengono descritti la struttura della versione on-line e un esempio completo di utilizzo del prototipo.

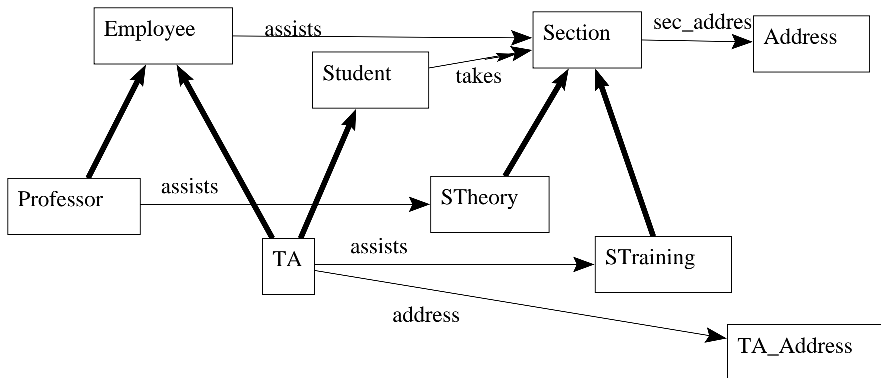



Figura 3.1: Rappresentazione grafica dello schema Università

## 3.1 Il linguaggio ODL

Object Definition Language (ODL) è il linguaggio per la specifica dell'interfaccia di oggetti e di classi nell'ambito della proposta di standard ODMG-93. Il linguaggio svolge negli ODBMS le funzioni di definizione degli schemi che nei sistemi tradizionali sono assegnate al Data Definition Language. Le caratteristiche fondamentali di ODL, al pari di altri linguaggi basati sul paradigma ad oggetti, possono essere così riassunte:

- definizione di classi e tipi valore;
- distinzione tra intensione ed estensione di una classe di oggetti;
- definizione di attributi semplici e multivalore (set, list, bag);
- definizione di relazioni e relazioni inverse tra classi di oggetti;
- definizione della signature dei metodi.

La sintassi di ODL estende quella dell'Interface Definition Language, il linguaggio sviluppato nell'ambito del progetto Common Object Request Broker Architecture (CORBA) [102].

### 3.1.1 Schema di esempio

Introduciamo ora uno schema che esemplifica la sintassi ODL utilizzata dal nostro prototipo.

L'esempio descrive la realtà universitaria rappresentata dai docenti, gli studenti, i moduli dei corsi e le relazioni che legano tra loro le varie classi. In particolare esiste la classe dei moduli didattici (`Section`), distinti in

moduli teorici (`STheory`) e di esercitazione (`STraining`). La popolazione universitaria è costituita da dipendenti (`Employee`) e studenti (`Student`). Un sottinsieme dei dipendenti è costituito dai professori (`Professor`) ed esiste la figura del docente assistente (`TA`) che è al tempo stesso sia un dipendente che uno studente dell'Università. I moduli vengono seguiti da studenti e sono tenuti da `TA`, o, solo per quelli teorici, da professori.

In figura 3.1 viene rappresentato lo schema risultante, in cui le classi sono rappresentate da rettangoli, le relazioni di aggregazione tra classi tramite frecce (semplici per quelli monovalore, doppie per quelli multivalore), e le relazioni di ereditarietà sono rappresentate graficamente da frecce più marcate.

Nella tabella 3.1 è riportata la definizione in ODL delle interfacce delle classi dello schema.

### 3.1.2 ODL e il modello OLCD

In questa sezione sono discusse le differenze tra ODL e il linguaggio OLCD.

Come visto nel Capitolo 2, OLCD prevede una ricca struttura per la definizione dei *tipi atomici*: sono presenti gli *integer*, *boolean*, *string*, *real*, *tipi mono-valore* e sottoinsiemi di tipi atomici, quali ad esempio intervalli di interi. A partire dai tipi atomici si possono definire *tipi valore*, attraverso gli usuali costruttori di tipo definiti nei CODM, quali *tuple*, *insiemi* e *tipi classe*, che denotano insiemi di oggetti con una identità ed un valore associato. Ai tipi può essere assegnato un nome, con la distinzione tra nomi per tipi-valore e nomi per tipi-classe (chiamati semplicemente *classi*). In tale assegnamento, il tipo può rappresentare un insieme di condizioni necessarie e sufficienti, o un insieme di condizioni solo necessarie. L'ereditarietà, sia semplice che multipla, è espressa direttamente nella descrizione di una classe tramite l'operatore di *congiunzione*.

Andiamo ora ad analizzare i concetti propri del formalismo OLCD che non trovano riscontro nello standard ODMG-93 e le relative estensioni da noi proposte al linguaggio ODL standard.

- tipo base `range`.

  In OLCD è possibile esprimere intervalli di interi, utilizzati per l'ottimizzazione semantica di query in OQL con predicati di confronto. Per ovviare alla mancanza abbiamo introdotto nella sintassi ODL il costrutto `range`. Ad esempio, si può introdurre un modulo di teoria avanzato `ADVSTheory` come un modulo di teoria `STheory` il cui livello è opportunamente elevato (compreso tra 8 e 10):

```
struct Address
{   string city;
    string street; };

interface Section ()
{   attribute string number;
    attribute  Address sec_address; };

interface STheory : Section()
{   attribute integer level; };

interface STraining : Section()
{   attribute string features; };

interface Employee  ()
{   attribute string name;
    attribute unsigned short annual_salary;
    attribute string domicile_city;
    attribute Section assists; };

interface Professor: Employee  ()
{   attribute string rank;
    attribute STheory assists; };

interface TA: Employee, Student ()
{   attribute STraining assists;
    attribute struct TA_Address { string city;
                                  string street;
                                  string tel_number; }
                                  address; };

interface Student ()
{    attribute string name;
     attribute integer student_id;
     attribute set<Section> takes; };
```

Tabella 3.1: Esempio ODL: database universitario

```
interface ADVSTheory : STheory()
{       attribute  range {8, 10}  level;   };
```

- viste o classi virtuali (`view`).

  OLCD introduce la distinzione tra *classe virtuale*, la cui descrizione rappresenta condizioni necessarie e sufficienti di appartenenza di un oggetto del dominio alla classe (corrispondente quindi alla nozione di vista) e *classe base (o primitiva)*, la cui descrizione rappresenta solo condizioni necessarie (corrispondente quindi alla classica nozione di classe). In altri termini, l'appartenenza di un oggetto all'interpretazione di una classe base deve essere stabilita esplicitamente, mentre l'interpretazione delle classi virtuali è calcolata sulla base della loro descrizione.

  Le classi base sono introdotte tramite la parola chiave `interface`, mentre per le classi virtuali si introduce il costrutto `view` che specifica la classe come virtuale seguendo le stesse regole sintattiche della definizione di una classe base.. Ad esempio, la seguente dichiarazione:

  ```
  view Assistant: Employee, Student ()
  { attribute Address address; };
  ```

  introduce la classe virtuale `Assistant` che rappresenta tutti gli oggetti appartenenti sia alla classe studenti che dipendenti e che, in piú, hanno un indirizzo rappresentato dalla struttura `Address` definita dagli attributi via e città.

- Vincoli di Integrità.

  I vincoli di integrità (Integrity Constraints) permettono la formulazione dichiarativa di un insieme rilevante di regole di integrità sotto forma di funzioni *if then* i cui antecedenti e conseguenti sono esprimibili come tipi OLCD. È possibile, in tal modo, dichiarare correlazioni fra proprietà strutturali della stessa classe o condizioni sufficienti per il popolamento di sottoclassi di una classe data.

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

  Una regola di integrità è dichiarata attraverso la seguente sintassi:

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

Le `condizioni`, antecedente e conseguente, hanno la medesima forma e sono costituite da una lista di espressioni booleane in `and` tra loro; all'interno di una condizione, attributi e oggetti sono identificati mediante la *dot notation*. Ad esempio, introduciamo le seguenti regole di integrità nello schema di riferimento:

```
rule rule_1 forall X in Professor: X.rank = "Full"
   then X.annual_salary >= 60000 ;

rule rule_2 forall X in Employee:  X.annual_salary < 30000
   then X in TA  ;

rule rule_3 forall X in Professor:
                        exists S in X.teaches: S.level > 7
   then X.rank = "Full" ;
```

La `rule_1` stabilisce che un professore di *rango* "Full" (cioé un professore ordinario) abbia un salario superiore a \$60000. La `rule_2` impone l'appartenenza alla classe degli assistenti a tutti i dipendenti con salario inferiore a \$30000. La `rule_3` asserisce che i professori che insegnano *almeno* una sezione di livello superiore a 7 hanno il rango pari a "Full".

Essendo un linguaggio di definizione di schemi del tutto generale, ODL contiene alcuni costrutti aggiuntivi che non sono presenti nel formalismo OLCD.

### Restrizioni di OLCD

- OLCD prevede soltanto la definizione di relazioni di aggregazione unidirezionali, quindi non è rappresentata l'operazione di inversione.
- Non sono supportati i tipi `enum` e `union`.
- In OLCD la parte intensionale ed estensionale di una classe sono referenziate dallo stesso nome.
- ODL permette una suddivisione gerarchica dello schema mediante l'uso dei *moduli*, mentre lo schema OLCD è piatto.

- Nell'attuale realizzazione del traduttore, le costanti possono essere solo dei *letterali*. Non possono essere ad esempio espressioni algebriche o valori strutturati costanti.

### 3.1.3 Validazione e Sussunzione

Uno dei problemi principali che il progettista di una base di dati deve affrontare è quello della consistenza delle classi introdotte nello schema. Infatti, molti modelli e linguaggi di definizione dei dati sono sufficientemente espressivi da permettere la rappresentazione di classi inconsistenti, cioè classi che non potranno contenere alcun oggetto della base di dati. Tale eventualità sussiste anche in OLCD: ad esempio, la possibilità di esprimere intervalli di interi permette la dichiarazione di classi con attributi omonimi vincolati a intervalli disgiunti. Il prototipo rivela durante la fase di validazione dello schema come inconsistente una eventuale congiunzione di tali classi. Ad esempio, introduciamo un modulo di teoria fondamentale `FSTheory` (con un livello compreso tra 2 e 6) e un modulo di teoria intermedio `ISTheory` che eredita sia da quello fondamentale che da quello avanzato `ADVSTheory`:

```
interface FSTheory : STheory()
{       attribute  range {2, 6}  level;   };

interface ISTheory : FSTheory, ADVSTheory() { };
```

La classe `ISTheory` risulta essere inconsistente in quanto il suo attributo `level` ha come dominio l'intersezione dei due intervalli disgiunti specificati in `FSTheory` e `ADVSTheory`.

Un'altra fonte di inconsistenza nella dichiarazione di classi deriva da attributi omonimi vincolati a strutture differenti, come nel seguente esempio:

```
interface New_STraining : STraining()
{  attribute  struct New_Address { string street;
                                   struct City { string name;
                                                 string state; }
                                                 city; }
                                    sec_address; };
```

Il concetto di *sussunzione* esprime la relazione esistente tra due classi di oggetti quando l'appartenenza di un oggetto alla seconda comporta necessariamente l'appartenenza alla prima. La relazione di sussunzione può essere calcolata automaticamente tramite il confronto sintattico tra le descrizioni delle classi; l'algoritmo di calcolo è stato presentato in [26]. Poichè accanto

alle relazioni di ereditarietà definite esplicitamente dal progettista possono esistere altre relazioni di specializzazione implicite, queste possono essere esplicitate dal calcolo della relazione di sussunzione presenti nell'intero schema: il prototipo, dopo aver verificato la consistenza di ciascuna classe, determina tali relazioni di specializzazione implicite fornendo un valido strumento inferenziale per l'utente progettista della base dei dati.

Ad esempio, il tipo `Address` sussume il tipo `TA_Address` in quanto tutti gli attributi di `Address` sono contenuti in `TA_Address` con lo stesso dominio e `TA_Address` aggiunge altri attributi. Come conseguenza di questa relazione di sussunzione si ha che la classe virtuale `Assistant` sussume la classe `TA`, in quanto tutti gli attributi di `Assistant` sono contenuti in `TA` ed inoltre l'attributo `address` di `Assistant` è definito su `Address` che sussume il dominio (`TA_Address`) dell'attributo `address` della classe `TA`. È importante notare che per l'individuazione della relazione `Assistant` sussume `TA`, la classe `Assistant` sia virtuale: infatti solo considerando la descrizione di `Assistant` come condizione sufficiente (e necessaria) si può affermare che ogni oggetto di `TA` è anche in `Assistant`.

## 3.2 Ottimizzazione semantica delle interrogazioni

L'ottimizzazione semantica di una interrogazione utilizza il processo di *espansione semantica*, attraverso il quale viene generata una interrogazione equivalente che incorpora ogni possibile restrizione non presente nell'interrogazione originale e tuttavia *logicamente implicata* dallo schema globale del database (classi + tipi + regole d'integrità). L'algoritmo di calcolo, di complessità polinomiale, è stato proposto in [15] e rappresenta il motore del modulo di ottimizzazione del prototipo.

Il linguaggio di interrogazione utilizzato è compatibile con OQL (Object Query Language), proposto in ODMG-93, sia per l'input delle query che per l'output restituito dopo l'ottimizzazione. Una interrogazione espressa in OQL potrà beneficiare del processo di ottimizzazione semantica se può essere tradotta, in modo equivalente, in una espressione di tipo di OLCD: vista la ricchezza di tale formalismo, un insieme rilevante di interrogazioni può essere ottimizzato. In particolare, si riescono a trattare interrogazioni riferite ad una singola classe con condizioni espresse sulla gerarchia di aggregazione. Tuttavia, poiché il linguaggio di interrogazione OQL è più espressivo del formalismo OLCD, introduciamo, seguendo l'approccio proposto in [41], una separazione ideale dell'interrogazione in una parte *clean*, che può essere rap-

presentata come tipo in OLCD, e una parte *dirty*, che va oltre l'espressività del sistema di tipi; l'ottimizzazione semantica sarà effettuata solo sulla parte *clean*.

Allo scopo di evidenziare le trasformazioni effettuate dal processo di ottimizzazione semantica, l'interrogazione viene riportata in uscita differenziandone graficamente i vari fattori sulla base della seguente classificazione:

- fattori specificati dall'utente e non modificati (`stampati`)
- fattori modificati o introdotti dall'ottimizzatore (sottolineati)
- fattori dirty, specificati dall'utente ma non trattati dall'ottimizzatore (*corsivati*)

Vediamo alcuni esempi di interrogazioni che illustrano il nostro metodo (lo schema di riferimento è quello introdotto in 3.1.1):

$Q_1$ : "Seleziona i dipendenti con stipendio annuale inferiore a $18000 che sono assistenti di una sezione A".

In ODB-Tools la query può essere scritta in linguaggio OQL come segue:

```
Q1 select   *
from        employee as E
where       annual_salary < 18000
and         assists in ( select  S
                         from    Section as S
                         where   number = "A" )
```

L'espansione semantica dell'interrogazione, che si ottiene applicando la regola di integrità `rule_2` ed usando la descrizione di `TA`, porta alla seguente interrogazione equivalente:

```
select  *
from    TA as E
where   annual_salary < 18000
and     assists in ( select  S
                     from    STraining as S
                     where   number = "A" )
```

Questo esempio mostra un'effettiva ottimizzazione dell'interrogazione, indipendente da ogni specifico modello di costo, dovuta alla sostituzione delle classi presenti nell'interrogazione con loro specializzazioni. Infatti, sostituendo `employee` con `TA` e `Section` con `STraining` si riduce l'insieme di oggetti da controllare per individuare il risultato dell'interrogazione.

Per illustrare la separazione di un'interrogazione nella parte *clean* e *dirty*, modifichiamo la precedente interrogazione:

$Q_2$ : "Seleziona i dipendenti con stipendio annuale inferiore a $18000 che sono assistenti di una sezione A tenuta nella stessa città in cui essi vivono".

```
select   *
from     employee as E
where    annual_salary < 18000
and      assists in ( select  S
                      from    Section as S
                      where   number = "A"
                      and     domicile_city ! = S.sec_address.city)
```

Il nuovo fattore rappresenta la parte *dirty* dell'interrogazione che non può essere tradotta nel formalismo OLCD (essenzialmente perchè esprime un confronto tra due attributi); mentre la parte *clean*, che corrisponde a $Q_1$, può essere ottimizzata come illustrato in precedenza:

```
select   *
from     TA as E
where    annual_salary < 18000
and      assists in ( select  S
                      from    STraining as S
                      where   number = "A"
                      and     domicile_city ! = S.sec_address.city )
```

L'ultimo esempio che viene riportato ha lo scopo di evidenziare sia l'uso dei quantificatori nella formulazione delle interrogazioni sia l'applicazione di più regole nel processo di ottimizzazione semantica:

$Q_3$ : "Seleziona i professori con stipendio annuale inferiore a $35000 che insegnano *almeno* un corso di livello 9".

```
select   *
from     Professor as P
where    annual_salary < 35000
and      exists S in P.teaches : S.level = 9
```

A tale interrogazione è applicabile la regola `rule_3`, che, introducendo il fattore (`rank` = "Full") rende applicabile anche la regola `rule_1`, la quale aggiunge il fattore (`annual_salary` >= 60000), incompatibile con (`annual_salary` < 35000) specificato nella interrogazione originale. Pertanto tale interrogazione viene rilevata come inconsistente rispetto allo schema del database e quindi, senza effettuare alcun accesso, si può stabilire che il suo risultato è l'insieme vuoto.

## 3.3 Confronti con altri lavori

Normalmente, nei sistemi per la rappresentazione della conoscenza basati sulle logiche descrittive [105] è prevista una componente che, utilizzando la sussunzione e le classi virtuali, effettua l'acquisizione e il controllo della gerarchia delle classi. In vari lavori [27, 13, 34, 26] questa idea è stata estesa anche agli schemi per basi di dati. Inoltre, nei sistemi CANDIDE [13] e CLASSIC [34] le interrogazioni sono espresse nello stesso linguaggio utilizzato per descrivere la base di conoscenza. In questo modo, le tecniche di inferenza possono essere applicate anche alle interrogazioni. D'altra parte, l'idea dell'ottimizzazione delle interrogazioni basata sul calcolo della sussunzione tra una interrogazione e una classe virtuale dello schema è stata importata anche nell'ambito OODB [41]. Gli autori definisco un formalismo per rappresentare parte dello schema e delle interrogazioni che risolve il problema della sussunzione in tempo polinomiale. D'altra parte, in questi sistemi non vengono normalmente trattate regole di integrità e quindi la sussunzione (e, di conseguenza, l'acquisizione dello schema delle classi e l'ottimizzazione semantica delle interrogazioni) è significativa solo in presenza di classi virtuali. Il nostro metodo estende l'ambito di questi lavori in quanto, definendo uno schema con regole, permette di ottenere nuove superclassi che sono sia classi virtuali che classi base.

Numerosi lavori relativi all'ottimizzazione semantica sono presenti in letteratura. In particolare, il metodo dell'espansione semantica, attraverso il quale vengono aggiunte all'interrogazione, in modo esaustivo, tutte le restrizioni possibili, è stato utilizzato anche in altri lavori [96, 90]. In [96] gli autori descrivono un metodo di ottimizzazione semantica, riferito al modello relazionale, basato sui grafi che sfrutta due tipi di vincoli di integrità: vincolo di sottoinsieme e vincolo di implicazione. In particolare, in questo lavoro viene chiamata espansione semantica quella fase della trasformazione che ha l'obiettivo di incorporare ogni restrizione o join che non era presente nell'interrogazione originale. Le principali fasi successive all'espansione semantica sono l'eliminazione di relazioni e l'eliminazione di join. In [90] è presentato un efficiente algoritmo di ottimizzazione semantica delle interrogazioni per un OODB che considera anche vincoli di integrità. L'approccio è simile all'espansione semantica in quanto tutte le trasformazioni sono applicate all'interrogazione; il compito di scegliere le trasformazioni utili è ritardato fino a quando tutte le trasformazioni sono state considerate. In questo modo le trasformazioni che sono state applicate non precludono trasformazioni future e l'ordine delle trasformazioni è irrilevante, pertanto gli autori affermano che l'algoritmo è di complessità polinomiale. L'algoritmo è riferito ad un particolare genere di schema e di vincoli di integrità (in forma di clausole di Horn)

espressi con un modello che non è ad oggetti complessi. Del modello non viene data la semantica e non viene presentata nessuna verifica di correttezza delle trasformazioni semantiche considerate.

Nell'ambito OODB, alcuni lavori propongono l'uso della relazione di ereditarietà tra classi per effettuare l'ottimizzazione semantica delle interrogazioni. In [99] è presentata una metodologia per l'elaborazione delle interrogazioni orientate ad oggetti. Al fine di ottimizzare le interrogazioni vengono introdotte regole di trasformazione distinte in *algebriche* e *semantiche*. Le trasformazioni algebriche si limitano a riscrivere una interrogazione, creando espressioni equivalenti basate sulla sostituzione testuale e il "pattern matching". Le trasformazioni semantiche trasformano l'interrogazione sulla base della definizione delle classi e della gerarchia di ereditarietà. In [58] viene investigata l'ottimizzazione di una classe di interrogazioni *congiuntive* per OODB. Per tali interrogazioni viene studiato e caratterizzato il problema del *contenimento* tra interrogazioni e della *minimizzazione* dello spazio di ricerca per le variabili coinvolte nell'interrogazione. Le informazioni utilizzate sono quelle "classiche" espresse in uno schema: il tipo degli attributi di una classe, la relazione di specializzazione e l'assunzione che un'oggetto sia istanziato in una ed na usola classe. In entrambi i lavori appena citati la classe delle interrogazioni e la categoria di problemi presi in esame sono più ampie di quelle considerate nel presente articolo. D'altra parte, oltre al fatto che il modello di riferimento di questi lavori non è ad oggetti complessi, dal punto di vista proprio dell'ottimizzazione semantica non sono stati considerati vincoli di integrità espressi come regole che risultano molto utili nella trasformazione di una interrogazione. Inoltre, l'espansione semantica e, più in generale, tutte le trasformazioni proposte con il nostro metodo sono *corrette*, cioè producono espressioni semanticamente equivalenti. Tale proprietà è formalmente garantita dalla precisa semantica sia del modello dei dati che della relazione di sussunzione. Il problema di una formale verifica della correttezza delle trasformazioni semantiche è stato affrontato solo in pochi lavori, tra i quali [32]. In questo lavoro ogni trasformazione è considerata singolarmente e viene dimostrata la sua correttezza.

Lavori più recenti di elevata generalità sia per il modello di dati che per il linguaggio di interrogazione sono stati proposti per basi di conoscenza orientate ad oggetti [98, 76]. In tali lavori viene utilizzato un linguaggio logico del primo ordine sia per la descrizione degli oggetti, in termini di classi, leggi deduttive e vincoli di integrità, sia per il linguaggio di interrogazione. Le opportunità di ottimizzazione derivano essenzialmente dalla natura deduttiva della base di conoscenza. In [98] viene descritta una tecnica di trasformazione semantica che utilizza regole di riscrittura per trasformare una interrogazione; inoltre vengono presentati alcuni criteri per ordinare i fattori nella interro-

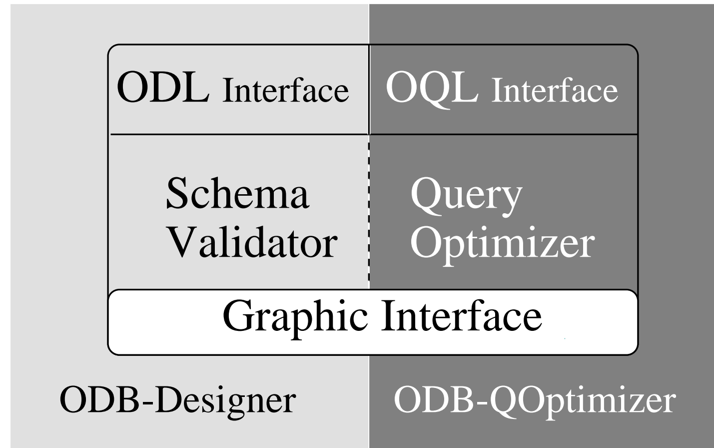



Figura 3.2: Architettura di ODB-Tool

gazione trasformata. In [76] i principi di astrazione di un modello orientato ad oggetti sono espressi come assiomi in una teoria del primo ordine. Tali assiomi vengono utilizzati quindi per l'ottimizzazione semantica; ad esempio, l'assioma che esprime il tipo di un attributo può essere utilizzato per eliminare, in una interrogazione, un predicato di appartenenza ad una classe. Le interrogazioni sono considerate classi virtuali, la cui semantica è definita tramite regole deduttive. In questo modo una interrogazione può essere classificata nella gerarchia delle classi ottenendo un'ulteriore ottimizzazione nel caso in cui l'interrogazione è una sottoclasse di una *vista materializzata*.

## 3.4 Architettura di ODB-Tools

ODB-Tool, sviluppato al Dipartimento di Scienze dell'Ingegneria dell'Università di Modena, è un prototipo software per la validazione di schemi e l'ottimizzazione di interrogazioni in ambiente OODB. L'architettura, mostrata in figura 3.2, presenta i vari moduli integrati che definiscono un ambiente *user-friendly* basato sul linguaggio standard ODMG-93. L'utente inserisce gli schemi in linguaggio ODL e le query in OQL ottenendo come risultato la validazione dello schema, l'ottimizzazione dell'interrogazione (in OQL) e la rappresentazione grafica della gerarchia di ereditarietà e di aggregazione dello schema.

Vediamo in dettaglio la descrizione di ciascun modulo:

- **ODL Interface**
  È il modulo di input degli schemi. Accetta la sintassi descritta in

appendice B e trasforma le classi in descrizioni native del formalismo OLCD.

- **OQL Interface**
  È il modulo di input e output delle interrogazioni. Utilizza il linguaggio OQL sia per l'input che per l'output della query ottimizzata, che viene trasformata in descrizioni del formalismo OLCD. I predicati booleani in output sono differenziati a seconda del proprio significato:

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

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

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

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

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

I moduli di interfaccia, validazione ed ottimizzazione sono stati realizzati in linguaggio C, utilizzando il compilatore gcc 2.7.2, i generatori flex 2.5 e bison 1.24, mentre il modulo di rappresentazione grafica è stato realizzato in JAVA, utilizzando il compilatore JDK 1.1. La piattaforma utilizzata è una SUN SPARCSTATION 20, con sistema operativo Solaris 2.5.

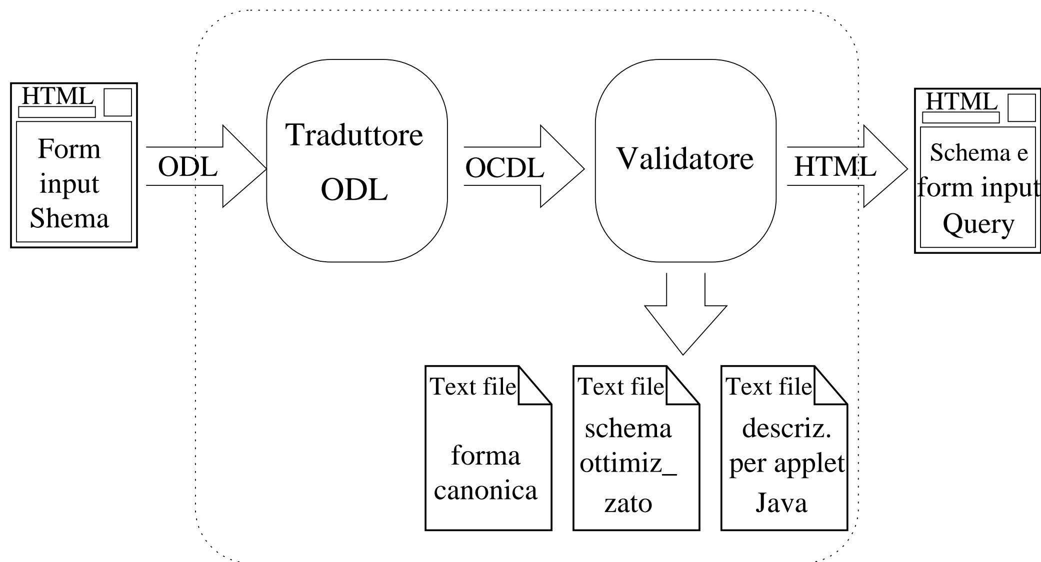



Figura 3.3: Validazione dello schema

## 3.5 Struttura del sito WWW e demo di esempio

In questa sezione presentiamo le operazioni che l'utente compie durante l'utilizzo di ODB-Tools e le corrispondenti elaborazioni del sistema, considerando come esempio lo schema presentato nella sezione 3.1.1.

ODB-Tools è disponibile su internet (`http://sparc20.dsi.unimo.it`) con possibilità di inviare schemi e interrogazioni da macchine remote sia per la validazione degli schemi che per l'ottimizzazione di interrogazioni.

Collegandosi al sito è possibile raggiungere la pagina di DEMO del tool, che richiede in ingresso uno schema di database scritto in linguaggio ODL. È possibile specificare lo schema sia inviando un file generato localmente, sia digitando direttamente lo schema in un apposita box visualizzata dal browser. Lo schema cosí specificato viene analizzato sintatticamente, tradotto in sintassi OLCD e inviato come input per il modulo di validazione. Al termine di

Netscape: Input query form ( form)

File Edit View Go Bookmarks Options Directory Window Help

Location: http://sparc20.dsi.unimo.it/cgi-bin/prototipo/sc_form.script

**VALIDATION RESULT**

Schema validation OK

You can see

- Validator execution time , ODL schema , validator output .
- OCDL schema , canonical form , subsumption form .

Student, Address, Section, Employee, TA_Address, Assistant, rule_3a, rule_2a, STraining, STheory, Professor, TA, rule_2c, ADVSTheory, rule_3c, rule_1a, rule_1c

Order Isa Rel Rules

Figura 3.4: Pagina di presentazione dello schema validato

Figura 3.5: Finestra di presentazione degli attributi di una classe

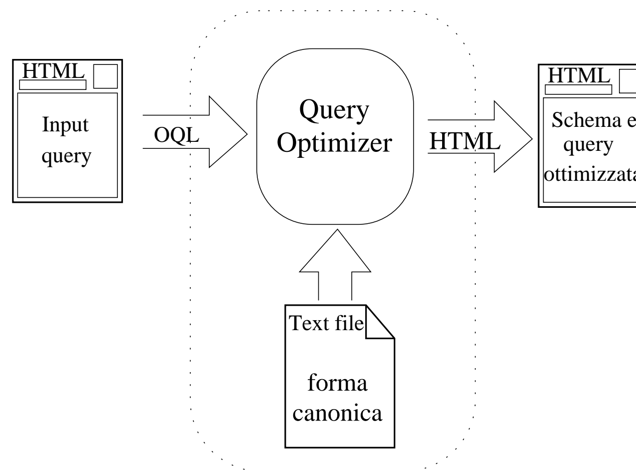



Figura 3.6: Ottimizzazione di una query

tale processo viene generata una pagina HTML che contiene la rappresentazione grafica dello schema validato (tramite un'applet JAVA) e una form per l'inserimento di query sullo schema (vedi fig. 3.3). In caso di errori sintattici o di rivelazione di classi inconsistenti, durante il processo di validazione viene creata un pagina HTML che informa dettagliatamente l'utente al riguardo.

Ad esempio, considerando lo schema di sezione 3.1.1 a cui è stata aggiunta la vista `Assistant`, il risultato è riportato nella figura 3.4, dove é possibile notare la nuova posizione della vista `Assistant` nella gerarchia di ereditarietà. Agendo sull'elemento che rappresenta una classe è possibile aprire una finestra che mostra l'elenco degli attributi (ad es. nella figura 3.5 viene mostrata la finestra relativa alla classe `employee`).

Dalla pagina generata in fase di validazione è possibile eseguire interrogazioni in OQL sullo schema che vengono analizzate dall'ottimizzatore seman-

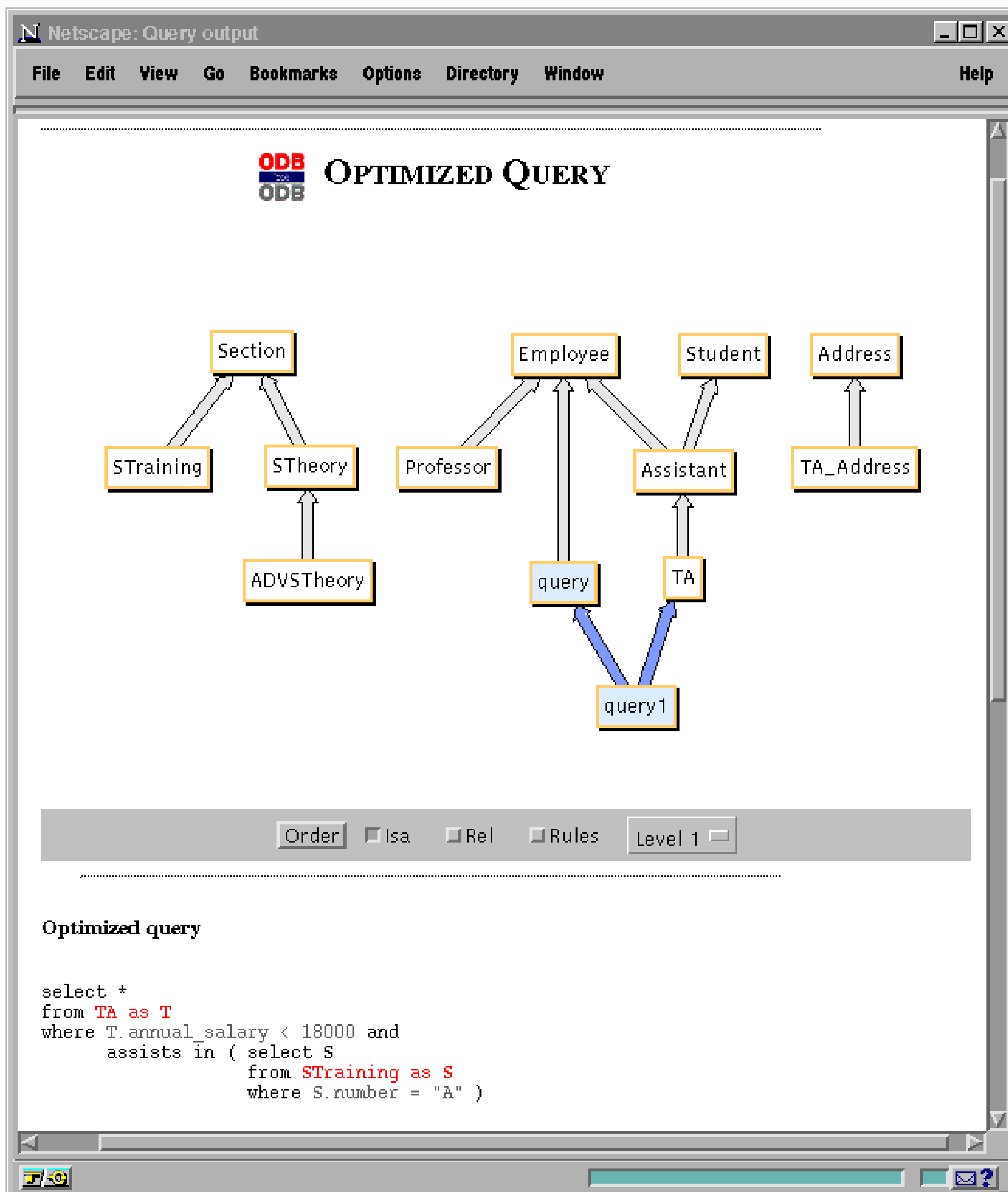



Figura 3.7: Lo schema Università prima della validazione

tico. Al solito, è possibile specificare un'interrogazione sia inviando un file che digitando direttamente la query nella dialog box. La query inviata viene elaborata dal modulo di ottimizzazione e al termine viene mostrata una pagina HTML con la rappresentazione della tassonomia dello schema contenente la query ottimizzata e la descrizione in sintassi OQL della query ottimizzata (vedi fig. 3.6). Nel caso in cui il processo di ottimizzazione sia avvenuto in più passi è possibile mostrare la classificazione della query all'interno dello schema al termine di ciascun passo.

Come esempio utilizziamo la query $Q_1$ vista nella sezione 3.2, che viene ottimizzata fornendo come uscita la pagina riportata nella figura 3.7, in cui è mostrata la interrogazione originale introdotta dall'utente (`query`) e la interrogazione ottenuta dall'ottimizzazione al primo (e in questo caso unico) passo del calcolo dell'espansione semantica (`query1`).

## 3.6 Risultati sperimentali

L'efficienza del prototipo è stata valutata verificando la diminuzione dei costi di esecuzione delle query ottimizzate negli OODBMS commerciali. In particolare sono stati utilizzati i sistemi O2 ver. 4.2 ed UniSQL ver. 3.1.2.

Per l'analisi sperimentale si è costruito un database relativo ad uno schema comprendente sei classi e, mediamente, due regole associate a ciascuna classe. Sono state successivamente scelte cinque query legate allo schema per le quali si è determinata l'ottimizzazione semantica e relativo costo di trasformazione del prototipo. Per ciascuna interrogazione si è calcolato il tempo di esecuzione su quattro istanze di DB differenti, le cui statistiche sono riportate in tabella 3.2.

| | **Class A** | **SubCl. A** | **Class B** | **SubCl. B** | **Class C** | **SubCl. C** |
|---|---|---|---|---|---|---|
| DB1 | 164 | 56 | 1151 | 534 | 1700 | 950 |
| DB2 | 1555 | 725 | 1718 | 691 | 2600 | 1481 |
| DB3 | 3220 | 1621 | 3436 | 1382 | 4900 | 2912 |
| DB4 | 4869 | 2515 | 5154 | 2073 | 8200 | 4668 |

Tabella 3.2: Dimensioni delle Classi del Database

Infine, per ciascun OODBMS, abbiamo riportato i valori medi del rapporto tra il costo di esecuzione per le query originali e quelle ottimizzate in funzione degli oggetti presenti nel database. Tali valori sono graficati in fig. 3.8: come è facilmente prevedibile il risparmio di costo è funzione crescente del numero di oggetti presenti nel DB. Tale considerazione è più evidente

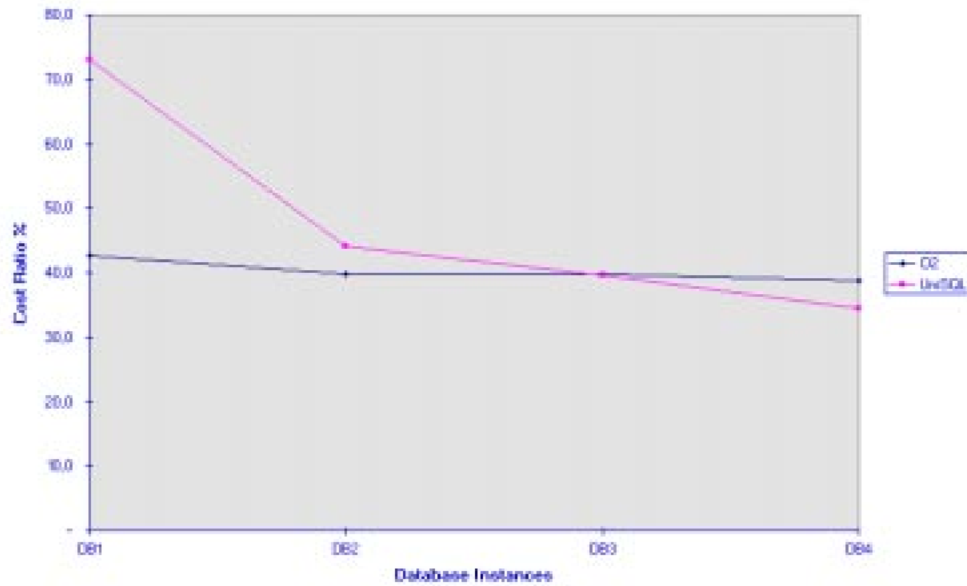



Figura 3.8: Rapporto tra Costo Ottimizzato ed Originale in funzione della dimensione de DB

nel caso di UniSQL, mentre per O2 il risparmio di costo è da subito rilevante (già con DB1) per poi rimanere pressochè costante. Per una descrizione più dettagliata si veda `http://sparc20.dsi.unimo.it/risultati.html`

# Capitolo 4

# Introduzione dei metodi in OLCD

Il formalismo OLCD, presentato in 2, é stato sviluppato a partire dalle Logiche Descrittive per la rappresentazione degli aspetti strutturali di classi di oggetti.

Un limite della espressivitá di OLCD rispetto ad una piena modellazione di classi object-oriented deriva dalla impossibilitá di definire metodi (o *operations* nella terminologia ODMG-93) all'interno dello schema, che sono invece ampiamente utilizzati nella realizzazione di applicazioni con l'ausilio degli OODBMS.

In questo capitolo il formalismo OLCD viene esteso introducendo la possibilitá di esprimere i metodi sia nella definizione delle classi che all'interno di vincoli di integritá. Viene inoltre descritto il controllo di consistenza dei metodi dello schema, confrontato con le proposte attualmente presenti in letteratura.

Infine viene illustrato brevemente un esempio dell'applicazione nell'ambito di impianti di automazione industriale.

### 4.0.1 Introduzione di uno schema con operazioni

Nella sezione 2.2 è stata presentata la logica descrittiva, che esprime le descrizioni strutturali di uno schema di basi di dati ad oggetti. Viene ora presentata l'integrazione con la definizione delle operazioni[1].

Sia **N** l'insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che **A**, **B**, e **N** siano a due a due disgiunti. **N** è partizionato in tre insiemi

[1] d'ora in poi le parole *operazione* e *metodo* saranno usati come sinonimi

**C**, **V** e **T**, dove **C** contiene i nomi per *tipi-classe base* $(C, C' \ldots)$, **V** contiene i nomi per *tipi-classe virtuali* $(V, V' \ldots)$, e **T** contiene i nomi per *tipi-valori* $(t, t', \ldots)$.

Si introducono ora i tipi adatti a rappresentare lo schema delle operazioni.

Sia **Att**={in,out,inout} l'insieme dei nomi dei tipi che indicano l' "attributo del parametro", (denotati da $Att_1$,$Att_2$,... e chiamati brevemente "tipi-parametro"). Tale insieme deve essere chiaramente disgiunto dall'insieme dei nomi dei tipi **N**.

I tipi-parametro servono per introdurre nel sistema dei tipi **S** il tipo "signature di operazione", denotato con $S_s$, che è un particolare tipo record:

$$[p_1\mathbf{:}S_1 \sqcap \mathbf{Att}_1\mathbf{,}\ p_2\mathbf{:}S_2 \sqcap \mathbf{Att}_2\mathbf{,}\ldots\mathbf{,}p_k\mathbf{:}S_k \sqcap \mathbf{Att}_k]$$

dove:
$k$ è il numero dei parametri della operazione
$p_1$**,**$p_2$**,**...**,**$p_k$ sono i nomi dei parametri formali della operazione
$S_i \in \mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})\ \forall \mathbf{i=1,...k}$ sono i tipi dei parametri formali.

Il tipo signature rappresenta interamente gli argomenti della signature di una operazione.

Nel seguito, il sistema dei tipi costruito partendo dall'insieme dei nomi dei tipi **N**, unito con **Att** ed esteso con il tipo "signature di operazione" verrà indicato, per semplicità, nuovamente come $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$.

Per i tipi-parametro si deve estendere la definizione di interpretazione e di istanza: in particolare, per estendere il concetto di istanza, occorre estendere la funzione $\sigma$ anche a tali nomi. Queste estensioni sono basate sul fatto che un tipo-parametro deve essere considerato un nome di tipo (sia classe che tipo-valore) primitivo e senza alcuna descrizione, quindi

- Interpretazione:

$$\mathcal{I}[\mathbf{Att}] \subseteq \mathcal{V}(\mathcal{O})$$

- Schema:

$$\sigma(\mathbf{Att}) = \top$$

- Istanza:

$$\mathcal{I}[\sigma(\mathbf{Att})] \subseteq \mathcal{I}[\mathbf{Att}]$$

Si introduce ora la definizione di operazione; informalmente, le operazioni sono rappresentate da quadruple contenenti:

1. la classe sulla quale sono definite
2. il nome
3. la signature, rappresentata da un tipo simile alle ennuple con l'aggiunta degli attributi dei parametri, anch'essi rappresentati da tipi
4. il tipo di ritorno

**Definizione 6 (Operazione)** *Una Operazione è una quadrupla* $(C_d, \mathbf{m}, T_s, \mathbf{R})$*, dove:*
$C_d \in \mathbf{C} \cup \mathbf{V}$ *rappresenta la classe su cui è definita l'operazione*
$\mathbf{m} \in \mathbf{M}$ *è il nome della operazione*
$T_s \in \mathbf{S_s}$ *è il tipo della signature della operazione*
$\mathbf{R} \in \mathbf{S}$ *è il tipo del risultato della operazione*

I linguaggi di programmazione orientati agli oggetti prevedono la possibilità, nota come "overloading", di dichiarare due operazioni definite sulla medesima classe con lo stesso nome. Per permettere l'overloading, la coppia nome e classe di definizione non sono sufficienti ad identificare univocamente una operazione, ma é invece necessario aggiungere l'intera signature della operazione per poterla identificare.

Per generalizzare il discorso anche al caso in cui una operazione venga introdotta direttamente in una rule (problema affrontato nella prossima sezione) è stata fatta la seguente scelta: per ogni operazione dichiarata nella definizione di una classe il sistema crea un identificatore univoco, detto *identificatore di operazione*, e vi associa la quadrupla che rappresenta l'operazione stessa. Si noti che una stessa operazione (stessa quadrupla) può essere associata a più identificatori.

Formalmente, si introduce $\mathbf{I_M}$, l'insieme numerabile degli identificatori delle operazioni (denotati da $Op_1, Op_2, \ldots$) e si rappresenta questa corrispondenza tramite il concetto di *Schema di operazioni.*

**Definizione 7 (Schema di operazioni)** *Uno* schema di operazioni $\sigma_{\mathbf{M}}$ *su* $\mathbf{I_M}$ *è una funzione totale*
$\sigma_{\mathbf{M}} : \mathbf{I_M} \rightarrow (\mathbf{C} \cup \mathbf{V})\ X\ \mathbf{M}\ X\ \mathbf{S_s}\ X\ (\mathbf{S})$
*che associa ad ogni identificatore la descrizione della propria operazione.*

### 4.0.2 Operazioni e Vincoli di Integritá

Nel paragrafo precedente è stato introdotto lo schema di operazioni. Quando viene dichiarata una operazione all'interno di una rule, essa viene inserita nello schema delle operazioni mediante una quadrupla ed un identificatore, come avviene per le operazioni dichiarate all'interno delle interface delle classi.

Esiste comunque una differenza fondamentale tra le operazioni dichiarate all'interno della interface delle classi e quelle dichiarate all'interno delle rule. All'interno di una classe l'operazione viene dichiarata, mentre all'interno delle rule le operazione vengono "invocate", viene cioé utilizzata una loro istanza. In generale esistono infinite possibilità di invocare una operazione, coincidenti a tutti i possibili valori assunti dai parametri in ingresso alla operazione stessa. É necessario poter distinguere il tipo di ritorno della operazione in base alla chiamata effettuata, poichè nel momento in cui la rule sarà eseguita si otterranno risultati diversi. Per fare questo è necessario introdurre un nome che appartenga al sistema dei tipi che univocamente rappresenti quella chiamata. Questo è possibile realizzarlo utilizzando gli identificatori di operazioni definiti nel precedente paragrafo. Utilizzando gli identificatori di operazioni all'interno delle rule, e ponendosi l'obiettivo di mantenere distinte i diversi usi delle medesime operazioni si ottiene una funzione $\sigma_{\mathbf{M}}$ che mette due in relazione due diversi identificatori con la medesima quadrupla.

In una rule contenente un'operazione si deve affrontare anche il problema della sua traduzione in un tipo del formalismo OLCD. Nel seguito viene presentato un esempio per chiarire quali informazioni dell'operazione vengono riportate nella rule tradotta e quali informazioni vengono invece riportate nello schema delle operazioni.

Data la seguente porzione di schema:

```
// --------- Circuito
interface Circuito : Elemento_primario
( extent Circuiti
```

```
  keys codice_circuito
)
{
  attribute string tipo_linea;
  attribute string tipo_circuito;
  attribute string tipo_alimentazione;
  attribute range{1,6} tensione;
  attribute range{1,2} frequenza;
  relationship set<Elemento_elettrico> alimenta
       inverse Elemento_elettrico::alimentato;
};


// --------- Elemento_elettrico

interface Elemento_elettrico : Elemento
( extent Elementi_elettrici
)
{
  attribute range{0.0,+inf} corrente_assorbita_per_mille;
  relationship Circuito alimentato
       inverse Circuito::alimenta;
  relationship list<Morsetto> possiede
       inverse Morsetto::fa_parte;
  relationship set<Contatto> contatti
       inverse Contatto::fa_parte;
};

// --------- Elemento_calcolato

interface Elemento_calcolato : Elemento_elettrico
( extent Elementi_calcolati
)
{
  attribute string tipo_elemento_calcolato;
  relationship set<Elemento_primario> deriva
       inverse Elemento_primario::comanda;
};

// --------- Filo
```

```
interface Filo : Elemento_calcolato
( extent Fili
)
{
  attribute range{1,2} tipo_filo;
   // 1 significa 'di fase'
   // 2 significa 'di neutro'
  attribute range{0.0,+inf} sezione;
  attribute range{0.0,+inf} corrente_impiego;
};
```

Si supponga di voler esprimere la proposizione "se un filo fa parte di un circuito di potenza allora la sezione del filo è dato dalla corrente_impiego/2.5".

Per esprimere una condizione di questo tipo occorre estendere la sintassi delle regole di ODL per ammettere la possibilita' di descrivere le operazioni.
Per la regola in questione, denominata "Fil2", si scriverà:

```
rule Fil2 forall X in Filo: X.alimentato.tipo_circuito="potenza"
     then X.sezione=
          range{0.0,50.0} dividi(real X.corrente_impiego,real 2.5);
```

dove seguendo la sintassi propria di ODMG-93 per le operazioni, il nome dell'operazione "dividi" viene preceduto dal risultato dell'operazione stessa.

Consideriamo ora la rappresentazione di questa regola secondo il formalismo OLCD.
L'antecedente è una espressione completamente traducibile in OLCD:

Fil2a : $\mathit{Filo} \sqcap \triangle[\mathit{alimentato}\colon \triangle[\text{tipo_circuito} : \text{"potenza"}]]$

Del conseguente, invece, riportiamo in OLCD solo il tipo del risultato dell'operazione intersecato ad un nuovo tipo che rappresenta in modo univoco il risultato della specifica funzione "dividi"

Fil2c : $\mathit{Filo} \sqcap \triangle[\mathit{sezione} : 0.0 \div 50.0 \sqcap \mathit{Tipo1}]$

dove "Tipo1" deve essere un nome di tipo (sia classe che tipo-valore) primitivo senza alcuna descrizione, distinto da tutti gli altri nomi di tipo utilizzati nello schema, il cui scopo è quello di evitare che il sistema confonda un generico tipo "range{0.0,50.0}" dal particolare tipo risultato dell'operazione "dividi" richiamata con particolari parametri attuali.

In base a quanto detto in precedenza sull'univocità degli identificatori di operazioni, possiamo indicare come *Tipo1* l'identificatore $Op1$, e quindi scrivere
Fil2c : $\mathit{Filo} \sqcap \triangle[\mathit{sezione} : \ 0.0 \div 50.0 \sqcap \mathit{Op1}]$

A livello formale di modello OLCD, questo comporta il dover considerare anche gli identificatori di operazioni come tipi del modello, e in particolare, come nomi di tipo (sia classe che tipo-valore) primitivi senza alcuna descrizione; si deve quindi ripetere il discorso fatto per i tipi-parametro.

Sinteticamente: un identificatore di operazione viene considerato come un nome di tipo, con

- Interpretazione:

$$\mathcal{I}[Op] \subseteq \mathcal{V}(\mathcal{O})$$

- Schema:

$$\sigma(Op) = \top$$

- Istanza:

$$\mathcal{I}[\sigma(Op)] \subseteq \mathcal{I}[Op]$$

In questo modo, la rule
Fil2 :
$\mathit{Filo} \sqcap \triangle[\mathit{alimentato}\colon \triangle[\text{tipo_circuito} : \ \text{"potenza"}]]$
$\rightarrow$
$\mathit{Filo} \sqcap \triangle[\mathit{sezione} : \ 0.0 \div 50.0 \sqcap \mathit{Op1}]$

risulta essere formalmente una rule espressa sul sistema di tipi esteso.

Dall'altra parte, *Op1* appartiene all'insieme $\mathbf{I_M}$ e permette, mediante la funzione $\sigma_{\mathbf{M}}$ di accedere alla quadrupla

**(Filo,dividi,[param1:real⊓in,param2:real⊓in],range{0.0,50.0})**

che rappresenta l'operazione "dividi".

La dichiarazione di una rule contenente una diversa invocazione della operazione dividi comporta la creazione di un nuovo identificativo.

## 4.1 Principio di covarianza e controvarianza

Dopo aver introdotto nel formalismo la possibilitá di dichiarare le operazioni si cerca ora di definire le regole per il controllo di consistenza degli schemi che contengono le operazioni. Vediamo dapprima quali sono gli orientamenti delle proposte nell'ambito della ricerca per poi definire la nostra proposta.

In letteratura sono presentati due approcci complementari per controllare la consistenza delle interfaccie dei metodi:

1. Un primo approccio applica ai parametri delle operazioni il medesimo principio applicato agli attributi delle classi, ossia il principio di covarianza.

2. Un secondo approccio applica ai parametri delle operazioni il principio opposto a quello applicato agli attributi delle classi, ossia il principio di controvarianza.

Di seguito viene riportata la definizione di sottotipo secondo la teoria dei tipi di dati (E. Bertino e L.D. Martino [31]; Bruce and Wegner 1986 [40]; Cardelli 1984 [48]; Albano et al. 1985 [5]).

**Definizione 8 (sottotipo)** *Un tipo $t$ è sottotipo del tipo $t'$ ($t \leq t'$) se:*
*(1) le proprieta' di $t'$ sono un sottoinsieme di quelle di $t$*
*(2) per ogni operazione $m'_t$ di $t'$ esiste la corrispondente operazione $m_t$ di $t$ tale che:*
*(a) $m_t$ e $m'_t$ hanno lo stesso nome*
*(b) $m_t$ e $m'_t$ hanno lo stesso numero di argomenti*
*(c) l'argomento $i$ di $m'_t$ è un sottotipo dell'argomento $i$ di $m_t$* **(regola di controvarianza)**
*(d) $m_t$ e $m'_t$ restituiscono un valore o entrambi non hanno alcun parametro di ritorno*
*(e) se $m_t$ e $m'_t$ restituiscono un valore allora il tipo del risultato di $mt$ è un sottotipo del tipo del risultato di $m'_t$* **(regola di covarianza)**

In base a tale definizione è valido il *principio di sostituibilita'*, secondo il quale una istanza di un sottotipo puo' essere sostituita da un supertipo in ogni contesto nel quale quest'ultimo puo' essere legalmente usato.

Altri gruppi di ricerca, come gli autori che hanno sviluppato il sistema $O_2$ [10], hanno adottato la regola di covarianza anche sui singoli parametri delle operazioni: in questo modo però si possono produrre degli errori in fase di run-time a fronte di una correttezza sintattica.

In Bruce and Wegner 1986 [40] si riporta una dimostrazione rigorosa di quanto affermato sopra, sottolineando soprattutto la differenza esistente tra i tipi degli attributi e i tipi dei parametri formali delle operazioni.

Per capire questa affermazione si propone un semplice esempio:
si supponga di modellare l'aspetto geometrico dei punti del piano cartesiano. I punti sono degli oggetti definiti da due coordinate (x,y), e possono essere positivi oppure negativi.
I punti positivi sono contenuti interamente nel primo quadrante del piano cartesiano, mentre quelli negativi nel terzo quadrante.
Introduco una operazione denominata "disegna" con le due coordinate come parametri di input.

```
interface Punto
 (extent Punti)
{
 attribute range{-1000,1000} x;
 attribute range{-1000,1000} y;
 void disegna (in range{-1000,1000} px,in range{-1000,1000} py);
};
interface Punto_positivo : Punto
 (extent Punti_positivi)
{
 attribute range{0,1000} x;
 attribute range{0,1000} y;
 void disegna (in range{0,1000} px,in range{0,1000} py);
};
```

In questo esempio ho applicato la regola di covarianza anche sui due parametri dell'operazione (metodo di $O_2$). Supponiamo di invocare l'operazione disegna(-10,-10) su un generico oggetto della classe "Punto", se questo oggetto è anche istanza di "Punto_positivo" allora verra' eseguita

l'operazione nella classe piu' specializzata, causando un errore di run-time.
Da un punto di vista pratico questo esempio esclude il principio di covarianza e giustifica solo in parte il principio di controvarianza applicato ai parametri delle funzioni. Per evitare errori in fase di esecuzione delle operazioni, è sufficiente dichiarare il tipo dei parametri delle sottoclassi **uguale** a quello delle superclassi. Infatti nell'esempio riportato di seguito non vi è alcun motivo per ampliare il range {0,1000} dei parametri dell'operazione Punto_positivo::disegna.

```
interface Punto
 (extent Punti)
{
 attribute range{-1000,1000} x;
 attribute range{-1000,1000} y;
 void disegna (in range{-1000,1000} px,in range{-1000,1000} py);
};
interface Punto_positivo : Punto
 (extent Punti_positivi)
{
 attribute range{0,1000} x;
 attribute range{0,1000} y;
 void disegna (in range{-1000,1000} px,in range{-1000,1000} py);
};
```

In conclusione il principio di controvarianza è stato adottato per motivi di flessibilita', infatti risulta piuttosto vincolante imporre, nella signature delle operazioni, l'uso degli stessi tipi dichiarati nelle superclassi.

## 4.2 Controllo di consistenza in schemi con operazioni

In questa sezione viene definita la strategia per il controllo di consistenza degli schemi contenenti operazioni.
In particolare ci interessa controllare le situazioni in cui un dato metodo può produrre inconsistenze, dovute a invocazioni run-time utilizzando argomenti non definiti nella *signature* del metodo stesso.
Seguendo l'approccio proposto nella definizione del modello dei dati di $O_2$ [11], definiamo condizione di coerenza dello schema la *specializzazione*

*covariante* di un metodo. Ciò significa che nel nostro modello la dichiarazione coerente dei metodi coincide con la tipizzazione sicura (safe type) dei parametri contenuti nella signature, cioè:

ciascun parametro della signature del metodo può essere specializzato solo da termini i cui tipi sono "sottotipi" di quelli di partenza. Viene adottata la nozione di "sottotipi" ("subtypes") definita in [49].

Come descritto in [50], la *specializzazione covariante* garantisce la tipizzazione corretta dei metodi solo nel caso statico mentre non è sicura rispetto ad errori di run-time. Per prevenire errori di run-time è necessario utilizzare la regola di controvarianza per i parametri della signature [6], in modo tale che i tipi dei parametri del metodo padre sono sottotipi dei parametri dei metodi ereditati. Come estensione a OLCD andiamo a definire entrambe le regole di controllo di schema *variante* e *controvariante*.

1. **Definizione 9 (Schema covariante)** *Uno schema $\sigma_{\mathbf{M}}$ è covariante sse*
   *∀ coppia di operazioni (C,m,$T_s$,R),(C',m,$T'_s$,R') tali che*

   (a) *C' $\sqsubseteq_{\mathrm{R}}$ C,*

   (b) *R' $\sqsubseteq_{\mathrm{R}}$ R*

   (c) *$T_s$ e $T'_s$ hanno lo stesso numero di parametri*

   *è verificato che $T'_s \sqsubseteq_{\mathrm{R}} T_s$*

2. **Definizione 10 (Schema controvariante)** *Uno schema $\sigma_{\mathbf{M}}$ è controvariante sse*
   *∀ coppia di operazioni (C,m,$T_s$,R),(C',m,$T'_s$,R') tali che*

   (a) *C' $\sqsubseteq_{\mathrm{R}}$ C,*

   (b) *R' $\sqsubseteq_{\mathrm{R}}$ R*

   (c) *$T_s$ e $T'_s$ hanno lo stesso numero di parametri*

   *è verificato che $T_s \sqsubseteq_{\mathrm{R}} T'_s$*

Avendo a disposizione i due controlli di consistenza è possibile richiedere, in alternativa, uno dei due metodi o entrambi. Infatti i concetti di *covarianza* e *controvarianza* non sono antagonisti, bensì concetti distinti con la propria dignità in sistemi object oriented. L'indipendenza dei due meccanismi, mostrata in [50], potrebbe (e ancora meglio dovrebbe) essere integrata in un ambiente di controllo type-safe per lunguaggi object oriented.

## 4.3 Utilizzo nelle applicazioni industriali

L'introduzione relativa alla dichiarazione e controllo dei metodi risulta particolarmente utile durante il progetto di sistemi di automazione industriale, come dimostra l'applicazione reale realizzata in collaborazione con l'azienda AST di Soliera (Modena) per la progettazione di impianti elettrici industriali [93, 29].

### 4.3.1 Descrizione della problematica

In ambito industriale, cresce sempre più la richiesta e l'utilizzo dell'Information Technology come supporto alla progettazione e alla realizzazione degli impianti.
Nel caso di studio, si voleva realizzare uno strumento software di supporto e di aiuto alla stesura della documentazione tecnica e commerciale relativa alla progettazione della parte elettrica di un impianto di automazione industriale.
Il progettista di impianti elettrici industriali deve fornire ai propri clienti:

- lo **schema elettrico** dell'impianto
- la lista dei materiali utilizzati
- la collocazione specifica dei materiali nel luogo dove viene eseguita materialmente l'automazione (**schema topografico**)

Attualmente, chi opera nel settore della automazione industriale, trova numerosi pacchetti software che assistono il progettista nell'attività di disegno degli schemi elettrici, riconducibili ai sistemi CAD (Computer Aided Design) disponibili sul mercato.
Più difficile risulta la ricerca di strumenti in grado di svolgere sia le attività di progetto citate che quelle di catalogazione e reperimento dei materiali presenti in un impianto elettrico.

Da queste problematiche, è nata l'idea di progettare e realizzare un ambiente software di ausilio al progettista di impianti elettrici in grado di memorizzare e gestire tutte le informazioni di un impianto di automazione industriale, partendo dallo schema elettrico (che risulta essere sempre il documento principale a cui fare riferimento) fino ad arrivare a preventivi e consuntivi sui costi di progetto.
L'attivita' di progetto ha quindi riguardato lo studio di fattibilita', il progetto e la realizzazione di un'ambiente software denominato "**Elet-Designer**".

Alcune componenti di Elet-Designer sono state sviluppate con Microsoft Visual C++ in ambiente Microsoft Windows NT. L'ambiente software è basato su un database contenente tutti i dati relativi alla parte elettrica di un impianto di automazione industriale.
Il suddetto database è stato progettato utilizzando ODB-Tools, fornendo le estensioni necessarie al miglioramento dell'efficacia nell'aiuto al progettista di basi di dati complesse.
Questo aiuto si è evidenziato soprattutto nella realizzazione e nella validazione della base di dati di Elet-Designer Le principali fasi del lavoro hanno riguardato:

- analisi dei requisiti del dominio Elet
- estensione di ODB-Tools per rispondere alle nuove esigenze introdotte dal dominio Elet
- progettazione del database Elet con l'ausilio di ODB-Tools
- sviluppo di un prototipo di interfaccia tra il database e il progettista di impianti elettrici, denominato "**Prelet**"
- sviluppo di uno strato software, denominato "**Object-World**", che permette a Prelet di usare le potenzialita' di un OODBMS (Object Oriented Database Management System) pur avendo a disposizione un RDBMS (Relational Database Management System).
  In particolare la scelta di realizzare Object-World nasce dalla piu' consolidata e diffusa tecnologia relazionale rispetto alla piu' recente tecnologia ad oggetti.

### 4.3.2 Risultati conseguiti

Naturalmente, in questa sede, interessa mostrare e commentare la parte relativa all'ausilio fornito da ODB-Tools alla progettazione dello schema rispetto all'intero strumento software realizzato.

Per questo, in figura 4.3.2 viene riportata la rappresentazione grafica dello schema del database ottenuto utilizzando ODB-Tools, in cui le classi sono rappresentate con rettangoli in chiaro, mentre in colore più scuro sono rappresentati i vincoli di integrità (divisi in parte antecedente e conseguente).

L'utilizzo di ODB-Tools durante la fase di design ha assicurato il controllo di coerenza rispetto alle classi, i vincoli di integrità e la definizione dei metodi. Occorre sottolineare che nell'applicazione Prelet, a causa della

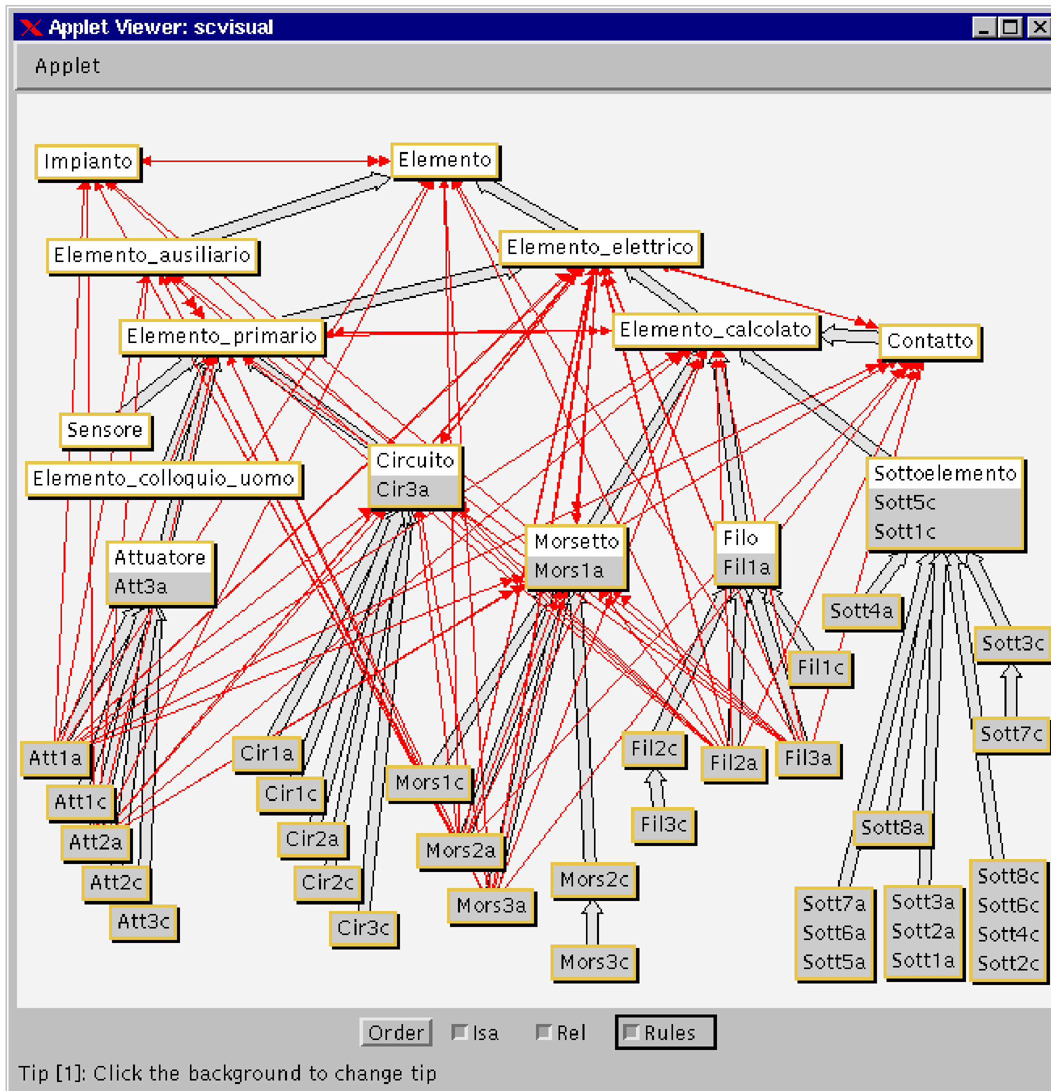



Figura 4.1: Schema del database con regole

natura stessa del problema, i vincoli di integrità e i metodi rappresentano più della metà della conoscenza informativa da introdurre nelo schema: infatti la quasi totalità dell'informazione è rappresentata da grandezze elettriche le cui relazioni sono note a priori e fissate dai documenti dell'Unione Europea (in particolare DIN VDE 0660, N. 102,107,200). Grazie alla possibilità di inserire in forma dichiarativa vincoli di integrità e metodi è possibile inserire a livello concettuale (direttamente nello schema) le regole di consistenza atte a garantire le normative elettriche vigenti.

Il controllo sullo schema covariante permette la verifica automatica della definizione di classi e metodi, che risulta molto utile rispetto alla riduzione dei tempi di sviluppo di progetto ed alla correttezza dello schema. In questo modo il progettista può concentrare la propria attenzione sugli aspetti semantici delle classi e dei vincoli di integrità, senza l'assillo di controllare manualmente la coerenza dello schema ad ogni minima modifica dello schema.

Al termine del progetto di schema del database è possibile fare alcune considerazioni:

- ODB-Tools è in grado di mostrare relazioni tra classi non esplicitate direttamente da progettista ma ottenute dalla gerarchia di ereditarietà ed aggregazione, fornendo uno strumento per l'analisi approfondita delle implicazioni tra classi e vincoli di integrità. Ad esempio osservando la rule "Att1a", è possibile notare, tra le altre, relazioni con le classi: "Impianto", "Contatto" e "Morsetto". Questo avviene poichè la rule "Att1a" eredita tutti gli attributi e le relazioni delle sue superclassi.

- ODB-Tools è in grado di rilevare le classi equivalenti. Esse si possono individuare facilmente, in quanto sono rappresentate nello stesso rettangolo nello schema grafico.
  Ad esempio ciò avviene per la classe primitiva "Sottoelemento" e per la classe virtuale "Sott5c". Poichè "Sott5c" è il conseguente di una regola, questa equivalenza dimostra la ridondanza di "Sott5c" e la sua sostituibilita' con la classe "Sottoelemento"

- È possibile rilevare gerarchie di ereditarietà (non esplicitate dal progettista) relative a vincoli di integrità, guidando il progettista nella modifica dello schema. Ad esempio abbiamo ereditarietà tra le regole "Mors3c" e "Mors2c". In questo caso gli oggetti che soddisfano la rule "Mors3c" soddisfano anche la rule "Mors2c".

# Capitolo 5

# Ottimizzazione semantica di schemi ciclici

L'algoritmo di espansione semantica introdotto in 2.2.3 ha validità per schemi e query aclicici, mentre non può essere utilizzato per ottimizzare query basate su schemi ciclici con regole. In questo capitolo viene analizzata la problematica introdotta dai cicli e viene presentato il nuovo algoritmo che effettua l'ottimizzazione semantica negli schemi ciclici. L'algoritmo è stato progettato, realizzato e verificato, ed è stato integrato nel sistema ODB-Tools.

## 5.1 Limiti dell'algoritmo di espansione semantica

Un assunto fondamentale dell'algoritmo di espansione semantica presentato nella sezione 2.2.3 riguarda la condizione di terminazione.

In particolare, la condizione di terminazione viene garantita dalla condizione sul numero di regole presenti sempre finito e dall'assunto che una regola non può essere applicata piú di una volta al tipo $S_1 \in \mathbf{CT}^+(S)$ grazie al controllo $S_k^c \notin GS(S_1)$.

In assenza di cicli questa condizione é comunque verificata, mentre in presenza di classi e regole cicliche puó ingenerare dei cicli infiniti. In altri termini, si puó verificare che una regola venga iterativamente applicata un numero infinito di volte. Alcuni esempi potranno chiarire meglio questo concetto.

$$(\triangle.a\colon > 10) \quad \rightarrow \quad (\triangle.c. \triangle .a\colon > 20)$$

L'esempio sopra introdotto riporta una regola che agisce su una classe ciclica. Analizzando l'algoritmo che realizza la forma canonica dello schema

(vedi 2.2.3), si vede facilmente che prevede la possibilitá di avere classi o regole cicliche. L'acquisizione dello schema dunque non comporta problemi di terminazione e viene calcolata correttamente la forma canonica.

A questo punto, tuttavia, una qualsivoglia interrogazione che faccia scattare la regola comporta la non terminazione dell'algoritmo di espansione semantica. Infatti, pur essendoci una sola regola (quindi un numero finito), questa stessa viene applicata un numero infinito di volte per via della ciclicitá della classe su cui agisce. Una query con questa caratteristiche é proposta di seguito.

$$\sigma_V^0(N) \quad = \quad (\triangle.a\colon > 45)$$

Il punto critico, per un algoritmo così concepito, risiede nel fatto che questo non confronta semanticamente i tipi per i quali si appresta a calcolare l'espansione con quelli giá considerati nel corso delle precedenti iterazioni. Puó accadere dunque che una stessa regola (o un insieme finito di regole) continui ad essere applicata all'infinito ad uno stesso tipo poiché ogni volta quest'ultimo non viene riconosciuto come giá espanso. É proprio in questo caso che la condizione di terminazione dell'algoritmo diventa inefficace.

Una situazione del tutto analoga ricorre introducendo il seguente schema:

$$\begin{array}{lcl} (\triangle.a\colon > 20) \sqcap (\triangle.b\colon > 30) & \rightarrow & (\triangle.c.\triangle.b\colon > 60) \\ (\triangle.a\colon > 41) \sqcap (\triangle.b\colon > 33) & \rightarrow & (\triangle.c.\triangle.a\colon > 75) \end{array}$$

e la query:

$$\sigma_V^0(N) \quad = \quad (\triangle.a\colon > 81) \sqcap (\triangle.b\colon > 99)$$

Al fine di sottolineare ulteriormente i limiti dell'algoritmo, possiamo calcolare l'espansione semantica delle regole e della query proposte. Con

riferimento alle regole il risultato sarebbe:

$$
\begin{aligned}
\nu^0(N_1) &= \triangle N_2 \\
\nu^0(N_2) &= (a\colon > 20) \sqcap (b\colon > 30) \\
\nu^0(N_3) &= \triangle N_4 \\
\nu^0(N_4) &= [c\colon N_5] \\
\nu^0(N_5) &= \triangle N_6 \\
\nu^0(N_6) &= (b\colon > 60) \\
\nu^0(N_{11}) &= \triangle N_{12} \\
\nu^0(N_{12}) &= (a\colon > 41) \sqcap (b\colon > 33) \\
\nu^0(N_{13}) &= \triangle N_{14} \\
\nu^0(N_{14}) &= [c\colon N_{15}] \\
\nu^0(N_{15}) &= \triangle N_{16} \\
\nu^0(N_{16}) &= (a\colon > 75)
\end{aligned}
$$

$$
\begin{aligned}
N_1 &\rightarrow N_3 \\
N_{11} &\rightarrow N_{13}
\end{aligned}
$$

Sviluppando il calcolo dell'espansione per la query:

$$
\begin{aligned}
\nu^0(N) &= \triangle N_{20} \\
\nu^0(N_{20}) &= (a\colon > 81) \sqcap (b\colon > 99) \\
\sigma_V^1(N) &= \sigma_V^0(N) \sqcap N_3 \Rightarrow \nu^1(N) = \triangle N_{21} \\
\nu^1(N_{21}) &= (a\colon > 81) \sqcap (b\colon > 99) \sqcap [c\colon N_5] \\
\sigma_V^2(N) &= \sigma_V^1(N) \sqcap N_{13} \Rightarrow \nu^2(N) = \triangle N_{22} \\
\nu^2(N_{22}) &= (a\colon > 81) \sqcap (b\colon > 99) \sqcap [c\colon N_{23}] \\
& \quad \text{dove } \ \sigma_V^2(N_{23}) = N_5 \sqcap N_{15} \Rightarrow \nu^2(N_{23}) = \triangle N_{24} \\
\nu^2(N_{24}) &= (a\colon > 75) \sqcap (b\colon > 60) \\
\sigma_V^3(N_{23}) &= \sigma_V^2(N_{23}) \sqcap N_3 \Rightarrow \nu^3(N_{23}) = \triangle N_{25} \\
\nu^3(N_{25}) &= (a\colon > 75) \sqcap (b\colon > 60) \sqcap [c\colon N_5] \\
\sigma_V^4(N_{23}) &= \sigma_V^3(N_{23}) \sqcap N_{13} \Rightarrow \nu^4(N_{23}) = \triangle N_{26} \\
\nu^4(N_{26}) &= (a\colon > 75) \sqcap (b\colon > 60) \sqcap [c\colon N_{27}] \\
& \quad \text{dove } \ \sigma_V^4(N_{27}) = N_5 \sqcap N_{15} \Rightarrow N_{27} = N_{23} \\
\nu^4(N_{26}) &= (a\colon > 75) \sqcap (b\colon > 60) \sqcap [c\colon N_{23}]
\end{aligned}
$$

Procedendo in questo modo, si é pervenuti ad una espansione corretta in un numero finito di passi. Questo perché, tenendo traccia dei tipi espansi, viene rilevato il ciclo allorché si cerca di introdurre un nuovo nome che corrisponde, in realtá, ad uno giá espanso ($N_{27} = N_{23}$).

Si impone allora la necessitá di rivistare l'algoritmo di espansione, allo scopo di consentire la trattazione dei cicli eventualmente presenti nelle definizioni dello schema.

## 5.2 Algoritmo di Espansione con soglia

L'algoritmo di espansione con soglia rappresenta la prima soluzione proposta in ordine alla trattazione di interrogazioni ricorsive. L'assunzione fondamentale del metodo parte dal presupposto che, per un dato schema con regole e per una qualsivoglia interrogazione, l'espansione semantica di quella interrogazione rispetto allo schema deve necessariamente concludersi in un numero finito di iterazioni e che tale numero é determinabile a priori. Di qui il nome algoritmo con soglia. La soglia rappresenta, dunque, il massimo numero di iterazioni previsto per realizzare l'espansione semantica.

Qualora l'espansione semantica dovesse richiedere un numero di passi superiore rispetto alla soglia calcolata, si tratterebbe sicuramente di una situazione ciclica.

Da quanto detto si puó facilmente dedurre che, al fine della determinazione dei cicli e, quindi, dell'affidabilitá stessa del metodo, risulta di fondamentale importanza il criterio in base al quale la soglia viene determinata.

Le considerazioni che portano alla determinazione della soglia possono essere riassunte come segue. In primo luogo, va osservato che la soglia non puó essere inferiore al numero di regole dello schema: puó infatti verificarsi che, durante l'espansione di una query, ciascuna regola venga applicata in corrispondenza di una diversa iterazione. In altri termini accade che, in uno schema con $k$ regole, l'applicazione della $i-esima$ regola, alla $n-esima$ iterazione, comporti l'applicazione della $j-esima$ regola alla $i+1-esima$ iterazione, e così via per tutte le $k$ regole dello schema. Se la soglia, in questo caso, fosse minore di $k$, l'espansione semantica terminerebbe prematuramente.

Una seconda considerazione fondamentale riguarda l'interrogazione da ottimizzare. Puó infatti accadere che una determinata *path query* richieda, per essere correttamente ottimizzata, un numero di iterazioni del metodo superiore al numero di regole dello schema. Per rendersene conto é sufficiente considerare uno schema senza regole di integritá: ebbene l'espansione

semantica di qualunque interrogazione, anche in assenza di regole, richiede una numero di passi maggiore di zero. Ricordiamo infatti che una query puó essere riclassificata anche in base alla tassonomia della classi dello schema.

Partendo dalle considerazione svolte possiamo fissare la soglia sulla base del massimo tra due valori: il numero di regole dello schema e la lunghezza del massimo cammino presente nella query ove, per massimo cammino, si intende il numero massimo di attributi, in dot notation, che compaiono nell'interrogazione considerata.

Una soglia calcolata in questo modo consente, come abbiamo visto, di giungere comunque alla corretta espansione di una query non ricorsiva. In ordine alla trattazione dei cicli, invece, é facile rendersi conto che, da un lato, quando la soglia coincide col numero di regole presenti nello schema, se si eccede tale valore, significa che almeno una delle regole continuerebbe ad essere applicata ciclicamente. D'altro canto, se la soglia coincide con il massimo path, allorché il numero di iterazioni oltrepassa la soglia, significa che uno stesso tipo continuerebbe ad essere espanso ciclicamente.

Per quanto riguarda la terminazione del metodo, va osservato che la condizione di terminazione dell'algoritmo 2.2.3 permane e continua a valere nei casi non ricorsivi. Quando invece, in presenza di cicli, il metodo tenderebbe a superare la soglia, interviene una terminazione forzata che interrompe l'espansione.

L'algoritmo di espansione con soglia é presentato in tabella 5.1.

Una caratteristica determinante dell'algoritmo 5.1 sta nel fatto che la soglia viene calcolata una volta per tutte in fase di inizializzazione del calcolo. Benché questo approccio, come detto, assicuri la terminazione del metodo, allo stesso tempo puó ridurne drasticamente l'efficienza.

Consideriamo, come esempio, uno schema con classi cicliche e un certo numero di regole di integritá. Supponiamo che la soglia coincida con tale numero. Se si intende ottimizzare una query ricorsiva che determini un loop infinito sin dalle primissime iterazioni, la terminazione forzata occorre solo dopo un numero di iterazioni pari al valore della soglia. Se il numero delle regole é elevato, quindi, verranno eseguite inutilmente tutte la iterazioni successive al verificarsi delle condizioni di ciclo fino al raggiungimento della soglia.

Si consideri lo schema di tabella 5.2

Supponiamo ora di dover ottimizzare la seguente interrogazione:

$select\ *$
$from\ test$
$where\ a\ >=\ 6$

Il massimo cammino dell'interrogazione considerata vale uno. Con

**Algoritmo 3** *Espansione semantica con soglia di un tipo S*

- **Inizializzazione**:
  *Acquisizione del tipo S*
  $threshold = max(max_path, rule_number)$
  $j = 0$

- **Iterazione**:
  *if* $j < threshold$
  $\forall\ S_1 \in \mathbf{CT}^+(S)$

$$
\begin{array}{rcl}
Forma\ canonica & : & \nu(S_1) \\
Incoerenza & : & S_1 \in \widetilde{\Phi}_\nu \\
Sussunzione & : & GS(S_1)
\end{array}
$$

$$
\begin{array}{rcl}
se\ S_1 \notin \widetilde{\Phi}_\nu : & & \\
,\,^i(S_1) & = & \left\{ \begin{array}{lll} S_1 \sqcap \bigsqcap_k S_k^c & \forall R_k : & S_k^a \in GS(S_1), \\ & & S_k^c \notin GS(S_1) \\ S_1 & & altrimenti \end{array} \right. \\
j & = & j+1
\end{array}
$$

  *else Stop*

- **Stop**:
  $\forall\ S_1 \in \mathbf{CT}^+(S),\ ,\,^{i+1}(S_1) = ,\,^i(S_1),$
  *or*
  $\exists\ S_1 \in \mathbf{CT}^+(S),,\,^i(S_1) = \bot.$

  *L'espansione semantica è* $\tilde{,}\,(S) = ,\,^i(S)$

Tabella 5.1: Algoritmo di espansione con soglia

```
prim test = ^ [ a : range 1 +inf , b : range 1 +inf ,
               c : classe , d : range 1 +inf ] ;
antev rule1a = test & ^ [ a : range 5 +inf ] ;
consv rule1c = test & ^ [ c : ^ [ a : range 10 +inf ] ] ;
antev rule2a = test & ^ [ b : range 6 +inf ] ;
consv rule2c = test & ^ [ c : ^ [ d : range 20 +inf ] ] ;
antev rule3a = test & ^ [ d : range 15 +inf ] ;
consv rule3c = test & ^ [ c : ^ [ d : range 6 +inf ] ] ;
antev rule4a = test & ^ [ d : range 15 +inf ] ;
consv rule4c = test & ^ [ c : ^ [ b : range 6 +inf ] ] ;
antev rule5a = test & ^ [ b : range 15 +inf ] ;
consv rule5c = test & ^ [ c : ^ [ d : range 2 +inf ] ] .
```

Tabella 5.2: Schema con regole di prova

riferimento al formalismo di tabella 5.1 avremo:

$$threshold \;=\; max(1,5) \;=\; 5$$

vale a dire che l'espansione semantica per la query considerata deve prevedere al piú cinque iterazioni. Tuttavia osserviamo dallo schema che, sin dal secondo passo di espansione si verifica una condizione di ciclo.

Come é stato detto, l'algoritmo in questione continua a iterare il procedimento fino a raggiungere la soglia: ne caso in esame cinque volte. Il risultato che si ottiene é il seguente:

```
5 rule(s) detected
query longest path = 1 (d)
Max 5 iteration(s)

select *
from classe as S
where S.a > 5 and
      S.c.a > 9 and
      S.c.c.a > 9 and
      S.c.c.c.a > 9 and
      S.c.c.c.c.a > 9 and
      S.c.c.c.c.c.a > 9
```

In definitiva questo metodo non é in grado di adattarsi alle condizioni nelle quali si calcola l'espansione semantica; questa staticitá é appunto la

causa del degrado delle prestazioni nei casi sopra descritti. Si impone dunque la necessitá di realizzare una nuova versione dell'algoritmo di espansione semantica che rechi le medesime garanzie di terminazione nei casi ricorsivi, ma che sia contemporaneamente in grado di adattarsi alla query da espandere al fine di presentare, in tutti i casi, la stessa efficienza computazionale. L'algoritmo in questione viene proposto nella sezione successiva.

## 5.3 Lavori in corso

È attualmente in corso di sviluppo un nuovo algoritmo di espansione semantica più completo in grado di memorizzare le informazioni possibili sulle iterazioni giá effettuate. Questo approccio consente di mantenere invariato, rispetto alla versione precedente, il controllo sulla terminazione poiché, utilizzando le informazioni relative alle iterazioni precedenti, é possibile prevenire la trattazione dei cicli. La formalizzazione dell'argoritmo ed i risultati sperimentali sono ancora parziali e pertanto non viene riportato nella presente tesi.

Un'altra parte attualmente oggetto di studio riguarda la fattorizzazione di una classe e l'eliminazione dei fattori. L'eliminazione dei fattori permette di rimuovere da una query i fattori che sono logicamente implicati dallo schema o dalle regole e che, perció, non influenzano il risultato dell'ottimizzazione semantica. Un'eliminazione completa dei fattori ridondanti genera una *forma minima* dell'interrogazione che é semanticamente equivalente a quella originaria.

# Parte II

# Integrazione di Informazioni

# Capitolo 6

# L'Integrazione Intelligente di Informazioni

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

ed affidandosi a complicati algoritmi di "mantenimento" di questi dati, per assicurare la loro consistenza a fronte di cambiamenti nelle sorgenti originali. Con *Integration of Information* invece, come è descritto in [65], si rappresentano in letteratura tutti quei sistemi in grado di combinare tra di loro dati provenienti da intere sorgenti o parti selezionate di esse, senza fare uso della replicazione fisica delle informazioni, bensì basandosi sulle loro descrizioni. Quando inoltre questa integrazione utilizza tecniche di intelligenza artificiale, sfruttando le conoscenze acquisite, possiamo parlare di Intelligent Integration of Information ($I^3$), che si distingue quindi dalle altre forme di integrazione prefiggendosi non una semplice aggregazione di informazioni, bensì un aumento del loro valore, ottenendo nuove informazioni dai dati ricevuti.
Con questi obiettivi si è quindi inserita, nell'ambito dell'integrazione, l'Intelligenza Artificiale (IA), che già aveva dato buoni risultati in domini applicativi più limitati. Naturalmente, è ovvio come sia pressocché impossibile pensare ad un sistema che sia adatto a tutti i domini applicativi, e che magari integri un numero altissimo di sorgenti. Per questo motivo, per realizzare sistemi molto generali e di grande eterogeneità, è stata proposta una partizione delle risorse e dei servizi che questi sistemi devono supportare, e che si articola su due dimensioni:

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

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

I domini nei vari livelli si scambieranno dati e informazioni tra di loro, ma non saranno strettamente collegati. Per esempio, in un sistema di recapito merci navale, le informazioni sulle navi saranno integrate da un modulo intermedio, quelle sul tempo nelle varie regioni da un altro modulo intermedio, ed un ulteriore modulo, ad un livello superiore, provvederà all'integrazione dei dati che gli verranno forniti dai *mediatori* (o *facilitatori*) sottostanti.
In questo quadro, dal 1992, si inserisce il progetto di ricerca $I^3$ fondato e sponsorizzato dall'ARPA, agenzia che fa capo al Dipartimento di Difesa americano [1]. $I^3$ si focalizza sul livello intermedio della partizione sopra descritta, livello che media tra gli utilizzatori e le sorgenti. All'interno di questo livello staranno diversi moduli (per una descrizione più dettagliata si rimanda al paragrafo successivo di questo capitolo), tra i quali i più importanti sono:

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

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

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

L'architettura di riferimento presentata in questo paragrafo è stata tratta dal sito web [1], e rappresenta una sommaria categorizzazione dei principi e dei servizi che possono e devono essere usati nella realizzazione di un integratore *intelligente* di informazioni derivanti da fonti eterogenee. Alla base del progetto $I^3$stanno infatti due ipotesi:

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

In questo ambito, l'obiettivo del programma $I^3$ è di ridurre considerevolmente il tempo necessario per la realizzazione di un integratore di informazioni, raccogliendo e "strutturando" le soluzioni fino ad ora prevalenti nel campo della ricerca. Da sottolineare, prima di passare alla descrizione dell'architettura di riferimento, che questa non implica alcuna soluzione implementativa, bensì vuole rappresentare alcuni dei servizi che deve includere un qualunque

integratore di informazioni, e le interconnessioni tra questi servizi. Inoltre, è opportuno rimarcare che non sarà necessario, ed anzi è improbabile, che ciascun sistema che si prefigge di integrare informazioni (o servizi, o applicazioni) comprenda l'intero insieme di funzionalità previste, bensì usufruirà esclusivamente delle funzionalità necessarie ad un determinato compito.

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

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

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

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

- *la grande differenza tra le fonti di informazione*:
  - le fonti informative sono semanticamente differenti, e si possono individuare dei livelli di differenze semantiche [95];
  - le informazioni possono essere memorizzate utilizzando differenti formati, come possono essere file, DB relazionali, DB ad oggetti;

  - possono essere diversi gli schemi, i vocabolari usati, le ontologie su cui questi si basano, anche quando le fonti condividono significative relazioni semantiche;
  - può variare inoltre la natura stessa delle informazioni, includendo testi, immagini, audio, media digitali;
  - infine, può variare il modo in cui si accede a queste sorgenti: interfacce utente, linguaggi di interrogazione, protocolli e meccanismi di transazione;

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

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

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

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

Passiamo ora ad analizzare l'architettura vera e propria di un sistema $I^3$, riportata in Figura 6.1. L'architettura di riferimento dà grande rilevanza ai Servizi di Coordinamento. Questi servizi giocano infatti due ruoli: come prima cosa, possono localizzare altri servizi $I^3$ e fonti di informazioni che possono essere utilizzati per costruire il sistema stesso; secondariamente, sono responsabili di individuare ed invocare a run-time gli altri servizi necessari a dare risposta ad una specifica richiesta di dati.

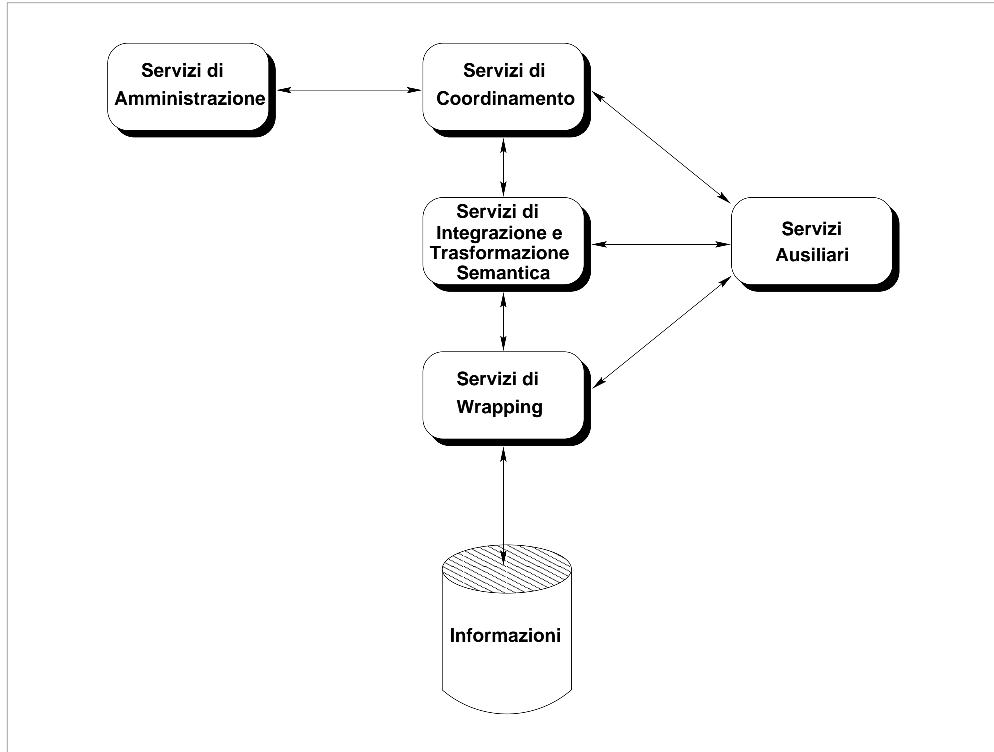



Figura 6.1: Diagramma dei servizi $I^3$

Sono comunque in totale cinque le famiglie di servizi che possono essere identificati in questa architettura: importanti sono i due assi della figura, orizzontale e verticale, che sottolineano i differenti compiti dei servizi $I^3$.
Se percorriamo l'asse verticale, si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i servizi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti, che sono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passati ai servizi di Coordinamento che ne avevano fatto richiesta. L'asse orizzontale mette invece in risalto il rapporto tra i servizi di Coordinamento e quelli di Amministrazione, ai quali spetta infatti il compito di mantenere informazioni sulle capacità delle varie sorgenti (che tipo di dati possono fornire ed in quale modo devono essere interrogate). Funzionalità di supporto, che verranno descritte successivamente, sono invece fornite dai Servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti.
Analizziamone in dettaglio funzionalità e problematiche affrontate.

### 6.1.2 Servizi di Coordinamento

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

1. **Facilitation e Brokering Services**: l'utente manda una richiesta al sistema e questo usa un deposito di metadati per ritrovare il modulo che può trattare la richiesta direttamente. I moduli interessati da questa richiesta potranno essere uno solo alla volta (nel qual caso si parla di Brokering) o più di uno (e in questo secondo caso si tratta di facilitatori e mediatori, attraverso i quali a partire da una richiesta ne viene generata più di una da inviare singolarmente a differenti moduli che gestiscono sorgenti distinte, e reintegrando poi le risposte in modo da presentarle all'utente come se fossero state ricavate da un'unica fonte). In questo ultimo caso, in cui una query può essere decomposta in un insieme di sottoquery, si farà uso di servizi di Query Decomposition e di tecniche di Inferenza (mutuate dall'Intelligenza Artificiale) per una determinazione dinamica delle sorgenti da interrogare, a seconda delle condizioni poste nell'interrogazione.
   I vantaggi che questi servizi di Coordinamento portano stanno nel fatto che non è richiesta all'utente del sistema una conoscenza del contenuto delle diverse sorgenti, dandogli l'illusione di interagire con un sistema omogeneo che gestisce direttamente la sua richiesta. E' quindi esonerato dal conoscere i domini con i quali i vari moduli $I^3$ hanno a che fare, ottenendone una considerabile diminuzione di complessità di interazione col sistema.

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

### 6.1.3 Servizi di Amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti *utili*, per determinare le loro capacità, e per creare ed interpretare TEMPLATE. I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da

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

### 6.1.4 Servizi di Integrazione e Trasformazione Semantica

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

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

individuazione dei concetti presenti in diverse fonti, e la riconciliazione delle diversità presenti sia nelle strutture, sia nei significati dei dati.

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

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

### 6.1.5 Servizi di Wrapping

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

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

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

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

### 6.1.6 Servizi Ausiliari

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

# Capitolo 7

# Stato dell'arte nell'ambito dell'integrazione di informazioni

Il presente capitolo si pone l'obiettivo di fornire una panoramica dello stato dell'arte della ricerca relativamente all'integrazione intelligente di informazioni da fonti eterogenee. Sono stati esaminati tre progetti, TSIMMIS, GARLIC e SIMS, tra i più importanti presenti nell'ambito della ricerca. La scelta di questi sistemi è motivata anche dal loro diverso approccio all'integrazione, così da mostrare la pluralità delle tendenze di ricerca.

Nella sezione 7.1 viene presentato il progetto di ricerca TSIMMIS, sviluppato presso l'Università di Stanford. Le sottosezioni 7.1.1 e7.1.2 descrivono, rispettivamente, il modello dei dati OEM e il linguaggio MSL utilizzati nel progetto. Le successe sottosezioni descrivono l'architettura e i moduli del sistema di integrazione proposto; infine la sottosezione 7.1.6 descrive brevemente pregi e difetti dell'approccio.
La sezione 7.2 presenta il progetto GARLIC, sviluppato presso il centro di ricerca IBM di Almaden, descrivendo il linguaggio GDL (in 7.2.1), il query planning (in 7.2.2) e pregi e difetti dell'approccio (in 7.2.3).
La sezione 7.3 presenta il progetto SIMS, sviluppato presso l'Università della Southern California, descrivendo il processo di integrazione delle sorgenti (in 7.3.1), il query processing (in 7.3.2) e pregi e difetti dell'approccio (in 7.3.3).

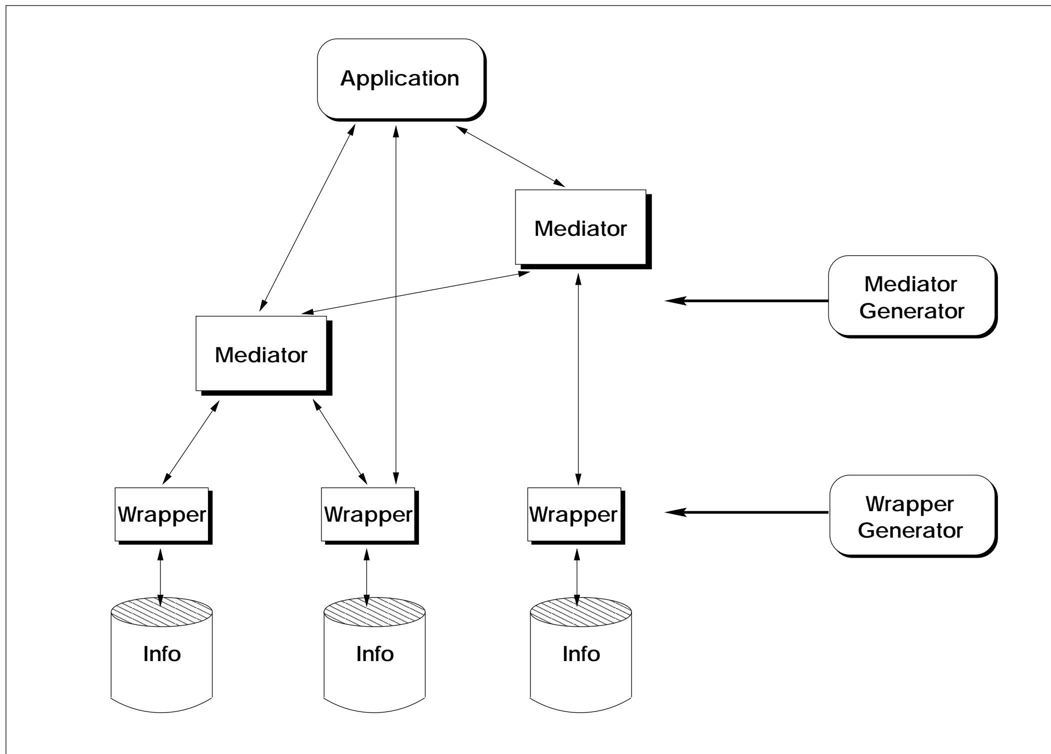



Figura 7.1: Architettura TSIMMIS

## 7.1 TSIMMIS

**TSIMMIS** (**T**he **S**tanford- **I**BM **M**anager of **M**ultiple **I**nformation **S**ources) [70, 91, 106] è sicuramente uno dei progetti più interessanti in questo campo: sviluppato presso l'Università di Stanford in collaborazione con il Centro di ricerca IBM di Almaden, si pone come obiettivo lo sviluppo di strumenti che facilitino la rapida integrazione di sorgenti testuali eterogenee, includendo sia sorgenti di dati strutturati che **semistrutturati**. Questo obiettivo è raggiunto attraverso un'architettura comune a molti altri sistemi: i *wrapper* convertono i dati in un modello comune mentre i *mediator* combinano ed integrano i dati ricevuti dai wrapper. I wrapper inoltre forniscono un linguaggio di interrogazione comune per l'estrazione delle informazioni, mostrando un'interfaccia verso l'esterno uguale a quella dei mediatori: in questo modo, l'utente può porre le interrogazioni attraverso un unico linguaggio sia ai mediatori (ricevendo dati integrati da più sorgenti), sia direttamente ai wrapper (interrogando in questo modo un'unica fonte).

In Figura 7.1 è mostrata l'architettura: ad ogni sorgente corrisponde un wrapper (o *traduttore*) che converte nel modello comune i dati estratti dalla sorgente; sopra i wrapper stanno i mediatori. I wrapper convertono inoltre le query scritte utilizzando il modello comune in richieste comprensibili dalla particolare sorgente da loro servita.

La peculiarità di questo progetto si manifesta nel modello comune dei dati utilizzato per rappresentare le informazioni. **OEM** (Object Exchange Model) [107] è un modello a *etichette* basato sui concetti di identità di oggetto e annidamento particolarmente adatto a descrivere dati la cui struttura non è nota o è variabile nel tempo. In aggiunta a questo, sono disponibili ed utilizzati dal sistema due linguaggi di interrogazione (*OEM-QL*) e *MSL* (Mediator Specification Language), per ottenere i dati dalle fonti ed integrarli opportunamente.

Per facilitare il compito dell'amministratore del sistema, sono stati progettati due moduli (*translator generator* e *mediator generator*) che supportano le fasi di realizzazione rispettivamente dei wrapper e dei mediatori, fornendo un insieme di librerie di funzioni predefinite.

### 7.1.1 Il modello OEM

Si presenta brevemente il modello OEM, fondamentale per capire il tipo di approccio (*strutturale*) dell'intero progetto TSIMMIS. OEM fa parte dei cosiddetti *self describing model*, dove ad ogni informazione è associata una etichetta che ne descrive il significato. Esso comprende caratteristiche tipiche dell'approccio ad oggetti ma in modo molto semplificato: non fa uso di un forte sistema dei tipi (in pratica sono ammessi solamente i tipi base), non supporta direttamente né le classi, né i metodi, né l'ereditarietà, bensì solo l'identità e il nesting tra oggetti.

Un esempio di descrizione dell'oggetto **persona** è il seguente:

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

L'oggetto `person` considerato viene rappresentato da un'insieme di oggetti annidati: ogni oggetto è rappresentato da stringhe separate da virgole e comprese fra i simboli $<$ e $>$. La prima stringa (ad es. `ob1`) rappresenta l'identificatore, la successiva è un'etichetta che ne qualifica il genere, la terza

e la quarta specificano il tipo ed il valore. Fondamentale, per la semantica dell'oggetto stesso, è l'etichetta, che ne descrive il ruolo all'interno del contesto in cui è utilizzato. Nell'esempio riportato, sono descritti in totale sei oggetti: un oggetto che si potrebbe definire di top-level (`person`) e cinque sotto-oggetti, che a `person` sono collegati, come mostra l'ultima stringa dell'oggetto `ob1`. È importante sottolineare che, usando l'OEM, non è necessario che oggetti che si descrivono tramite la stessa etichetta abbiano pure lo stesso schema: per esempio, potrebbe esistere un altro oggetto `person` con un diverso insieme di tipi di sotto-oggetti. In questo modo risulta molto semplificata l'integrazione di oggetti provenienti da schemi diversi e semi-strutturati, non dovendo predefinire le strutture che questi oggetti dovranno seguire, ed anzi accettando tutti gli oggetti con una determinata etichetta, qualunque schema seguano.

### 7.1.2 Il linguaggio MSL

MSL è il linguaggio utilizzato per definire, in modo dichiarativo, i mediatori: attraverso l'uso di *rule*, si può specificare il punto di vista del mediatore, ed in questo modo definire lo "schema globale" in modo manuale. A run time, ricevuta una particolare richiesta, l'interprete del mediatore (MSI) raccoglie ed integra le informazioni ricevute dalle sorgenti, in accordo con le specifiche definite nella vista. MSL è dunque un linguaggio dichiarativo per la definizione di *viste* in grado di interfacciarsi con OEM e quindi di supportare evoluzioni degli schemi, irregolarità strutturali degli oggetti delle sorgenti, strutture sconosciute senza dover ogni volta ridefinire le viste. Ogni regola è costituita da una *testa* e da una *coda*, separate dal simbolo :-. La *coda* descrive dove andare a recuperare l'oggetto che si vuole ricevere, mentre l'intestazione definisce la struttura che questo oggetto dovrà avere, una volta estratto e ricostruito. Un esempio di regola MSL sarà presentato nella Sezione 7.1.4

### 7.1.3 Il generatore di Wrapper

TSIMMIS include un insieme di strumenti, chiamati *OEM Support Libraries*, per realizzare facilmente i wrapper, i mediatori e le interfacce utenti. Questi strumenti comprendono diverse procedure per lo scambio di oggetti OEM in un architettura Client/Server, dove un Client può essere indifferentemente un mediatore, una applicazione o un'interfaccia utente, mentre il server può essere sia un wrapper, sia un mediatore. In questo modo, si è cercato di semplificare il più possibile la realizzazione dei wrapper, ed in particolare del modulo di Conversione, che ha il compito di convertire una query espressa

nel linguaggio MSL in una interrogazione comprensibile dalla sorgente con la quale il wrapper interagisce.

Per facilitare l'intero processo di integrazione, e per poter servire anche sorgenti dalle limitate possibilità di risposta ad interrogazioni, nel progetto TSIMMIS si è ipotizzato di dover esprimere esplicitamente l'intero insieme delle query a cui la sorgente può rispondere (dunque un insieme comunque limitato di query) e di predisporre, attraverso i cosiddetti *query templates*, per ogni query supportabile in MSL, la rispettiva traduzione nel linguaggio di interrogazione proprio della sorgente stessa. Un esempio potrà sicuramente semplificare la descrizione di tutto il processo. Supponiamo di interagire con una base di dati universitaria, *WHOIS*, che mantiene informazioni sugli studenti e sui professori di una data università, e con possibilità molto limitate: le uniche operazioni consentite sono il ritrovamento dei dati di una persona, conoscendone il nome o il cognome (od entrambi). Le interrogazioni supportate dalla sorgente, espresse nel suo linguaggio, potrebbero essere le seguenti:

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

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

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

Ad ogni *query template*, è associata una azione, scritta in questo caso in C, che realizza la traduzione da MSL a linguaggio "nativo". Naturalmente, il sistema sarebbe molto limitato se permettesse di rispondere solamente a questo insieme predefinito di interrogazioni: esiste dunque un modulo che permette di definire, data una query in input (che può essere ricevuta da un mediatore, come pure direttamente da una applicazione o da un'interfaccia

utente), se ad essa la sorgente sia in grado di dare risposta. In particolare, oltre alle query predefinite, sono supportate tutte le query q tali che:

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

### 7.1.4 Il generatore di Mediatori

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

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

mentre la seconda è una sorgente ad oggetti, *WHOIS*. Ad ogni sorgente corrisponde, come già mostrato nell'architettura, un wrapper: **CS** esporta le informazioni della prima, **WHOIS** quelle della seconda (esempi di oggetti esportati da questi wrapper sono rispettivamente riportati in Figura 7.2 e in Figura 7.3).

Si vuole sviluppare un modulo mediatore, chiamato **MED**, che integri tutte le informazioni inerenti una persona, ricavate dai due wrapper. Per esempio, supponendo che la persona in questione si chiami 'Chung', ed appartenga al dipartimento 'CS' (ovvero Computer Science), la risposta che si vorrebbe ottenere è rappresentata in Figura 7.4.

Per realizzare questa integrazione, TSIMMIS fa uso di un sistema di regole, MSL (Mediator Specification Language), che permettono di specificare la struttura dell'oggetto integrato, a partire dalle strutture degli oggetti da recuperare nelle sorgenti. Le regole devono essere quindi esplicitamente specificate dall'amministratore del sistema, che è l'unico che deve conoscere lo schema di *persona* delle sorgenti e del mediatore. In particolare, riferendoci all'esempio, la rule che realizza il processo specificato sarà:

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

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

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

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

Figura 7.2: Oggetti esportati da **CS** in OEM

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

Figura 7.3: Oggetti esportati da **WHOIS** in OEM

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

La *testa* della regola `MS1` specifica la struttura che avrà l'oggetto unificato: sarà esportato da **MED** con etichetta `cs_person` con un nome (`name`), un ruolo (`relation`), ed un insieme non specificato di altri attributi, ovvero con tutte le altre informazioni che sarà possibile recuperare dalle due sorgenti (rispettivamente `Rest1` e `Rest2`). Nella *coda* sono invece definiti i percorsi dove devono essere recuperate le informazioni: dalla sorgente **WHOIS** si ricavano oggetti di tipo `person` con un nome definito (lo stesso espresso nell'intestazione della rule), un ruolo, e l'attributo dipartimento uguale a 'cs'; dalla sorgente **CS** sono invece recuperati degli oggetti di tipo `R` (dove `R` è

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

Figura 7.4: Oggetti esportati da **MED**

il ruolo specificato nell'intestazione, e può valere '`employee`' o '`student`), e di nome e cognome specificato. In ausilio alla rule vi è inoltre una funzione esterna, `decomp`, che realizza la trasformazione da `name` a `first_name` e `last_name` e viceversa.
In tutto il processo non permane alcuna ambiguità: per prima cosa, sono recuperati dalle sorgenti locali gli oggetti la cui struttura (e le cui etichette) sono conformi alla struttura specificata nella *coda* della rule, poi questi oggetti sono unificati e presentati in modo integrato come specificato nella *testa*.

### 7.1.5 Il Linguaggio LOREL

Evidenziato il punto debole del tipo di integrazione realizzata nella scarsa flessibilità dovuta alle interrogazioni predefinite dai *query template*, si è cercato di porvi rimedio sviluppando un linguaggio di interrogazione ad hoc: LOREL (Ligtweight Object REpository Language) [63]. Caratteristica saliente di questo linguaggio, la cui sintassi si ispira ad SQL ed OQL, consiste nel fatto che esso è in grado di sfruttare strutture predefinite, se presenti, ma non ne ha necessariamente bisogno per fornire risposte significative ad un'interrogazione. Come OEM è un modello ad "oggetti leggeri", nel senso che presenta un ridotto numero di caratteristiche dei tradizionali modelli ad oggetti, così LOREL è definiti un linguaggio di interrogazione per "oggetti leggeri". In questo paragrafo si illustrano solo le caratteristiche salienti di questo linguaggio, rimandandone l'approfondimento alla letteratura.

LOREL si distingue dagli altri linguaggi di interrogazione perchè:

- è in grado di recuperare informazioni anche quando certi dati non sono presenti, attraverso un assegnamento parziale, e non totale, delle tuple o degli oggetti
- non distingue tra attributi multi_valore e a valore singolo (in SQL at-

tributi multivalore non sono ammessi, in OQL sono distinti da una diversa sintassi)

- opera uniformemente su dati che hanno tipi differenti perchè non esegue controllo dei tipi
- consente di interrogare sorgenti la cui struttura è parzialmente sconosciuta utilizzando wildcards in sostituzione dei nomi degli attributi

Queste caratteristiche sono in varia misura dovute al fatto che, al contrario degli altri linguaggi, LOREL non impone un rigido sistema di tipi, anzi tutti i dati, compresi i valori scalari, sono rappresentati da oggetti (con lo stesso paradigma di OEM, quindi): ogni oggetto ha un identificatore, un'etichetta ed un valore, quest'ultimo può essere tanto uno scalare quanto un set di oggetti innestati nell'oggetto iniziale.

### 7.1.6 Pregi e difetti di TSIMMIS

Il punto di forza di questo progetto è sicuramente il modello comune OEM adottato: esso permette l'integrazione tanto di oggetti la cui struttura è fissa e nota o variabile nel tempo, quanto di oggetti di struttura non predefinita (si veda nell'esempio l'uso degli attributi `Rest1` e `Rest2` in Sezione 7.1.4), rendendo possibile ciò che è negato in qualunque ambiente convenzionale orientato agli oggetti che non aderisca ad una semantica di mondo aperto. Queste possibilità rendono l'intero sistema estremamente flessibile, permettendo l'integrazione di sorgenti il cui schema può essere sia parzialmente sconosciuto, sia variabile nel tempo.

Il meccanismo dei *query template* sostiene il fatto che le sorgenti abbiano diverse capacità di rispondere ad un'interrogazione (è infatti inverosimile pensare che tutte possiedano le funzionalità di un DBMS) e semplifica la fase di interrogazione: a causa della assenza di una struttura predefinita l'interrogazione coi tradizionali statement SQL/OQL diventa aleatoria. A questo processo può però essere mosso un appunto: esprimendo le capacità di risposta di una fonte di informazioni attraverso query predefinite, è improbabile che si riesca a rappresentare completamente il range di informazioni estraibili dalle sorgenti: può accadere il caso in cui la fonte non risponde ad una query q ma è invece in grado di realizzare una query q' più generale di q. Basterebbe allora, per realizzare q, avere un filtro presso il mediatore, ed eseguire una ulteriore selezione sui dati ricevuti in risposta a q'.

Per quanto riguarda la soluzione proposta con l'introduzione di LOREL, a causa della novità della proposta, un indubbio svantaggio risiede nel fatto

che non si possono comunque manipolare le query e gli oggetti facendo affidamento sulle specifiche funzionalità di query processing dei DBMS esistenti, mentre occorre sviluppare moduli dedicati per le fasi di parsing, query rewrite, optimization and execution. Il vantaggio principale consiste invece nel riuscire a superare alcune delle limitazioni di SQL ed OQL. Inoltre, rispetto alla soluzione coi query template, lascia all'utente una maggior flessibilità di interrogazione (ma indubbiamente anche una maggior complessità).

Rimane poi un altro problema aperto: il processo di integrazione di informazioni è di per sè un processo ambiguo, in cui non si possono conoscere le sorgenti se non in modo approssimativo; perciò risulta una forzatura eccessiva ipotizzare di lasciare esclusivamente al progettista del sistema il compito di individuare i concetti comuni e di integrarli opportunamente. Questo grosso svantaggio è imputabile anche all'approccio *strutturale* scelto, che non richiede (e quindi non utilizza) gli schemi concettuali delle sorgenti. A questo si contrappone l'approccio *semantico*, nel quale sono sfruttate le informazioni codificate negli schemi al fine di pervenire ad una totale integrazione di tutte le fonti di informazioni.

## 7.2 GARLIC

L'obiettivo del progetto **GARLIC** [85, 94] (sviluppato presso il centro di ricerca IBM di Almaden) è la costruzione di un Multimedia Information System (MMIS) in grado di integrare dati che risiedono in differenti basi di dati, nonché in un insieme di server di diversa natura, mantenendo la reciproca indipendenza dei server ed evitando la replicazione fisica dei dati. *Multimedia* deve in questo contesto essere interpretato in senso molto ampio, indicando non solo immagini, video, e audio ma anche testi e tipi di dati specifici di alcune applicazioni (disegni CAD, mappe, ... ). Poiché molte di queste informazioni sono già modellate tramite oggetti, GARLIC fornisce un modello orientato ad oggetti che permette a tutte le differenti sorgenti di descriversi in modo uniforme. A questo modello si aggiunge inoltre un linguaggio di interrogazione, anch'esso orientato ad oggetti (ottenuto con un'estensione al linguaggio SQL) e utilizzato dal livello intermedio dell'architettura GARLIC, i cui compiti sono:

- presentare lo schema globale alle applicazioni,
- interpretare le interrogazioni,
- creare piani di esecuzione per le interrogazioni e
- riassemblare i risultati in un'unica risposta omogenea.

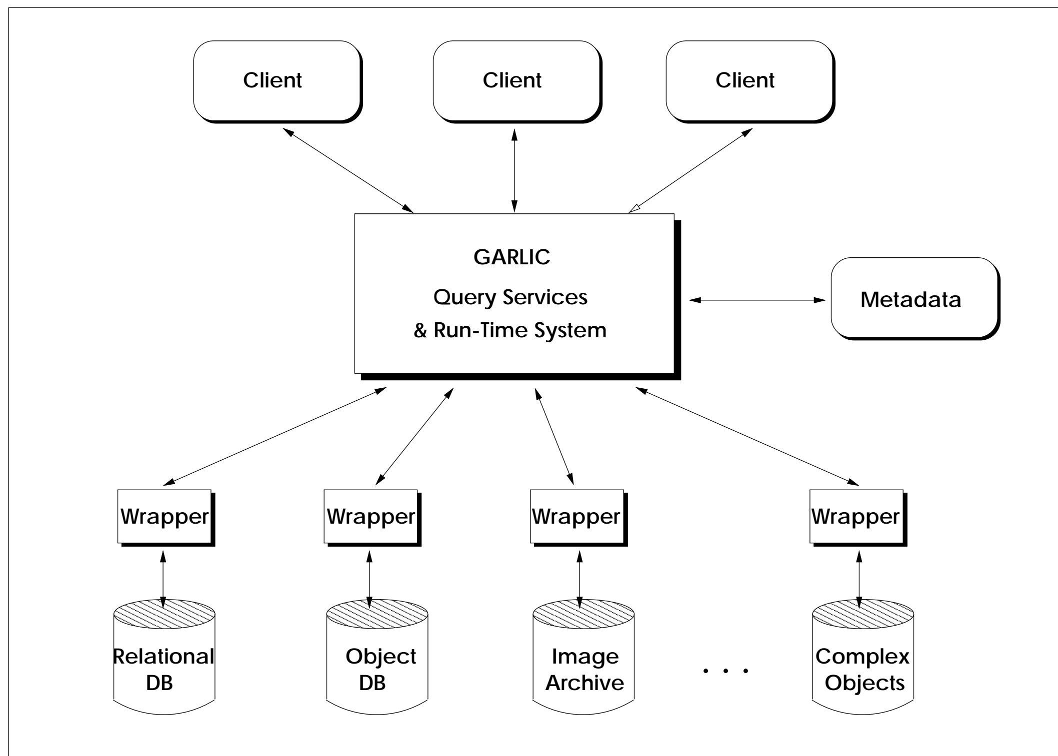



Figura 7.5: Architettura GARLIC

La Figura 7.5 rappresenta l'architettura di GARLIC. Alla base della figura vi sono le sorgenti di informazioni che devono essere integrate (tra le quali basi di dati relazionali, ad oggetti, file system, document manager, image manager, ... ). Sopra ogni sorgente è posizionato un wrapper, che traduce le informazioni sugli schemi, sugli accessi ai dati e sulle interrogazioni, dal protocollo interno di GARLIC ai protocolli nativi delle sorgenti. Le informazioni riguardanti lo schema unificato sono mantenute nel deposito di metadati. L'altra sorgente di informazioni alla base della figura (Complex objects) serve invece per memorizzare gli *oggetti complessi* di GARLIC, utilizzati dalle applicazioni per unire i dati originariamente separati (siano essi appartenenti a schemi distinti, o allo stesso schema). I servizi di query processing sono forniti dal componente *Query Services & Run-Time System*: ad esso spetta il compito di presentare alle applicazioni una visione unificata, ad oggetti, del contenuto del sistema e di gestirne le richieste (interrogazioni o modifiche). Lo schema globale è presentato all'utente, e alle applicazioni, attraverso il modello di dati di GARLIC (**G**arlic **D**ata **M**odel): è costituito

dalla **unione** degli schemi locali. Fra questi è pure disponibile lo schema degli *oggetti complessi*, creati ad hoc dalle applicazioni per avere una visione integrata di oggetti preesistenti. Il fornire una descrizione dei dati attraverso il GDL (**G**arlic **D**ata **L**anguage) permette inoltre di identificare i dati che si vogliono integrare e parallelamente escludere dalla descrizione quelli che non si vogliono rendere accessibili dall'esterno.

In sostanza GARLIC presenta all'utente ed alle loro applicazioni i benefici dei database caratterizzati da un preciso schema (simile a quello offerto dai DBMS ad oggetti o object_relational) ma senza replicazione fisica. Internamente le differenze da un DBMS tradizionale sono invece più evidenti: il sistema riunisce informazioni provenienti da più sorgenti e per fonderle utilizza due strumenti:

- il modello dei dati orientato agli oggetti
- la memorizzazione degli oggetti complessi[1].

### 7.2.1 Il linguaggio GDL

Tra i vari compiti dei wrapper vi è quello di fornire una descrizione del contenuto della sorgente da loro servita, utilizzando il **G**arlic **D**ata **L**anguage, o **GDL**. GDL è un variante dell'ODMG[2] **O**bject **D**escription **L**anguage [57]: attraverso le interface, ed un forte sistema dei tipi, si possono descrivere gli oggetti ed il loro comportamento, e memorizzare la loro descrizione in un *repository schema*. I vari *repository* sono quindi registrati come parti di un GARLIC Database, e *fusi* nello schema globale presentato all'utente. Un esempio di GDL è riportato in Figura 7.6.

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

[1] Gli Oggetti Complessi servono nell'integrazione dei dati multimediali con quelli tradizionali, per aggiungere metodi che implementino nuovi comportamenti realizzabili grazie alla visione completa delle informazioni.

[2] Le differenze dallo standard ODL sono dovute alla necessità di esprimere situazioni non presenti in ambiente centralizzato.

Relational Repository Schema

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

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

Web Repository Schema

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

Image Server Repository Schema

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

Figura 7.6: GDL schema

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

me oggetti veri e propri.
Interessante è anche l'uso della parola chiave `readonly`, che permette di discriminare tra fonti aggiornabili e fonti da cui si può esclusivamente estrarre dati, anche se poi non viene spiegato alcun meccanismo di propagazione degli update.

### 7.2.2 Query Planning

La fase di query planning porta, da una interrogazione posta sullo schema unificato, alla definizione di un insieme di query che le sorgenti locali devono eseguire. Parte attiva in questo processo la hanno i wrapper: le loro conoscenze sono utilizzate per formulare differenti piani di accesso, e per determinare il più efficiente tra questi. Durante la fase di pianificazione della query l'ottimizzatore di GARLIC identifica il frammento maggiore possibile che coinvolge una particolare sorgente, e lo spedisce al corrispondente wrapper: questo determina zero o più piani di accesso che realizzino, in toto o in parte, la query a lui assegnata. A questo punto l'ottimizzatore memorizza tutti i piani di tutti i wrapper interrogati, ed eventualmente aggiunge le operazioni da effettuare nel caso in cui una parte della query originale non sia eseguibile da alcun wrapper. È infatti da sottolineare come GARLIC sia in grado di gestire pure sorgenti con particolari restrizioni: oltre ad ipotizzare che una sorgente non sia in grado di effettuare, ad esempio, join a più vie, il wrapper può essere a conoscenza di particolarissime limitazioni sulle operazioni realizzabili, come possono essere la lunghezza massima di una stringa, il valore massimo di una costante in una interrogazione,ecc ... Il vantaggio consiste nel fatto di non dover comunicare tutte le restrizioni all'ottimizzatore, bensì di incapsularle a livello di wrapper. A sua volta l'ottimizzatore, in grado di realizzare funzioni tipiche di un DBMS, potrà effettuare quelle operazioni scartate dalle sorgenti. Oltre a questo, la possibilità di limitare le funzioni di risposta di una sorgente agevola notevolmente la fase di sviluppo di un wrapper: in un primo momento si possono realizzare solo le funzioni più semplici (rendendo utilizzabili da subito le sue informazioni), rimandando ad un secondo momento lo sviluppo di funzioni di ricerca più avanzate.

### 7.2.3 Pregi e difetti di GARLIC

Nonostante GARLIC sia da considerare la versione commerciale di TSIMMIS (sono entrambi stati sviluppati in ambienti IBM), l'intera impostazione del progetto è stata modificata. La differenza maggiore è senza dubbio l'abbandono del modello OEM (vedi Sezione 7.1.1), a cui è stato preferito l'utilizzo degli schemi locali (descritti attraverso il modello GDL): a fronte di una per-

dita di flessibilità dell'intero sistema (è ora praticamente impossibile gestire anche sorgenti semi-strutturate), l'intera architettura risulta semplificata, così come pure è semplificata la fase di integrazione degli schemi. Purtroppo però non sono stati fatti passi avanti nella automazione della fase di integrazione: l'utente, o l'applicazione, deve definirsi una visione ad-hoc di tutti gli schemi, o deve considerarne semplicemente l'unione (con tutte le duplicazioni di informazioni che ne conseguono). Molto approfondita è invece la fase di query planning, in senso di tradizionali operazioni di ottimizzazione dei costi di accesso ai DB, che non si è potuto descrivere meglio in questi paragrafi per motivi di spazio, ma a cui si rimanda negli articoli in bibliografia [85, 94].
Sostanzialmente diverse da progetti analoghi sono comunque le funzioni del wrapper: da semplice traduttore di linguaggi e protocolli, in GARLIC il wrapper include molte delle funzioni demandate in altri sistemi al mediatore vero e proprio. Se dal punto di vista architetturale questo potrebbe anche essere considerato un errore, risulta sorprendente (e forse poco credibile) la sostanziale differenza dei tempi di sviluppo dichiarata da TSIMMIS e da GARLIC: mentre in TSIMMIS si considera ragionevole impiegare circa 6 mesi per realizzare un semplice wrapper, nel progetto GARLIC può essere sviluppato in poche settimane ...

## 7.3 SIMS

**SIMS** [7, 8], sviluppato presso l'Università della Southern California, è un mediatore di informazioni che si occupa di fornire accesso e integrazione ad una molteplicità di sorgenti eterogenee. Il cuore del progetto, ed il suo punto di forza, è la sua dichiarata abilità nel ritrovare e trattare le informazioni in modo *intelligente*, ovvero utilizzando tecniche di intelligenza artificiale. SIMS si distingue infatti dai progetti precedentemente esposti, e ne migliora l'approccio all'interrogazione, per la sua abilità nell'ottimizzare la fase di query processing, essendo in grado di manipolare, attraverso tecniche di intelligenza artificiale, le descrizioni semantiche delle sorgenti. Rimane invece manuale la fase di integrazione delle informazioni (ovvero la costruzione dello schema globale a partire da quelli locali), nonostante sia stata automatizzata, attraverso il modulo LIM, la traduzione di tutti gli schemi locali dal modello originale al modello di conoscenza di SIMS, che si basa sulla logica descrittiva LOOM.

Il progetto si basa sulle seguenti idee di fondo:

- *Rappresentazione e Modellazione della conoscenza*: è usata per descrivere il dominio che accomuna le informazioni, come pure le strutture

ed il contenuto delle sorgenti di informazioni stesse. Il modello di ogni sorgente deve indicare il modello dei dati da questa utilizzato, il linguaggio di interrogazione, la dimensione, e deve descrivere il contenuto di tutti i suoi campi usando la terminologia di un predefinito modello comune del dominio (denominato *domain model*, e che costituisce in pratica lo schema globale);

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

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

### Componenti di SIMS

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

1. **LOOM**: è il sistema di rappresentazione della conoscenza utilizzato per descrivere sia le fonti locali di informazioni, sia lo schema globale, sia la fonte di informazione rappresentata dalle risposte alle query già ottenute, e memorizzate, da una determinata applicazione. Si tratta di una logica descrittiva derivato dal KL-ONE (modello sviluppato da Brachman nel 1976 in [37]).

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

3. **Prodigy**: È il modulo che risolve i problemi di selezione delle sorgenti e pianificazione delle interrogazioni: partendo da una query sullo schema globale, utilizzando una serie di operatori da applicare alla query ed alla conoscenza memorizzata, ottiene uno stato finale caratterizzato dalle risposte che soddisfano la query.

### 7.3.1 Integrazione delle sorgenti

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

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

In aggiunta ai modelli delle sorgenti, viene definito un modello del dominio applicativo, definito *domain model*, che costituirà lo schema globale, l'unico col quale l'utente dovrà interagire. Questo è costituito da una base di conoscenza terminologica organizzata in modo gerarchico (attraverso il linguaggio LOOM), dove i nodi rappresentano tutti gli oggetti, le azioni, gli stati, possibili all'interno del dominio.

Le entità del dominio non devono però necessariamente corrispondere a classi appartenenti ad una determinata sorgente: il *domain model* deve essere inteso come la descrizione del dominio applicativo dal punto di vista dell'utente, e solo con esso l'utente avrà a che fare. In particolare, per porre una query, l'utente compone uno statement LOOM (un esempio di interrogazione è riportato in Figura 7.7, che richiede la profondità del porto di SAN-DIEGO) usando solo la terminologia del *domain model*, essendo in questo modo esonerato dal dover conoscere tutte le sorgenti integrate nel sistema (benché possa pure interagire direttamente con alcune di esse, se ha particolare familiarità con i loro termini).

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

Figura 7.7: Esempio di query SIMS

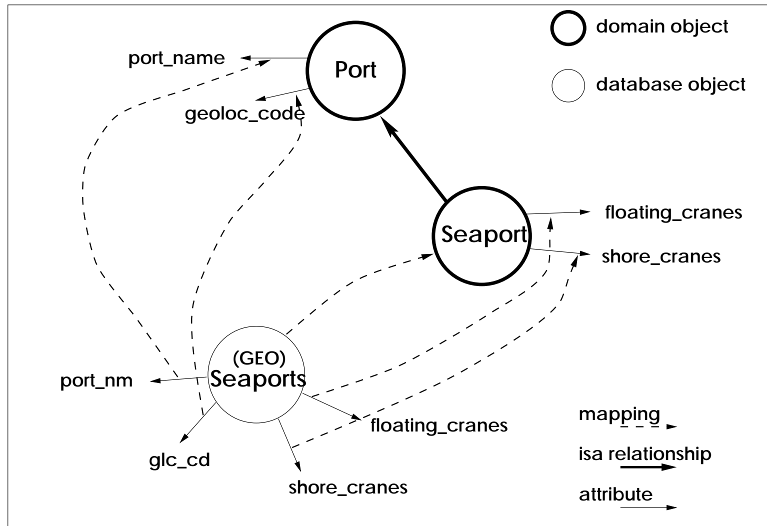



Figura 7.8: Mapping tra *domain model* e modello locale

La parte critica dell'intero processo è invece la fase di integrazione delle sorgenti, ovvero il collegare lo schema globale alle varie sorgenti locali. Questo consiste nel descrivere tutti i concetti e le relazioni di una sorgente attraverso i termini del *domain model*. Questo mapping deve essere fatto per tutte le sorgenti, ed in modo particolarmente accurato: un anello di mapping tra un concetto globale ed uno locale significa che i due concetti rappresentano la stessa classe di informazioni (ed analogamente per gli attributi). Un esempio qualitativo di questo mapping è riportato in Figura 7.8: se l'utente richiede tutti gli elementi della classe globale `Seaport`, andrà interrogata la classe `Seaports` della sorgente `GEO`.

### 7.3.2 Query Processing

L'intero processo che porta dalla formulazione di una query da parte dell'utente, posta sullo schema globale, alla presentazione dei risultati, è rappresentato in Figura 7.9.

**Selezione delle fonti informative** Il primo passaggio da realizzare, una volta in possesso dell'interrogazione dell'utente formulata usando la terminologia dello schema globale, è identificare le appropriate sorgenti che saranno interessate dalla query. Per esempio, se l'utente richiede informazioni sui `porti` ed esiste in una base di dati un concetto che contiene i porti, il mapping è facilissimo, come pure la riformulazione della query. In realtà, molto

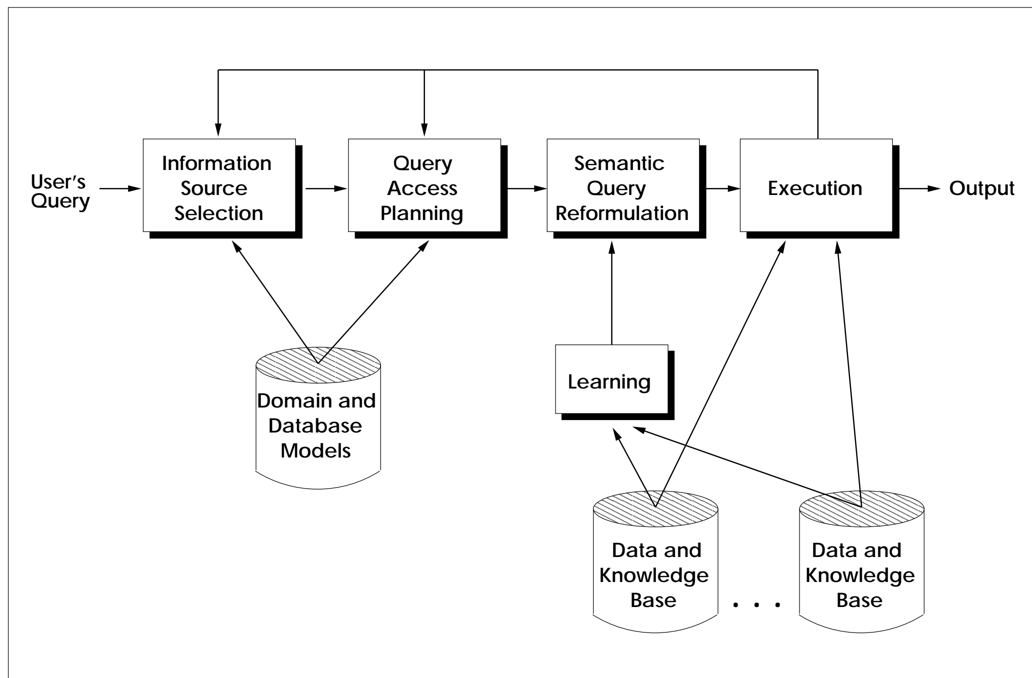



Figura 7.9: Query Processing

spesso non esisterà un mapping diretto e sarà necessario riformulare la query utilizzando termini propri delle sorgenti locali. A questo scopo, vengono utilizzati una serie di operatori di riformulazione, diretti a definire una query equivalente alla originale. Tra di essi, operatori di generalizzazione (che fanno uso delle relazioni tra classe e super-classe e spostano ad un livello superiore dell'albero gerarchico la query, magari aggiungendovi delle limitazioni ai domini degli attributi in modo da ottenere ancora una query equivalente), operatori di specializzazione (che attraverso tecniche di intelligenza artificiale cercano di riclassificare la query ad un livello gerarchico inferiore, in modo da determinare l'insieme di classi da interrogare), operatori di partizione.

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

analoghe a quelle di un ottimizzatore DBMS.

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

### 7.3.3 Pregi e difetti di SIMS

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

Rimane intentata, come negli altri progetti, l'automazione della fase di integrazione degli schemi locali, mentre risulta possibile usufruire della conoscenza semantica degli schemi sorgenti, cosa che negli altri approcci è stata trascurata.

# Capitolo 8

# Il sistema di integrazione intelligente di informazioni MOMIS

Il presente Capitolo illustra un sistema $I^3$, chiamato MOMIS (Mediator envirOnment for Multiple Information Sources), per l'integrazione di sorgenti di dati strutturati e semistrutturati [23, 22, 24]. MOMIS nasce all'interno del progetto MURST 40% *INTERDATA*, come collaborazione tra l'unitá operativa dell'Universitá di Milano e l'unitá operativa dell'Universitá di Modena di cui l'autore é membro. MOMIS si basa su due tool preesistenti, chiamati ARTEMIS (Analysis of Requirements: Tool Environment for Multiple Information Systems) e ODB-Tools e li integra per fornire un'architettura $I^3$ che permetta l'integrazione degli schemi e la query optimization.

Il componente di MOMIS obiettivo della tesi è il **mediatore**, ovvero del modulo intermedio dell'architettura precedentemente descritta, che si pone tra l'utente e le sorgenti di informazioni. Secondo la definizione proposta da Wiederhold in [104] "un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore ... Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti."
Compiti di un mediatore sono allora:

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

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

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

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

3. wrapper: ogni wrapper gestisce una singola sorgente, convertendo le richieste del mediatore in una forma comprensibile dalla sorgente, e le informazioni da essa estratte nel modello usato dal mediatore.

Facendo riferimento ai servizi descritti nelle sezioni precedenti, l'architettura del mediatore che si è progettato è riportata in Figura 8.1. In particolare, in questa tesi, sono state esaminate le seguenti funzionalità:

- servizi di Coordinamento: sul modello di facilitatori e mediatori, il sistema sarà in grado, in presenza di una interrogazione, di individuare automaticamente tutte le sorgenti che ne saranno interessate, ed eventualmente di scomporre la richiesta in un insieme di sottointerrogazioni diverse da inviare alle differenti fonti di informazione;

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

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

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

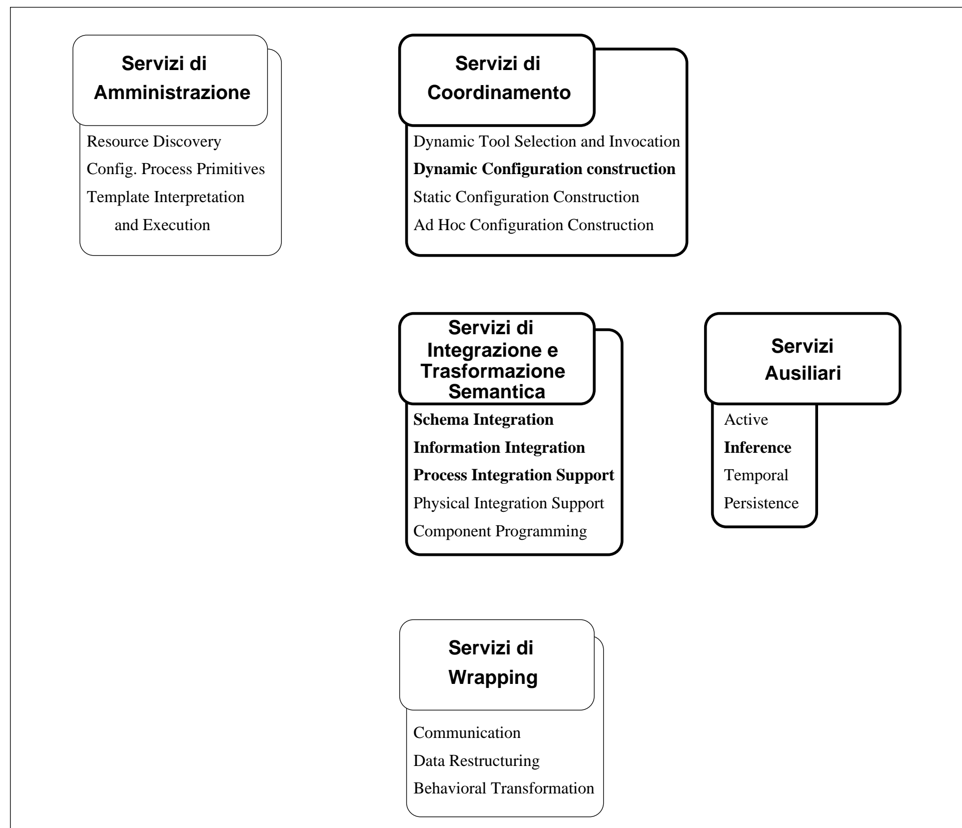



Figura 8.1: Servizi $I^3$ presenti nel mediatore

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

- la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo: il mediatore non si basa infatti su una descrizione predefinita degli sche-

mi delle sorgenti, bensì sulla descrizione che ogni singolo oggetto fa di sè. Oggetti simili provenienti dalla stessa sorgente possono quindi avere strutture differenti, cosa invece non ammessa in un ambiente tradizionale object-oriented;

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

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

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

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

### 8.0.4 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, le relazioni che possono legarli, né tantomeno è banale realizzare una loro coerente integrazione. Mettendo da parte per un attimo le differenze dei sistemi fisici (alle quali dovrebbero pensare i moduli wrapper) i problemi che si è dovuto risolvere, o con i quali occorre giungere a compromessi, sono (a livello di mediazione, ovvero di integrazione delle informazioni) essenzialmente di due tipi:

1. problemi ontologici;
2. problemi semantici.

Vediamoli più in dettaglio.

### 8.0.5 Problemi ontologici

Come riportato nel glossario A, per ontologia si intende, in questo ambito, "l'insieme dei termini e delle relazioni usate in un dominio, che denotano concetti ed oggetti". Con ontologia quindi ci si riferisce a quell'insieme di termini che, in un particolare dominio applicativo, denotano in modo univoco una particolare conoscenza e fra i quali non esiste ambiguità poiché sono condivisi dall'intera comunità di utenti del dominio applicativo stesso. Non è certamente l'obiettivo né di questo paragrafo, né della tesi in generale, dare una descrizione esaustiva di cosa si intenda per ontologia e dei problemi che essa comporta (ancorché ristretti al campo dell'integrazione delle informazioni), ma mi limito a riportare una semplice classificazione delle ontologie (mutuata da Guarino [89, 88], per inquadrare l'ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate) sono essenzialmente quattro:

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

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

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

Come ipotesi semplificativa di questo progetto, si è considerato di muoversi all'interno delle *domain ontology*, ipotizzando quindi che tutte le fonti informative condividano almeno i concetti fondamentali (ed i termini con cui identificarli). A tale scopo, come si vedrà nel seguito, si potrebbe utilizzare sistemi lessicali, quali, ad esempio, WordNet [72, 84].

### 8.0.6 Problemi semantici

Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare, e quindi un insieme di concetti comuni, niente ci dice che i diversi sistemi usino esattamente gli stessi vocaboli per

rappresentare questi concetti, né tantomeno le stesse strutture dati. Poiché infatti le diverse sorgenti sono state progettate e modellate da persone differenti, è molto improbabile che queste persone condividano la stessa "concettualizzazione" del mondo esterno, ovvero non esiste nella realtà una semantica univoca a cui chiunque possa riferirsi.
Se la persona P1 disegna una fonte di informazioni (per esempio DB1) e un'altra persona P2 disegna la stessa fonte DB2, le due basi di dati avranno sicuramente differenze semantiche: per esempio, le coppie sposate possono essere rappresentate in DB1 usando degli oggetti della classe COPPIE, con attributi MARITO e MOGLIE, mentre in DB2 potrebbe esserci una classe PERSONA con un attributo SPOSA.
Come riportato in [95] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma non è l'unica. Le differenze nei sistemi di DBMS possono portare all'uso di differenti modelli per la rappresentazione della porzione di mondo in questione: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate attraverso un modello relazionale, o ad oggetti.
L'obiettivo dell'integratore, che è fornire un accesso integrato ad un insieme di sorgenti, si traduce allora nel non facile compito di identificare i concetti comuni all'interno di queste sorgenti e risolvere le differenze semantiche che possono essere presenti tra di loro. Possiamo classificare queste contraddizioni semantiche in tre gruppi principali:

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

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

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

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

## 8.1 La proposta di integrazione

Come ampiamente presentato nei capitoli precedenti, la continua crescita di informazioni accessibili (sia sulla rete Internet, sia all'interno di una azienda) ha portato all'esigenza di costruire sistemi che aiutino il loro reperimento, ovvero che si preoccupino di ritrovare le informazioni (e questo può essere realizzato attraverso i cosiddetti motori di ricerca) ma che diano anche una visione integrata di queste, in modo da eliminare le incongruenze e le ridondanze necessariamente presenti. Per questo scopo è stato introdotto da Wiederhold [104] il concetto di *mediatore*, un modulo il cui compito principale è quello di reperire ed integrare informazioni da una molteplicià di sorgenti eterogenee. I problemi che devono essere affrontati in questo ambito sono principalmente dovuti ad eterogeneità strutturali e applicative, nonché alla mancanza di una ontologia comune, che porta a differenze semantiche tra le fonti di informazione. A loro volta, queste differenze semantiche possono originare diversi tipi di conflitti, che vanno dalle semplici incongruenze nell'uso dei nomi (quando nomi differenti sono utilizzati in sorgenti diverse per identificare gli stessi concetti) a conflitti strutturali (quando modelli diversi sono utilizzati per rappresentare le stesse informazioni). Come è già stato descritto nel Capitolo 7, in letteratura sono stati presentati diversi sistemi diretti all'integrazione di database convenzionali, come pure applicabili all'integrazione di dati semistrutturati. Di questi, seguendo quanto esposto in [30], si può realizzare una categorizzazione sulla base del diverso tipo di approccio utilizzato, distinguendoli tra *semantici* e *strutturali*. Per quanto riguarda l'approccio *strutturale* (e tra questi l'esempio piú importante è senza dubbio costituito dal progetto TSIMMIS, descritto in Sezione 7.1) si possono sottolineare i seguenti punti caratterizzanti:

- utilizzo di un modello self-describing per trattare tutti i singoli oggetti presenti nel sistema, sopperendo all'eventuale mancanza degli schemi concettuali delle diverse sorgenti;

- inserimento delle informazioni semantiche in modo esplicito attraverso l'utilizzo di regole dichiarative (ed in particolare, in TSIMMIS, attraverso le MSL rule);

- utilizzo di un linguaggio self-describing che facilita l'integrazione anche e soprattutto di dati semi-strutturati;

- l'assenza degli schemi concettuali non permette, in caso di database di grandi dimensioni, una ottimizzazione semantica delle interrogazioni;

- sono eseguibili solo le interrogazioni predefinite dall'operatore (per le quali viene preventivamente memorizzato un piano di accesso), limitando in questo modo la libertà di richieste all'utente del sistema.

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

- per ogni sorgente sono a disposizione dei metadati, codificati nello schema concettuale;

- nello schema concettuale sono pure presenti le informazioni semantiche, che possono essere utilmente sfruttate sia nella fase di integrazione delle sorgenti, sia in quella di ottimizzazione delle interrogazioni;

- deve essere presente nel sistema, per poter descrivere in modo uniforme tutte le informazioni che devono essere condivise, un modello di dati comune;

- viene realizzata una unificazione (parziale o totale) degli schemi concettuali (realizzata in modo automatico o manuale), per arrivare alla definizione di uno schema globale.

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

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

2. la natura stessa degli schemi che utilizzano i modelli ad oggetti, attraverso l'uso delle primitive di generalizzazione e di aggregazione, permette la riorganizzazione delle conoscenze estensionali;

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

4. ampi sforzi sono sono già stati realizzati per lo sviluppo di standard rivolti agli oggetti: CORBA [102] per lo scambio di oggetti attraverso sistemi diversi; ODMG-93 [57] (e con esso i modelli ODM e ODL per la descrizione degli schemi, e OQL come linguaggio di interrogazione) per gli object-oriented database;

5. l'adozione di una semantica di *mondo aperto* permette il superamento delle problematiche legate all'uso di un convenzionale modello ad oggetti per la descrizione di dati semistrutturati: gli oggetti di una classe condividono una struttura minima comune (che è quindi la descrizione della classe stessa), ma possono avere ulteriori proprietà non esplicitamente comprese nella struttura della classe di appartenenza.

Vediamo ora una sommaria descrizione del nostro approccio. All'interno del sistema sono stati adottati un modello comune dei dati ($\mathrm{ODM}_{I^3}$) ed un linguaggio di definizione comune ($\mathrm{ODL}_{I^3}$) per descrivere l'integrazione delle informazioni. $\mathrm{ODM}_{I^3}$ e $\mathrm{ODL}_{I^3}$ sono definiti come sottoinsiemi dei corrispondenti modello e linguaggio nello standard ODMG-93 [57], seguendo la proposta di standard per un linguaggio di mediatore sviluppata dal gruppo di lavoro $I^3$/POB [45]. In piú, $\mathrm{ODL}_{I^3}$ introduce nuovi costruttori per il supporto al processo di integrazione. MOMIS si appoggia alla Description Logics OLCD (Object Language with Complements allowing Descriptive cycles [16, 25]), come linguaggio interno e ad ODB-Tools [21, 18, 2] come tool di supporto per il ragionamento e l'inferenza per l'integrazione di sorgenti e l'ottimizzazione di query. D'altra parte, per supportare il processo semiautomatico di integrazione degli schemi, viene utilizzato ARTEMIS, un tool basato sulle tecniche di clustering gerarchico.

Nel processo di integrazione viene dapprima costruito un Thesaurus comune (Common Thesaurus) di relazioni terminologiche (*terminological relationships*) basate sulla descrizione delle sorgenti in linguaggio $\mathrm{ODL}_{I^3}$. Il thesaurus rappresenta il punto di partenza per la seconda fase, dove le tecniche di clustering sono impiegate per identificare le classi che descrivono informazioni simili in sorgenti differenti e fonderle in una descrizione globale ed integrata delle sorgenti analizzate. Gli aspetti salienti ed innovativi della nostra proposta riguardano i seguenti aspetti:

- Modellazione di schema (che chiameremo *object patterns*) per le sorgenti semistrutturate, investigando nello stesso modo che in [73]. Sono

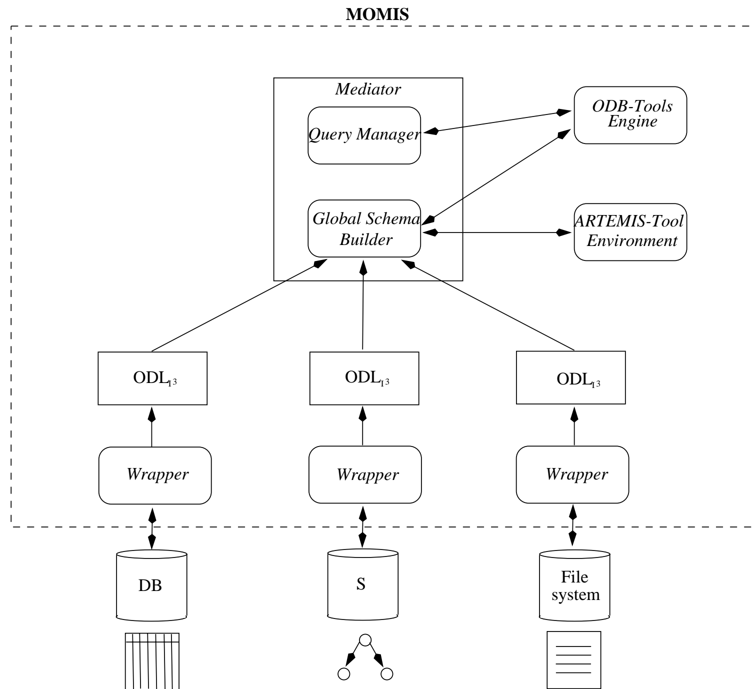



Figura 8.2: Architettura del sistema MOMIS

presentate e discusse le estensioni al linguaggio ODL$_{I^3}$ ed alla corrispondente DL OLCD per permettere l'introduzione degli *object patterns* nei processi di integrazione e di query processing.

- Estensione delle procedure di clustering basate sulle tecniche di affinitá per permettere il trattamento degli *object patterns*.

- Query processing e ottimizzazione usando la conoscenza racchiusa nello schema integrato risultante. In quest'ambito viene inoltre descritto il ruolo di ODB-Tools relativamente alla decomposizione ed alla ottimizzazione delle query globali.

## 8.2 L'architettura di MOMIS

MOMIS é stato progettato per fornire un accesso integrato ad informazioni eterogenee memorizzate sia in database di tipo tradizionale (e.g. relazionali, object-oriented) o file system, sia in sorgenti di tipo semistrutturato.

Seguendo l'architettura di riferimento $I^3$ [1], in MOMIS si possono distinguere quattro componenti principali (come si puó vedere da Fig. 8.2):

1. *Wrapper*: posti al di sopra di ciascuna sorgente, sono i moduli che rappresentano l'interfaccia tra il mediatore vero e proprio e le sorgenti locali di dati. La loro funzione è duplice:

   - in fase di integrazione, forniscono la descrizione delle informazioni in essa contenute. Questa descrizione viene fornita attraverso il linguaggio $\text{ODL}_{I^3}$ (descritto in Sezione 8.4);
   - in fase di query processing, traducono la query ricevuta dal mediatore (espressa quindi nel linguaggio comune di interrogazione $\text{OQL}_{I^3}$, che sarà definito in analogia al linguaggio OQL) in una interrogazione comprensibile (e realizzabile) dalla sorgente stessa. Devono inoltre esportare i dati ricevuti in risposta all'interrogazione, presentandoli al mediatore attraverso il modello comune di dati utilizzato dal sistema;

2. *mediatore*: è il cuore del sistema, ed è composto da due moduli distinti.

   - Global Schema Builder (GSB): è il modulo di integrazione degli schemi locali, che, partendo dalle descrizioni delle sorgenti espresse attraverso il linguaggio $\text{ODL}_{I^3}$, genera un unico schema globale da presentare all'utente. Questa fase di integrazione, realizzata in modo semi-automatico con l'interazione del progettista del sistema, fa uso degli ODB-Tools (sfruttati dal modulo software $\mathbf{SIM}_1$) e delle tecniche di clustering;
   - Query Manager (QM): è il modulo di gestione delle interrogazioni. In particolare, genera le query in linguaggio $\text{OQL}_{I^3}$ da inviare ai wrapper partendo dalla singola query formulata dall'utente sullo schema globale. Servendosi delle tecniche di Logica Descrittiva, il QM genera automaticamente la traduzione della query sottomessa nelle corrispondenti sub-query delle singole sorgenti.

3. *ODB-Tools Engine*, un tool basato sulla OLCD Description Logics [16, 25] che compie la validazione di schemi e l'ottimizzazione di query [21, 18, 2].

4. *ARTEMIS-Tool Environment*, un tool che compie analisi e clustering di schemi [51].

Il principale scopo che ci si è preposti con MOMIS è la realizzazione di un sistema di mediazione che, a differenza di molti altri progetti analizzati, contribuisca a realizzare, oltre alla fase di query processing, una reale integrazione delle sorgenti.

**Processo di Integrazione.** L'integrazione delle sorgenti informative strutturate e semistrutturate viene compiuta in modo semi-automatico, utilizzando le descrizioni degli schemi locali in linguaggio $\mathrm{ODL}_{I^3}$ e combinando le tecniche di Description Logics e di clustering. Le attivitá compiute sono le seguenti:

1. *Estrazione di relazioni terminologiche*, grazie al supporto di ODB-Tools. Durante questo passo viene costruito un Thesaurus Comune di *relazioni terminologiche*. Le relazioni terminologiche esprimono la conoscenza di inter-schema su sorgenti diverse e corrispondono alle asserzioni intensionali utilizzate in [56]. Le relazioni terminologiche sono derivate in modo semi-automatico a partire dalle descrizioni degli schemi in $\mathrm{ODL}_{I^3}$, attraverso l'analisi strutturale e di contesto delle classi coinvolte, utilizzando ODB-Tools e le tecniche di Description Logics. L'estrazione delle relazioni terminologiche sará discussa in Sezione 8.6.

2. *Affinity-based clustering di classi* $\mathit{ODL}_{I^3}$, con il supporto dell'ambiente ARTEMIS-Tool, le relazioni terminologiche contenute nel Thesaurus vengono utilizzate per valutare il livello di affinitá tra le classi $\mathrm{ODL}_{I^3}$ in modo di identificare le informazioni da essere integrate a livello globale. A tal fine, ARTEMIS calcola i coefficienti che misurano il livello di affinitá delle classi $\mathrm{ODL}_{I^3}$ basandosi sia sui nomi che sugli attributi. Le classi $\mathrm{ODL}_{I^3}$ con maggiore affinitá vengono raggruppate utilizzando le tecniche di clustering [67].

3. *Costruzione dello schema globale di mediatore*, con il supporto di ODB-Tools i cluster di classi $\mathrm{ODL}_{I^3}$ affini sono analizzate per costruire lo schema globale del Mediatore. Per ciascun cluster viene definita una classe globale $\mathrm{ODL}_{I^3}$ che rappresenta tutte le classi che sono riferite al cluster, e che é caratterizzata dall'unione dei loro attributi. L'insieme delle classi globali definite costituisce lo schema globale di Mediatore che deve essere usato per porre le query alle sorgenti locali integrate. In questa fase OLCD e ODB-Tools sono utilizzati per realizzare una generazione semi-automatica delle classi globali.

**Query processing e ottimizzazione.** Quando l'utente pone una query sullo schema globale, MOMIS analizza la query e produce un insieme di sub-query che saranno inviate a ciascuna sorgente informativa coinvolta. In accordo ad altri approcci proposti in quest'ambito [7, 8], il processo consiste di due attivitá principali:

1. *Ottimizzazione Semantica.* Il Mediatore agisce sulla query sfruttando la tecnica di ottimizzazione semantica supportata da ODB-Tools in modo da ridurre il costo del query access plan. L'ottimizzazione é basata sull'inferenza logica a partire dalla conoscenza contenuta nei vincoli di integritá dello schema globale.
2. *Formulazione del piano di accesso.* Dopo aver ottenuto la query ottimizzata, viene generato l'insieme di sub-query relative alle sorgenti coinvolte. A questo scopo, il Mediatore utilizza l'associazione tra classi globali e locali per esprimere la query globale in termini degli schemi locali.

L'Ottimizzazione Semantica e il query plan sará descritto in Sezione 8.10.

Pur non essendo centrale per il nostro sistema, vogliamo descrivere il processo che rende disponibile al Mediatore gli schemi sorgenti in linguaggio $\text{ODL}_{I^3}$. Per quanto riguarda i database strutturati convenzionali, la descrizione dello schema é sempre disponibile e puó quindi essere tradotto automaticamente in linguaggio $\text{ODL}_{I^3}$ da uno specifico wrapper. Diverso é il discorso delle sorgenti semistrutturate, dove, a priori, non é definito alcuno schema. Per questa ragione risulta necessaria una procedura di estrazione dello schema per le sorgenti semistrutturate (e.g., nella forma di dataguides [73]). In MOMIS introduciamo perció il concetto di *object pattern* per rappresentare a livello intensionale la struttura degli oggetti di una sorgente semistrutturata. Questo processo verrá discusso nella sezione seguente.

### 8.2.1 Semistructured data and object patterns

L'incremento nel numero e nella dimensione delle sorgenti informative che in questi anni si sono rese disponibili all'utente generico e, a volte, casuale, è stato accompagnato da un'esplosione nella varietà dei formati in cui i dati vengono rappresentati. Nella maggior parte dei casi, purtroppo, sebbene si possa riconoscere nei dati una sorta di struttura, questa è talmente irregolare da non poter essere facilmente riconducibile nè ai consolidati modelli relazionali [100] e neppure a quelli più ricchi quali quelli ad oggetti [57]. In genere si parla di questi dati in termini di dati semistrutturati (semistructured data), come confermato da diversi interventi presenti in letteratura [3, 42]. A causa della natura frammentata dei dati, l'enfasi degli interventi viene posta sull'importanza di derivare una rappresentazione coincisa (indicata brevemente come "summary") della struttura in modo tale da fornire all'utente che si trova ad interagire con le informazioni un'idea della struttura e del contesto

riguardante la sorgente dei dati. Questa possibilità di giungere alla definizione di una struttura per dati semistrutturati facilita la formulazione di query e potrebbe essere usata per la fase di ottimizzazione delle query stesse.

La caratteristica di base dei dati semistrutturati risiede nella loro natura che può essere definita "auto-descrittiva" ("self-describing"). Questo significa che le informazioni, che generalmente vengono associate allo schema, sono direttamente specificate nei dati. Come detto, in letteratura vari sono i modelli proposti per la rappresentazione dei dati semistrutturati [42, 44, 108]. In accordo a questi modelli, la nostra proposta prevede di rappresentare sorgenti semistrutturate come grafi ad albero etichettati, dove i dati semistrutturati possono rappresentare sia i nodi che le etichette sugli archi. Infatti, supponiamo di utilizzare sorgenti semistrutturate conformi al modello OEM [4, 108], la cui definizione è la seguente:

**Definizione 11 (Oggetto Semistrutturato)** *Un* oggetto semistrutturato*, indicato con so, è una tripla nella forma* $\langle$`id`, `label`, `value`$\rangle$ *dove:*

- `id` *è l'idenficatore dell'oggetto;*
- `label` *è una stringa che descrive ciò che l'oggetto rappresenta;*
- `value` *è il valore dell'oggetto, che può essere atomico o complesso.*

Per quanto concerne i valori atomici, essi possono essere *integer*, *real*, *string*, *image*, mentre un valore complesso è un insieme di oggetti semistrutturati, cioè un insieme di coppie (*id,label*). In analogia a questa caratterizzazione, definiamo *oggetto semistrutturato atomico* un oggetto a valore atomico, e *oggetto semistrutturato complesso* un oggetto il cui valore è complesso.

Per esempio,$\langle$`16`, `name`, "`Ann Red`"$\rangle$ rappresenta un oggetto semistrutturato atomico con valore di tipo stringa, mentre l'oggetto $\langle$`8`, `exam`, {`(24, date), (25, type), (26, outcome)`}$\rangle$ rappresenta un oggetto semistrutturato complesso il cui valore è dato da tre oggetti atomici. L'oggetto complesso può quindi essere pensato come il padre di tutti gli oggetti che definiscono il suo valore (detti oggetti figli). In generale, un dato oggetto può avere uno o più genitori: nella nostra notazione indichiamo che un oggetto $so'$ è un oggetto figlio di un altro oggetto $so$ tramite $so \rightarrow so'$ e indichiamo con $label(so)$ l'etichetta di $so$. Perciò, intuitivamente, una sorgente semistrutturata $S$ può essere vista come un grafo orientato, in cui i nodi rappresentano gli oggetti semistrutturati e gli archi orientati rappresentano la relazione tra un oggetto complesso e tutti gli oggetti atomici che ne individuano il suo valore.

Un esempio di sorgente semistrutturata è rappresentato in Fig. 8.2.1, dove sono presenti informazioni relative al Dipartimento di Cardiologia (`Cardiology Department`) di un dato ospedale.

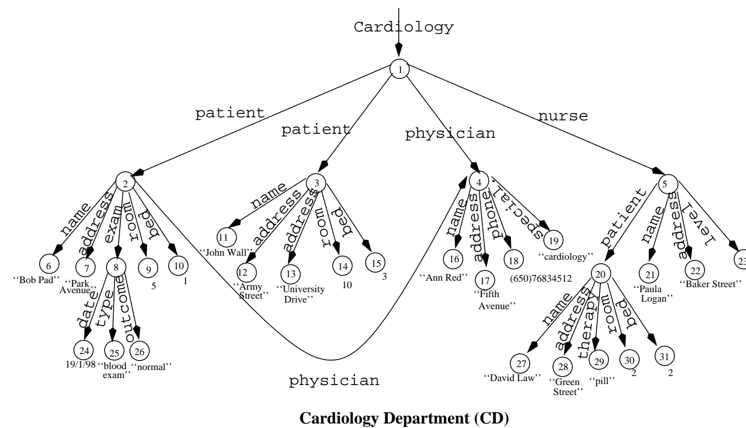



Figura 8.3: Cardiology Department (CD)

Nei modelli di dati semistrutturati, le etichette sono il più possibile esplicative e possono essere utilizzate per raggruppare gli oggetti assegnando la stessa etichetta ad oggetti correlati. Per questa ragione, data una sorgente semistrutturata $S$, vogliamo scoprire i tipi differenti di oggetti memorizzati attraverso un approccio basato sui pattern che tenga conto della semantica di mondo aperto caratteristico delle Description Logics. L'approccio si puó riassumere brevemente nel seguente modo: tutti gli oggetti complessi $so$ di $S$ vengono partizionati in base alle loro etichette e valori in insiemi disgiunti, indicati come $set_l$, in modo tale che tutti gli oggetti che fanno riferimento allo stesso insieme abbiano la stessa etichetta $l$. Successivamente ad ogni insieme associamo un *object pattern*. Formalmente, un *object pattern* é definito nel seguente modo:

**Definizione 12 (Object pattern)** *Poni $set_l$ un insieme di oggetti definito sulla sorgente semistrutturata $S$. L'*object pattern *dell'insieme $set_l$ é una coppia nella forma $\langle l,A \rangle$, dove $l$ é l'etichetta degli oggetti correlati all'insieme $set_l$, ed $A = \bigcup label(so')$ tale che esiste almeno un oggetto $so \in set_l$ con $so \rightarrow so'$.*

Partendo da questa definizione, é facile capire come un *object pattern* sia rappresentativo di tutti i diversi oggetti che descrivono lo stesso concetto in una data sorgente semistrutturata. Piú specificatamente, $l$ indica il concetto e $A$ le proprietá (dette anche attributi) che caratterizzano il concetto

Patient-pattern = (`Patient`,{`name`,`address`,`exam`*,`room`,`bed`,`therapy`*,`physician`*})
Physician-pattern = (`Physician`,{`name`,`address`,`phone`,`specialization`})
Nurse-pattern = (`Nurse`,{`name`,`address`,`level`,`patient`})
Exam-pattern = (`Exam`,{`date`,`type`,`outcome`})

Figura 8.4: Gli object pattern della sorgente semistrutturata di esempio

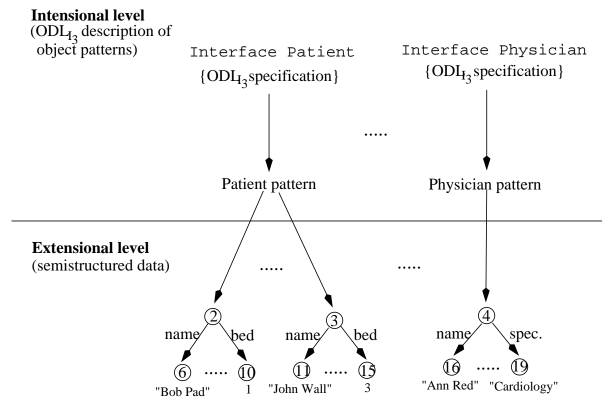



Figura 8.5: Livelli di astrazione degli oggetti semistrutturati

all'interno della sorgente. Poiché gli oggetti semistrutturati possono essere eterogenei, a volte le etichette nell'insieme $A$ di un *object pattern* possono essere definite solo per alcuni oggetti dell'insieme $\text{set}_l$ e non per tutti: chiameremo queste particolari etichette come "optional" e verranno indicate con il simbolo "*".

In Fig. 8.2.1 sono indicati tutti gli *object pattern* della sorgente semistrutturata considerata come esempio. Sono stati definiti quattro *object pattern*: `Patient` che contiene informazioni riguardo ai pazienti; `Physician` e `Nurse` contenenti informazioni riguardo al personale medico; `Exam` contenente informazioni riguardo la `date`, il `type` e `outcome` di un esame clinico.

Come detto la descrizione degli *object pattern* segue la semantica di mondo aperto *open world semantics* tipica dell'approccio seguito in Description Logics [27, 35, 47, 105]. Gli oggetti di un pattern condividono una struttura di minimo rappresentata dalle proprietá non opzionali, ma possono avere altre proprietá addizionali che caratterizzano il singolo oggetto (o un numero ridotto di oggetti) che chiamiamo optional. In questo modo, gli oggetti di una sorgente semistrutturata possono evolversi ed aggiungere nuove proprietá, pur rimanendo istanze valide dell'*object pattern* corrispondente e quindi mantenendo efficaci le query sull'*object pattern*.

**Intensive Care Department (ID)**

```
Patient(code, first_name, last_name, address, test, doctor_id)
Medical_Staff(id, first_name, last_name, phone, address, availability, position)
Test(number, type, date, laboratory, result)
Dis_Patient(code, date, note)
```

Figura 8.6: Intensive Care Department (ID)

Il nostro approccio si pone quindi l'obiettivo di distinguere, all'interno di una sorgente semistrutturata, la parte relativa alla struttura da quella dei dati: il livello intensionale contiene ciascuna descrizione degli *object pattern* secondo la descrizione $\mathrm{ODL}_{I^3}$. Il livello estensionale contiene invece gli oggetti semistrutturati (vedi Fig. 8.5).

## 8.3 Esempio di riferimento

In questo capitolo verrà utilizzato, per meglio chiarire tutti i passaggi che vengono effettuati sia nella fase di integrazione che in quella di query processing, il seguente esempio di riferimento riguardante un dominio applicativo di un'organizzazione medica.

Si consideri un ambiente ospedaliero ed in particolare i Dipartimenti di cardiologia (`Cardiology`) e quello di terapia intensiva (`Intensive Care`), dove, ovviamente, c'é l'esigenza di condividere le informazioni riguardo i propri pazienti. Il dipartimento `Cardiology` (brevemente indicato come `CD`) contiene oggetti di tipo semistrutturato relativi a pazienti con malattie di tipo ischemico, cardiocircolatorio ed ipertensione, ed informazioni riguardo il personale medico afferente il dipartimento. In Fig. 8.2.1 é riportata una porzione di esempio dei dati: abbiamo un oggetto complesso radice con quattro figli anch'essi di tipo complesso, due pazienti, un medico ed una nurse. (Ovviamente nell'applicazione reale ci sono molti pazienti, medici e nurse). Ciascun `Patient` é dotato di oggetti atomici `name`, `address`, `room`, e `bed`. `Physician` contiene a sua volta quattro oggetti atomici (i.e., `name`, `address`, `phone`, e `specialization`), mentre `Nurse` contiene come valori tre oggetti atomici (i.e., `name`, `address`, e `level`) ed uno complesso (i.e., il `patient` di cui é responsabile ).

Il dipartimento `Intensive Care` (`ID`) memorizza le proprie informazioni in un database relazionale relative a: pazienti la cui diagnosi riguarda trauma o infarto e sullo staff medico. Il database contiene quattro relazioni: `Patient`, `Medical_Staff`, `Test` e `Dis_Patient` (vedere Fig. 8.6). Le informa-

zioni personali relative ai pazienti sono mantenute nella relazione `Patient`. `Dis_Patient` é un sottotipo di `Patient` e contiene informazioni riguardo ai pazienti dimessi. Ciascun "clinician", cioé il personale medico quali dottori, infermieri, farmacisti (che costituiscono la relazione `Medical_Staff`) é caratterizzato da `first_name`, `last_name`, `phone`, `address`, `availability` e `position`. La relazione `Test` é usata per mantenere informazioni riguardanti tutti gli esami dei pazienti.

## 8.4 Il linguaggio ODL$_{I^3}$

Allo scopo di facilitare la comunicazione tra i wrapper ed il Mediatore, introduciamo un linguaggio di definizione di dati di alto livello, chiamato ODL$_{I^3}$.

In accordo alle raccomandazioni della proposta di standardizzazione per linguaggi di mediazione [46], secondo la quale i diversi sistemi di mediazione avrebbero potuto supportare sorgenti con modelli complessi, come quelli ad oggetti, e sorgenti molto più semplici, come file di strutture, ed al diffondersi del modello di dati ad oggetti (e il suo standard ODMG-93), ODL$_{I^3}$ é una estensione al linguaggio standard ODL che supporta le necessitá del nostro sistema di integrazione intelligente di informazioni.

Principali caratteristiche del linguaggio sono:

- supporto di sorgenti strutturate (database relazionali, ad oggetti, e file system) e semistrutturate;

- descrizione di ciascuna fonte di informazioni secondo il modello comune dei dati, e quindi utilizzare il concetto di *classe*, indipendentemente dal modello originale utilizzato. Sarà compito del wrapper provvedere alla traduzione dal modello originale di descrizione all'ODL$_{I^3}$, aggiungendo eventualmente in questa descrizione tutte le informazioni necessarie al mediatore (il nome e il tipo della sorgente, ... );

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

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

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

- traduzione automatica e trasparente all'utente delle descrizioni nella logica descrittiva OLCD, con conseguente possibilitá di utilizzarne le capacità nei controlli di consistenza e nell'ottimizzazione semantica delle interrogazioni;

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

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

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

Utilizzando questo linguaggio, il wrapper compie la traduzione delle classi da integrare e le fornisce al mediatore: da sottolineare che le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema, e quindi interrogabili. Non è quindi detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente, bensì ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa.
La sintassi del linguaggio ODL$_{I^3}$è riportata, in BNF, nell'Appendice B.

Come detto, object pattern e schemi sorgenti sono tradotti in descrizioni ODL$_{I^3}$: per gli object pattern la traduzione é diretta. In particolare, dato un a pattern $\langle l, A\rangle$ o una relazione di una sorgente relazionale abbiamo che: i) un nome di classe ODL$_{I^3}$ corrisponde a $l$ o al nome della relazione, e ii) per ciascuna etichetta $l' \in A$ o attributo della relazione, viene definito un attributo nella corrispondente classe ODL$_{I^3}$. Come esempio mostriamo le descrizioni delle classi `CD.Patient` e `ID.Medical_Staff`, rimandando invece all'Appendice D per l'intera descrizione dell'esempio di riferimento di Sezione 8.3:

```
interface Patient

 ( source semistructured Cardiology_Department )
{  attribute string    name;
   attribute string    address;
   attribute set<Exam> exam*;
   attribute integer   room;
   attribute integer   bed;
```

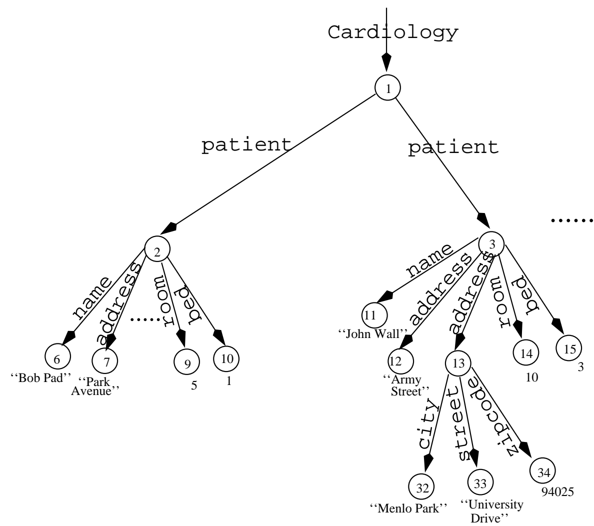



Figura 8.7: Un esempio di oggetti fortemente eterogenei

```
   attribute string    therapy*;
   attribute set<Physician> physician*;
   };

interface Medical_Staff
 ( source relational Intensive_Care
   extent Medical_Staffers
   key id )
{  attribute string  id;
   attribute string  first_name;
   attribute string  last_name;
   attribute integer phone;
   attribute string  address;
   attribute string  availability;
   attribute string  position; };
```

Un caso particolare avviene quando un oggetto "fortemente" eterogeneo é definito nella sorgente semistrutturata. In riferimento alla nostra sorgente semistrutturata CD di Fig. 8.2.1, consideriamo l'oggetto:
$so = \langle 2, \texttt{patient}, \{(6, \texttt{name}), (7, \texttt{address}), (8, \texttt{exam}), (9, \texttt{room}), (10, \texttt{bed})\}\rangle$.

Supponiamo che l'oggetto relativo al paziente `3` sia presente avendo un oggetto `address` non-atomico, cioé che `address` contenga come valori tre oggetti atomici (`city, street, zipcode`) (vedere Fig. 8.7). In questo

caso, l'estrazione del pattern dovrebbe essere leggermente modificata. In particolare viene estratto un nuovo pattern per address, `address-pattern` = (`address`, {`city`, `street`, `zipcode`}). Per comprendere e gestire anche questo tipo di eterogeneitá, in $\mathrm{ODL}_{I^3}$ definiamo il costruttore **union** per le classi. La definizione in $\mathrm{ODL}_{I^3}$ dei pattern `Patient` e `address` é mostrata Fig. 8.8.

```
interface Address
 ( source semistructured Cardiology_Department )
{  attribute string city;
   attribute string street;
   attribute string zipcode; };
 union
{  string; };

interface Patient
 ( source semistructured Cardiology_Department )
{  attribute string   name;
   attribute Address  address;
   .............................. };
```

Figura 8.8: Un esempio di tipo unione

La semantica del costruttore di union e di optional viene descritta nella prossima sezione, utilizzando la Description Logic OLCD.

## 8.5 Estensioni al formalismo OLCD per l'integrazione

Il formalismo OLCD é stato introdotto precedentemente nel capitolo 2.

L'estensione principale ad OLCD riguarda il costruttore di *union* ($\sqcup$). L'operatore *union* ($\sqcup$) puó essere usato per rappresentare la semantica dell'operatore di `union` del linguaggio $\mathrm{ODL}_{I^3}$. É stato formalizzato in [16], con il significato di unione di tutte le possibili istanze di un tipo.

Per esempio, il pattern `Address` di Fig. 8.8 viene tradotto in OLCD come segue:

$$\sigma_V(\texttt{Address}) \;=\; [\;\texttt{city} : \texttt{String}, \texttt{street} : \texttt{String}, \texttt{zipcode} : \texttt{String}\;]$$

L'operatore di *union* ($\sqcup$) é anche utile per definire nel nostro formalismo il concetto di attributo opzionale. Infatti un attributo opzionale specifica che il dato per una specifica istanza puó esistere oppure essere assente. In OLCD

descriviamo questo concetto come union tra il dominio dell'attributo e il tipo *bottom* ($\perp$).

Nel nostro esempio il pattern `Patient` puó essere rappresentato come segue: [1]:

$$\sigma_P(\texttt{Patient}) = \triangle\, [\ \texttt{name} : \texttt{String}, \texttt{address} : \texttt{String}, \texttt{exam} : \{\texttt{Exam}\}, \texttt{room} : \texttt{Integer}, \texttt{bed} : \texttt{Integer}, \texttt{therapy} : \texttt{String} \sqcup \perp, \texttt{physician} : \{\texttt{Physician}\}\ ]$$

Description Logic, e perció OLCD, permette, utilizzando la semantica dei tipi virtuali e del *type as set* per la descrizione dei tipi, per fornire tecniche di ragionamento rilevanti: calcolo della relazione di *subsumption* tra tipi (i.e. relazioni "isa" implicata dalla descrizione dei tipi), decidere l'*equivalence* tra tipi e rilevazione di tipi *incoerenti* (i.e., sempre empty).

Per comprendere l'utilizzo della subsumption nel contesto di attributi optional, supponiamo di considerare i tipi valore `A` e `B`,

$$\sigma_V(\texttt{A}) = [\ \texttt{att1} : \texttt{String}, \texttt{att2} : \texttt{String}\ ]$$
$$\sigma_V(\texttt{B}) = [\ \texttt{att1} : \texttt{String}, \texttt{att2} : \texttt{String} \sqcup \perp\ ]$$

Calcolando la subsumption tra i tipi `A` e `B` otteniamo che `B` subsume `A` (A isa B) anche se non é stato dichiarato esplicitamente.

## 8.6 Estrazione di Relazioni Terminologiche

Lo scopo di questa fase è la costruzione di un Thesaurus di *relazioni terminologiche* che rappresenti la conoscenza a disposizione sulle classi da integrare (ovvero sui nomi delle classi, sugli attributi) e che sarà la base per il calcolo di affinità tra le classi stesse. Definiamo quindi un modello di rappresentazione delle classi. Sia $S = \{S_1, S_2, \ldots, S_N\}$ un insieme di schemi di $N$ sorgenti eterogenee che devono essere integrate. Come richiesto dall'ODL$_{I^3}$, ogni schema sorgente $S_i$ è composto da un insieme di *classi*: una classe $c_{ji} \in S_i$ è caratterizzata da un nome e da un insieme di attributi, $c_{ji} = \langle n_{c_{ji}}, A(c_{ji})\rangle$. A sua volta, ogni attributo $a_h \in A(c_{ji})$, con $h = 1, \ldots, n$, è definito da una coppia $a_h = \langle n_h, d_h\rangle$, dove $n_h$ è il nome e $d_h$ è il dominio associato ad $a_h$. Si è inoltre ipotizzato che, per identificare in modo univoco all'interno del mediatore un nome (sia esso di attributo, sia esso di classe), sia rispettivamente necessaria la coppia *nome_sorgente*,*nome classe* e *nome_sorgente*,*nome_attributo*. Si parlerà inoltre genericamente di *termine* $t_i$ indicando con esso un nome di classe o di attributo.

Le relazioni che si possono definire all'interno del Thesaurus sono le seguenti:

---

[1] $\sigma_P$ e $\sigma_V$ introducono il tipo primitivo e virtuale rispettivamente

- SYN (synonym-of): definita tra due termini $t_i$ e $t_j$, con $t_i \neq t_j$, che sono considerati sinonimi, ovvero che possono essere interscambiati nelle sorgenti, identificando lo stesso concetto del mondo reale. Un esempio di relazione SYN nel nostro esempio è ⟨`ID.Test` SYN `CD.Exam`⟩. SYN é simmetrica, cioé, $t_i$ SYN $t_j \Rightarrow t_j$ SYN $t_i$.

- BT (broader-term): definita tra due termini $t_i$ e $t_j$ tali che $t_i$ ha un significato più ampio, più generale di $t_j$. Un caso di BT, nel nostro esempio, può essere ⟨`ID.Medical_Staff` BT `CD.Nurse`⟩.

- NT (narrower-term): concettualmente è la stessa relazione espressa con una BT, intesa dall'altro punto di vista, dunque $t_i$ BT $t_j \rightarrow t_j$ NT $t_i$. Lo stesso esempio potrebbe infatti essere ⟨`ID.Medical_Staff` NT `CD.Nurse`⟩.

- RT (related-term): definita tra due termini $t_i$ e $t_j$ che sono generalmente usati nello stesso contesto, tra i quali esiste comunque un legame generico. Per esempio, possiamo avere la seguente ⟨`CD.Patient` RT `CD.Exam`⟩. La relazione é simmetrica.

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

1. **Estrazione automatica di relazioni dagli schemi sorgenti.** Sfruttando le informazioni semantiche presenti negli schemi strutturati (sia basati sui modelli ad oggetti, sia relazionali) può essere identificato in modo automatico un insieme di relazioni terminologiche. In particolare durante questa preliminare fase di analisi, si puó automaticamente estrarre:

   (a) **schemi ad oggetti**:

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

   (b) **schemi relazionali**:

- relazioni BT e NT derivate dalle gerarchie di generalizzazione (foreign key definita su primary key in entrambe le relazioni: `ID.Dis_Patient` and `ID.Patient` );
- relazioni RT derivate dalle foreign key.

**Esempio 1** Considerando l'esempio di riferimento, le relazioni terminologiche automaticamente estratte sono le seguenti:

⟨`ID.Patient` RT `ID.Medical_Staff`⟩
⟨`ID.Patient` RT `ID.Test`⟩
⟨`CD.Patient` RT `CD.Physician`⟩
⟨`CD.Patient` RT `CD.Exam`⟩
⟨`ID.Patient` BT `ID.Dis_Patient`⟩
⟨`CD.Nurse` RT `CD.Patient`⟩

2. **Revisione/Integrazione delle relazioni.** Interagendo con il modulo, il progettista deve inserire tutte le relazioni terminologiche che non sono state estratte nel passo precedente, ma che devono comunque essere presenti per pervenire ad una esatta integrazione delle sorgenti. In particolare, possono essere inserite relazioni che coinvolgono classi appartenenti a schemi diversi, come pure relazioni che si riferiscono a nomi di attributi. Da sottolineare che, durante questa fase, tutte le relazioni inserite dal progettista hanno carattere esclusivamente *terminologico*, escludendo quindi qualunque considerazione sulle estensioni. È quindi una fase che, nei prossimi sviluppi del progetto, potrebbe facilmente essere coadiuvata dall'uso di un dizionario che evidenzi eventuali termini sinonimi, o correlati tra loro. Un approccio potrebbe essere quello di rivolgersi a sistemi lessicali (vedere, per esempio WordNet [72, 84]), che prevedono relazioni terminologiche tra termini a priori.

   In generale, relazioni terminologiche esplicite possono correlare classi $\mathrm{ODL}_{I^3}$ le cui descrizioni presentano conflitti semantici rispetto alle relazioni di subsumption/generalizzazione ed equivalenza, poiché queste ultime specificano relazione di supertipo/sottotipo e di uguaglianza tra tipi delle classi coinvolte.

   Per esempio, supponiamo che una relazione di SYN sia definita tra due classi aventi la stessa struttura (per esempio `ID.Test` e `CD.Exam`). Per rendere compatibile questa relazione terminologica con una relazione di equivalenza per ODB-Tools é necessario uniformare la descrizione di entrambe le classi. Il risultato della modifica per le classi esaminate diventa:

```
interface Test
( ... )
{ attribute string  type;
  attribute integer date;
  attribute string  laboratory;
  attribute string  result;
  attribute string  outcome; };

interface Exam ( ... )
{ attribute integer date;
  attribute string  type;
  attribute string  outcome;
  attribute string  laboratory;
  attribute string  result; };
```

Lo stesso tipo di problematica si evidenzia per la relazione di BT tra `ID.Patient` e `ID.Dis_Patient`. Tradurre questa relazione terminologica nella corrispondente relazione di subsumption implica la modifica delle descrizioni nel seguente modo:

```
interface Patient
(...)
{ attribute string  code;
  attribute string  first_name;
  attribute string  last_name;
  attribute string  address;
  attribute string  test;
  attribute string  doctor_id; };

interface Dis_Patient
( ...  )
{ attribute string  code;
  attribute integer date;
  attribute string  note;
  attribute string  position;
  attribute string  first_name;
  attribute string  last_name;
  attribute string  address;
  attribute string  test;
  attribute string doctor_id;
};
```

Infine quando viene asserita una nuova relazione di RT, ad esempio tra `CD.Patient` e `ID.Test`, un nuovo attributo di aggregazione viene inserito nel seguente modo:

```
interface Patient
( ... )
{  attribute string          name;
   attribute string          address;
   attribute set<Exam>       exam*;
   attribute integer         room;
   attribute integer         bed;
   attribute string          therapy*;
   attribute set<Physician>  physician*;
   attribute Test            newRT1};

interface Test
( ... )
{  attribute string  type;
   attribute integer date;
   attribute string  laboratory;
   attribute string  result;
};
```

Queste trasformazioni conducono alla generazione automatica di uno "schema virtuale" contenente, per ciascuno schema locale, le descrizioni modificate delle classi; tale schema viene utilizzato da ODB-Tools per eseguire ulteriori inferenze di relazioni per arricchire il Thesaurus.

**Esempio 1** Nel nostro esempio, supponiamo che il progettista inserisca le seguenti relazioni terminologiche relative sia a classi che ad attributi:

⟨`ID.Medical_Staff` BT `CD.Physician`⟩
⟨`ID.Medical_Staff` BT `CD.Nurse`⟩
⟨`ID.Test` SYN `CD.Exam`⟩
⟨`ID.Patient` SYN `CD.Patient`⟩
⟨`CD.Patient.physician` BT `ID.Patient.doctor_id`⟩
⟨`CD.Patient.name` BT `ID.Patient.first_name`⟩
⟨`CD.Patient.name` BT `ID.Patient.last_name`⟩
⟨`CD.Nurse.level` SYN `ID.Medical_Staff.position`⟩
⟨`CD.Exam.outcome` SYN `ID.Test.result`⟩

3. **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. In questo modo le relazioni terminologiche vengono distinte in *valide* e *invalide*. In particolare, dati i due attributi $a_t = \langle n_t, d_t \rangle$ e $a_q = \langle n_q, d_q \rangle$ coinvolti in una relazione, verranno eseguiti i seguenti controlli:

   - $\langle n_t$ SYN $n_q \rangle$: la relazione è considerata valida se $d_t$ e $d_q$ sono equivalenti, oppure uno è la specializzzazione dell'altro;
   - $\langle n_t$ BT $n_q \rangle$: la relazione è considerata valida se $d_t$ contiene od è equivalente a $d_q$;
   - $\langle n_t$ NT $n_q \rangle$: la relazione è considerata valida se $d_t$ è contenuto in, od è equivalente a $d_q$;

   Nel caso in cui il dominio dell'attributo contiene il connettore di unione, una relazione viene considerata valida quando almeno un dominio risulta compatibile. Questa fase di validazione è compiuta utilizzando ODB-Tools per il calcolo della relazione di subsumption e equivalence.

   **Esempio 2** Con riferimento al Thesaurus definito nell'esempio 1, riportiamo l'output della fase di validazione: per ciascuna relazione il flag di controllo [1] indica una relazione valida, mentre il flag [0] una invalida.
   ⟨`CD.Patient.physician` BT `ID.Patient.doctor_id`⟩ [0]
   ⟨`CD.Patient.name` BT `ID.Patient.first_name`⟩ [1]
   ⟨`CD.Patient.name` BT `ID.Patient.last_name`⟩ [1]
   ⟨`CD.Nurse.level` SYN `ID.Medical_Staff.position`⟩ [0]
   ⟨`CD.Exam.outcome` SYN `ID.Test.result`⟩ [1]

4. **Inferenza di nuove relazioni.** In questa fase vengono inferite nuove relazioni terminologiche utilizzando lo “schema virtuale” definito nella fase di revisione/integrazione e calcolando le nuove relazioni di subsumption e aggregazione. Queste nuove relazioni semantiche vengono tradotte nelle corrispondenti relazioni terminologiche arricchendo il Thesaurus che ora contiene sia le relazioni esplicite che quelle inferite. Il risultato ottenuto viene chiamato Common Thesaurus.

   **Esempio 3** Rispetto al nostro esempio, le relazioni terminologiche inferite sono:

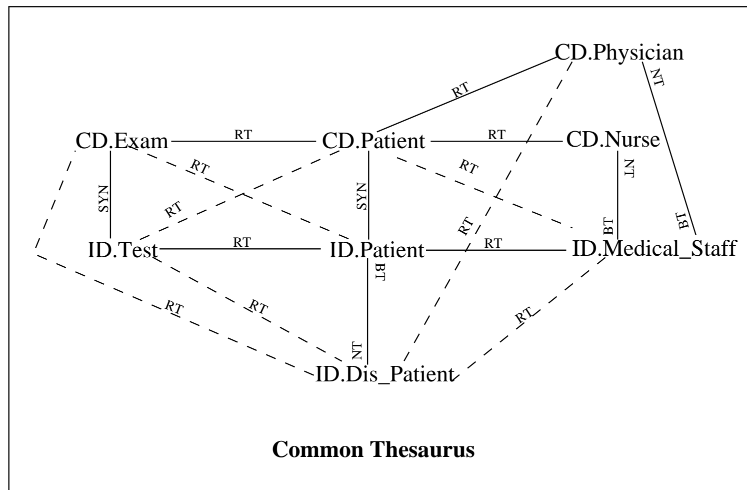



Figura 8.9: Common Thesaurus relativo ai Dipartimenti `Cardiology` e `Intensive Care`

⟨`ID.Patient` RT `CD.Physician`⟩
⟨`ID.Patient` RT `CD.Exam`⟩
⟨`CD.Patient` RT `ID.Medical_Staff`⟩
⟨`CD.Patient` RT `ID.Test`⟩
⟨`ID.Dis_Patient` RT `ID.Medical_Staff`⟩
⟨`ID.Dis_Patient` RT `CD.Exam`⟩
⟨`ID.Dis_Patient` RT `ID.Test`⟩
⟨`ID.Dis_Patient` RT `ID.Medical_Staff`⟩
⟨`ID.Dis_Patient` RT `CD.Physician`⟩

La rappresentazione grafica del Common Thesaurus relativo ai Dipartimenti `Cardiology` and `Intensive Care` è riportata in Fig. 8.9, dove le linee continue rappresentano le relazioni esplicite/estratte, le linee tratteggiate quelle inferite, e le etichette indicano il tipo di relazione terminologica.[2]

[2]Per semplità di rappresentazione sono riportate solamente le relazioni tra nomi di classi.

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

Per realizzare l'integrazione degli schemi ODL$_{I^3}$ delle differenti sorgenti in uno schema globale, abbiamo bisogno di tecniche per l'identificazione delle classi che descrivono le stesse informazioni (o informazioni semanticamente equivalenti), e che sono localizzate all'interno di sorgenti diverse. A questo scopo, le classi ODL$_{I^3}$ sono analizzate e raffrontate attraverso il concetto di *affinità*, che ci permette di determinare il livello di *similarità* tra classi.

Questa attività è compiuta con il supporto dell'ambiente di ARTEMIS. ARTEMIS è stato costruito per l'integrazione semi-automatica di database eterogenei di tipo strutturato [55]. Nell'ambito del progetto MOMIS, le potenzialità di ARTEMIS sono state estese per permettere l'analisi di affinità di descrizioni espresse in linguaggio ODL$_{I^3}$ e gestire i dati semistrutturati.

Le classi ODL$_{I^3}$ sono analizzate e valutate attraverso un *affinity coefficient* che permette di determinare il livello di similarità tra classi contenute in sorgenti diverse. In particolare, per le classi vengono analizzate le relazioni che esistono tra i loro nomi (attraverso il *Name Affinity Coefficient*) e tra i loro attributi (per mezzo dello *Structural affinity Coefficient*), per arrivare ad un valore globale denominato *Global Affinity Coefficient*.

La valutazione dei coefficienti di affinità si basa sulle relazioni terminologiche memorizzate nel Thesaurus. A questo scopo, il Thesaurus viene organizzato in una struttura simile alle Associative Networks [69], dove i nodi (ciascuno dei quali rappresenta genericamente un termine, sia esso il nome di una classe o il nome di un attributo) sono uniti attraverso relazioni terminologiche. A loro volta, tutte le relazioni presenti in questa rete sono percorribili in entrambi i sensi (dunque anche le BT e NT): due termini sono quindi affini se esiste un percorso che li unisce, formato da qualsivoglia relazioni. Per dare una valutazione numerica della affinità tra due termini, a ogni tipo di relazione viene associato un peso (denominato *strength* e denotato da $\sigma_{\Re}$), che sarà tanto maggiore quanto più questo tipo di relazione contribuisce a legare due termini (sarà quindi $\sigma_{syn} \geq \sigma_{bt/nt} \geq \sigma_{rt}$). In questa sezione si userà $\sigma_{ij_{\Re}}$ per denotare il peso della relazione terminologica $\Re$ definita tra i termini $t_i$ e $t_j$. Nel nostro esempio, e nelle sperimentazioni precedentemente realizzate presso l'Università di Milano, si è adottato $\sigma_{syn} = 1$, $\sigma_{bt} = \sigma_{nt} = 0.8$ e $\sigma_{rt} = 0.5$.

**Definizione 13 (Affinity Function)** Presi due termini, $t_i$ e $t_j$, possono essere presenti nel Thesaurus zero o più cammini che li uniscono, formati da relazioni. Ad ognuno di questi cammini corrisponde naturalmente un valore, dato dal prodotto dei pesi delle relazioni in esso coinvolte. La Funzione di Affinità $A_{thes}(t_i, t_j)$ tra due termini, $t_i$ e $t_j$, restituisce il valore maggiore tra

questi, corrispondente al cammino più *stringente*, che unisce questi termini (che non sempre coincide col cammino più breve), definito come segue:

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

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

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

**Esempio 2** Si consideri il Thesaurus illustrato in Figura 8.9. Esiste un cammino tra le classi `CD.Physician` e `ID.Patient`:

$$\texttt{CD.Physician} \rightarrow^{rt} \texttt{CD.Patient} \rightarrow^{syn} \texttt{ID.Patient}$$

Da questo si deduce che $A_{thes}(\texttt{CD.Physician}, \texttt{ID.Patient}) = 0.5 \cdot 1 = 0.5$.

**Definizione 14 (Termini Affini)** Due termini $t_i, t_j$ si dicono *affini*, e si denotano con $t_i \sim t_j$, se la loro Funzione dei Affinità restituisce un valore maggiore o uguale ad un predefinito valore di soglia $\alpha > 0$, cioè:

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

Per esempio, si supponga $\alpha = 0.4$.
In questo caso, dal momento che $A_{thes}(\texttt{CD.Physician}, \texttt{ID.Patient}) = 0.5$, possiamo concludere che $\texttt{CD.Physician} \sim \texttt{ID.Patient}$.

## 8.7.1 Coefficienti di Affinità

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

**Definizione 15 (Name Affinity Coefficient)** Misura la affinità di due classi calcolata rispetto ai loro nomi. Il *Name Affinity Coefficient* di due classi $c$ e $c'$ denotato da $NA(c,c')$, è la misura della affinità tra i loro nomi, $n_c$ e $n_{c'}$, calcolata come segue:

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

**Esempio 3** Si considerino le classi `CD.Physician` e `ID.Patient`. Si avrà che $NA(\texttt{CD.Physician}, \texttt{ID.Patient}) = 0.5$

**Definizione 16 (Structural Affinity coefficient)** Lo Structural Affinity Coefficient di due classi $c$ e $c'$, scritto $SA(c,c')$, è la misura dell'affinità dei loro attributi, calcolata come segue:

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

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

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

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

Lo Structural Affinity Coefficient è valutato utilizzando la funzione di Dice [52], e raffinato da un fattore di controllo $F_c$; essa restituisce un valore compreso nell'intervallo [0,1] proporzionale al numero di attributi *affini* tra le classi considerate.
Rispetto alla definizione originaria di $SA$ precedentemente sviluppata (si veda [53]), è stata proposta in questa tesi l'estensione realizzata dal fattore di controllo $F_c$. Il termine $F_c$ realizza un controllo sui domini degli attributi coinvolti nella relazione da esaminare (il controllo è quello esposto nel terzo passo della Sezione 8.6), permettendo quindi di non limitare la computazione di questo coefficiente alla sola analisi dei nomi che identificano gli attributi (analisi terminologica) e di estenderla alla considerazione dei domini che caratterizzano questi attributi. In pratica, una relazione che coinvolge attributi

viene pesata in modo maggiore o minore nel calcolo del coefficiente a seconda che questa relazione trovi o meno riscontro anche nei tipi dei domini, e non solo nei nomi degli attributi. Il termine $F_c$ va quindi a rifinire il coefficiente $SA$, moltiplicando la prima parte di questo per un termine compreso tra 0 e 1: in particolare, $F_c$ è il rapporto tra numero di relazioni validabili memorizzate nel Thesaurus tra attributi delle due classi, e numero di queste relazioni che sono state validate.
In questo modo, maggiore sarà il numero di attributi affini tra le due classi, e maggiore il numero di controlli positivi, più alto risulterà il valore dello *Structural Affinity Coefficient*.

Per la valutazione della Structural Affinity, gli attributi *optional* delle classi ODL$_{I^3}$ che rappresentano degli object patterns debbono essere analizzati in modo specifico ed appropriato. A seconda della tipologia di attributi considerati, sono possibili diverse opzioni per il calcolo del coefficiente $SA()$:

1. *All attribute-based.* Con questa opzione, gli attributi *optional* sono trattati alla stregua degli altri e sono sempre presi in considerazione nella valutazione dell'affinità tra classi ODL$_{I^3}$ che descrivono object patterns.

2. *Common attribute-based.* Con questa opzione, gli attributi *optional* non vengono considerati nella valutazione dell'affinità.

3. *Threshold-based.* Con questa opzione, gli attributi *optional* vengono considerati nella valutazione dell'affinità solamente nel caso in cui sono comuni ad una percentuale prefissata di oggetti contenuti negli object pattern.

La terza opzione è la più complessa da applicare, in quanto richiede di fissare un valore di soglia che molto spesso dipende dal singolo object pattern considerato o dalla sorgente. Ciascuna opzione modifica il valore di affinità risultante: a parità di coppie di attributi affini, la seconda opzione fornisce un valore di affinità maggiore rispetto alla prima opzione (infatti, scegliendo l'opzione 2, nel denominatore della formula del $SA()$ sono considerati un numero inferiore di attributi). D'altra parte, se la maggior parte degli attributi di una classe sono opzionali allora la scelta dell'opzione 1 risulta migliore per il calcolo dell'affinità. In definitiva, la scelta tra le prime due opzioni è legata al singolo dominio analizzato: nel nostro esempio applicheremo entrambe le opzioni discutendo i valori ottenuti.

**Esempio 4** Consideriamo le classi `CD.Physician` e `ID.Patient`. Applicando l'opzione *all attribute-based*, abbiamo che

$$SA(\texttt{CD.Physician}, \texttt{ID.Patient}) = \frac{2 \cdot 2}{4+6} = 0.4$$

dovuti alle seguenti affinità: `CD.name`$\sim${`ID.first_name`,`ID.last_name`} e `CD.address`$\sim$`ID.address`.

**Definizione 17 (Global Affinity Coefficient)** Il Global Affinity Coefficient di due classi $c$ e $c'$, denotato da $GA(c,c')$, è la misura della loro affinità calcolata come la somma pesata degli *Name Affinity Coefficient* e *Structural Affinity Coefficient*:

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

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

**Esempio 5** Considerando i Name e Structural Affinity coefficients dell'Esempio 3, ed Esempio 4, rispettivavemte, il Global Affinity coefficient tra le classi `CD.Physician` e `ID.Patient` è calcolato nel seguente modo:

$$GA(\texttt{CD.Physician}, \texttt{ID.Patient}) = 0.5 \cdot 0.5 + 0.5 \cdot 0.4 = 0.45$$

### 8.7.2 Considerazioni sul processo di affinità

La correttezza del processo di affinità dipende sia dall'affidabilità delle relazioni terminologiche, sia dai parametri (i.e., strengths, weights, thresholds) che intervengono nel calcolo dei coefficienti. Come si può facilmente intuire, il processo di affinità contiene intrinsecamente una parte di soggettività, dovuta al fatto che la conoscenza specifica e l'esperienza del progettista giocano

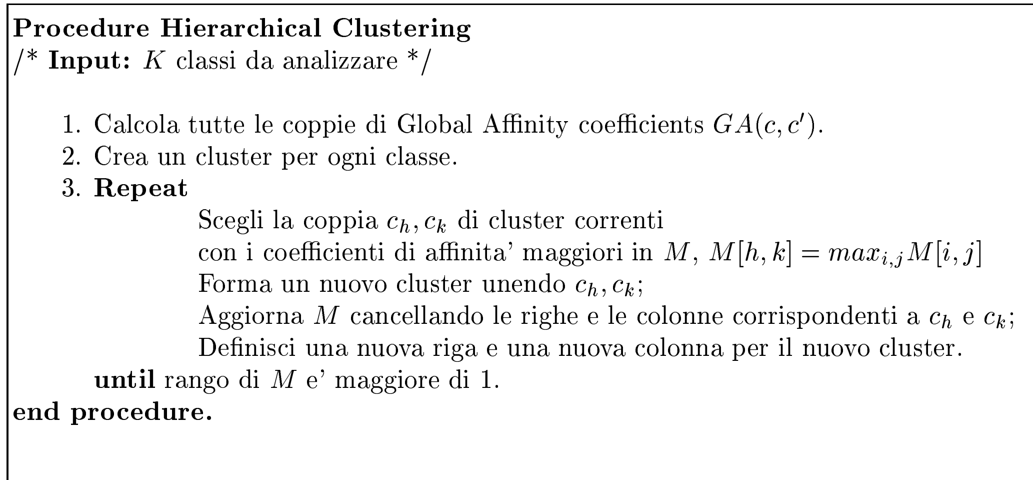
un ruolo fondamentale nella costruzione del Common Thesaurus e nella scelta dei parametri. Per rendere la fase di affinità più obiettiva, in MOMIS abbiamo introdotto funzionalità di tipo interattivo. In particolare, la costruzione e la validazione del Common Thesaurus in ODB-Tools è un processo interattivo, ed il progettista può aggiungere terminological relationships addizionali tipiche del dominio applicativo in esame. ODB-Tools valida tutte le relazioni contenute nel Thesaurus e ne inferisce di nuove, mantenendo il Common Thesaurus consistente e corretto. Il processo di valutazione dell'affinità in ARTEMIS è pure interattivo e basato sui pesi (modificabili), per permettere al designer di modificare in modo appropriato i parametri e validare le scelte compiute durante tutti gli step del processo. L'approccio di affinità utilizzato in ARTEMIS è stato sperimentato su insiemi diversi di schemi concettuali di database per selezionare i valori di default più appropriati, che sono quelli risulati più soddisfacenti nella maggior parte dei casi.

Per una discussione più approfondita delle sperimentazioni compiute su schemi forniti dalla Pubblica Amministrazione Italiana, si può fare riferimento a [53, 55]. Considerazioni di tipo più generali sull'utilizzo e sulla definizione delle tecniche semi-automatiche basate sui pesi per l'analisi di schemi concettuali si possono trovare in [54].

## 8.8 Generazione dei Cluster di classi ODL$_{I^3}$

Per l'identificazione degli insiemi di classi ODL$_{I^3}$ aventi elevata affinità negli schemi considerati sono utilizzate tecniche di clustering, attraverso le quali le classi sono automaticamente classificate in gruppi caratterizzati da differenti livelli di affinità, formando un albero [67].

Questa procedura di clustering (vedi Figura 8.10) utilizza una matrice $M$ di rango $k$, con $k$ uguale al numero totale di classi ODL$_{I^3}$ che devono essere analizzate. In ogni casella $M[h,k]$ della matrice è rappresentato il coefficiente globale $GA()$ riferito alle classi $c_h$ e $c_k$. La procedura di clustering è iterativa e comincia allocando per ogni classe un singolo cluster: successivamente, ad ogni iterazione, i due cluster tra i quali sussiste il $GA()$ di valore massimo nella matrice $M$ sono uniti. $M$ è così aggiornata dopo ogni operazione di fusione tra cluster, cancellando le righe e le colonne corrispondenti ai cluster unificati, e inserendo una nuova riga ed una nuova colonna che rappresenti il nuovo cluster determinato. Vengono quindi calcolati i coefficienti $GA()$ tra questo cluster aggiunto e tutti quelli già presenti nella matrice: in particolare, viene mantenuto il valore $GA()$ massimo tra i due che erano stati già calcolati tra i cluster rimossi ed il corrispondente cluster col quale si vuole determinare



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

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

   **until** rango di $M$ e' maggiore di 1.

**end procedure.**

Figura 8.10: Procedura di clustering

il nuovo valore del coefficiente globale. La procedura termina quando tutte le classi appartengono ad un unico cluster, ossia è definito completamente l'albero di affinitá.

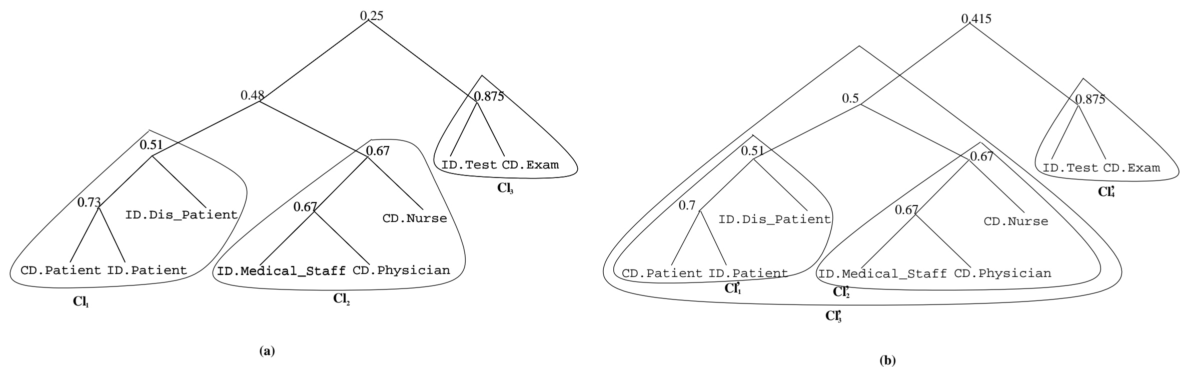



Figura 8.11: Affinity trees for the `Cardiology` and `Intensive Care` data sources

La Fig. 8.11 mostra l'albero di affinitá risultante dalla clusterizzazione del nostro insieme di classi $\text{ODL}_{I^3}$. In particolare, la Fig. 8.11(a) illustra l'albero di affinitá ottenuto utilizzando l'opzione "all attribute-based" nel calcolo del coefficiente della Structural Affinity, mentre la Fig. 8.11(b) illustra l'albero ottenuto applicando l'opzione "common attribute-based".

Dopo aver costruito l'albero di affinitá, il problema è quello di selezionare i cluster più appropriati per la definizione delle classi nello schema globale. La procedura di selezione dei cluster, in ARTEMIS, viene mantenuta interattiva

attraverso la modifica del valore di soglia. Il progettista specifica il valore della soglia $T$ ed i cluster caratterizzati da un valore di $GA()$ superiore o uguale a $T$ sono selezionati e proposti. Per alti valori di $T$ si ottengono cluster piccoli e molto omogenei tra loro. Diminuendo il valore di $T$, i cluster ottenuti contengono più classi e sono più eterogenei. Nel tool il valore di default di $T$ viene posto pari a 0.5. Tale valore può essere modificato dinamicamente, sulla base della specifica applicazione in esame.

Con riferimento alla Fig. 8.11, i cluster sono selezionati utilizzando i valori di default. Come si può vedere dall'albero di affinità, il numero di attributi considerati durante il processo di calcolo della Structural Affinity modifica il numero di cluster ottenuti. Nel primo caso (Fig. 8.11(a)), l'albero dei cluster contiene: $\mathbf{Cl_1}$, che rappresenta le informazioni riguardanti i pazienti; $\mathbf{Cl_2}$ che rappresenta le informazioni riguardanti lo staff medico e $\mathbf{Cl_3}$ che rappresenta le informazioni riguardanti gli esami clinici. Nel secondo caso (Fig. 8.11(b)), i cluster $\mathbf{Cl'_1}$ e $\mathbf{Cl'_2}$ sono ancora presenti, (e rappresentano ancora informazioni riguardanti i pazienti e lo staff medico rispettivamente). In più, un ulteriore cluster è stato formato e selezionato nell'albero di affinità, chiamato $\mathbf{Cl'_3}$, che unisce il cluster $\mathbf{Cl'_1}$ che il $\mathbf{Cl'_2}$. Questo accade in quanto il pattern `Patient` include alcuni attributi optional. Per questa ragione, applicando l'opzione 2., la Structural Affinity tra la classe $\mathrm{ODL}_{I_3}$ che rappresenta il pattern `Patient` e le altre classi $\mathrm{ODL}_{I_3}$ risulta più alta rispetto alla Structural Affinity ottenuta utilizzando l'opzione 1. (essendo il numero di coppie di attributi con affinità uguale nei due casi).

Pensiamo che entrambe queste opzioni siano valide in contesti applicativi diversi: per mantenere la flessibilità e l'approccio general-purpose, ARTEMIS permette di calcolare l'albero di affinità utilizzando entrambe le opzioni (1. e 2.) e lasciando al designer la scelta più opportuna.

## 8.9 Costruzione dello Schema Globale di mediatore

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

La prima fase di questo processo viene realizzata automaticamente e genera, per ogni cluster, una $global_class_i$ rappresentativa di tutte le classi che fanno parte del cluster (ovvero una classe che costituisca una visione unificata

di queste classi). Sia $Cl_i$ un cluster determinato nella fase precedente: ad esso viene associata la $global_class_i$, alle quale corrisponderà quindi un insieme di attributi globali. La fase di determinazione degli attributi è realizzabile in modo automatico, basandosi sulle relazioni tra attributi memorizzate nel Thesaurus (vedi Sezione 8.6) e seguendo i seguenti criteri:

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

Per esempio, per il cluster $\texttt{Cl}_1$ di Fig. 8.11 si ottiene il seguente insieme di attributi globali:

$$A = (\texttt{name, code, address, exam*, room, bed, therapy*, date,}\\ \texttt{note, physician*})$$

Per meglio completare la definizione delle classi $\text{ODL}_{I^3}$ globali occorre specificare ulteriori informazioni: in particolare MOMIS richiede al designer di fornire, in modo interattivo, informazioni riguardanti:

1. $global_class_i$ name
   il tool propone un insieme di nomi candidati per la classe globale, utilizzando le terminological relationships definite nel Common Thesaurus per le classi contenute in $\texttt{Cl}_\texttt{i}$. Basandosi su questi candidati, il designer può decidere il nome più appropriato per $global_class_i$. Nel caso in cui i nomi candidati non sono significativi, è comunque possibile assegnare nomi diversi. Nel nostro esempio possiamo deciderediassegnare il nome di `Hospital_Patient` alla classe globale definita in corrispondenza del cluster $\texttt{Cl}_1$.

   Perciò, rispetto a $\texttt{Cl}_1$ otteniamo la classe globale:

   $$\text{Hospital_Patient} = (\texttt{Hospital_Patient,\{name,code,address,}\\ \texttt{exam*,room,bed,therapy*,date,}\\ \texttt{note,physician*\}})$$

2. *mappings* tra gli attributi globali di *global_class*$_i$ ed i corrispondenti attributi delle classi in $\texttt{Cl}_\texttt{i}$

   In particulare, se un attributo globale è ottenuto da più di un attributo di una stessa classe in $\texttt{Cl}_\texttt{i}$, il designer deve specificare il tipo di *correspondence* che deve essere definita nell'attributo globale. In MOMIS vengono fornite le seguenti opzioni:

   - *and* correspondence

     specifica che un attributo globale corrisponde alla concatenazione degli attributi della classe $c_h \in \texttt{Cl}_\texttt{i}$.
     Per esempio, l'attributo globale `name` di `Hospital_Patient` corrisponde alla concatenazione degli attributi `first_name` e `last_name` della classe `ID.Patient` in $\texttt{Cl}_\texttt{1}$. Specificando l'*and* correspondence per l'attributo `name`, il progettista afferma che entrambi i valori degli attributi `first_name` e `last_name` debbono essere considerati quando è coinvolta la classe `ID.Patient`.

   - *union* correspondence

     specifica che l'attributo globale corrisponde ad almeno uno degli attributi specificati della classe $c_h \in \texttt{Cl}_\texttt{i}$;

3. *default values* che vengono assegnati agli attributi globali in corrispondenza della classe $c_h \in \texttt{Cl}_\texttt{i}$. Un valore di default di un attributo globale è sempre verificato quando viene valutato nella classe $c_h$.

4. *new attributes* per la classe globale.

   Per esempio, un nuovo attributo `dept` può essere aggiunto nella classe globale `Hospital_Patient`. Questo attributo è utilizzato per mantenere informazioni legate al dipartimento in cui un paziente viene ricoverato.

Un esempio di definizione di un classe globale in linguaggio ODL$_{I^3}$ è mostrato in Fig. 8.12 per la classe `Hospital_Patient`. Come si può vedere da questa figura, per ciascun attributo sono definite regole di mapping, in modo da specificare sia informazioni sulla corrispondenza tra attributo globale ed il corrispondente nelle classi locali associate, sia su eventuali valori di default o di null (mancanza della corrispondenza). Per esempio, per l'attributo globale `name`, la regola di mapping specifica gli attributi che debbono essere considerati in ciascuna classe locale del cluster $\texttt{Cl}_\texttt{1}$. Nel caso specifico, viene definita una *and* correspondence per la classe `ID.Patient`. Una diversa mapping rule è definita per l'attributo globale `dept` per specificare il valore

da essere associato con `dept` per le istanze di `CD.Patient`, `ID.Patient`, e `ID.Dis_Patient`.

Come scritto precedentemente, la *union* correspondance risulta utile quando un attributo globale corrisponde a due o più attributi di una classe di una sorgente, in base al valore di un attributo terzo chiamato *tag attribute*. Per esempio, supponendo di avere definito un cluster globale per la classe `automobile` e che in una classe di una sorgente viene memorizzato il prezzo delle auto nelle due valute di Lire Italiane e US Dollari e si abbia `country` come attributo tag. Il progettista può definire una *union* correspondance tra gli attributi `Italian_price` e `US_price` introducendo una rule che specifichi il mapping locale basato sul valore dell'attributo `country`. Utilizzando la sintassi delle mapping rule, la dichiarazione è la seguente:

```
...
 attribute integer price
    mapping rule (S.car.Italian_price union
                  S.car.US_price on Rule1),
                 ...
...
 rule Rule1 { case of S.car.country:
             ``Italy'' : S.car.Italian_price;
             ``US''    : S.car.US_price; }
```

Come si può vedere, nella definizione di un attributo globale non viene specificato alcun dominio: questo significa che MOMIS accetta i domini definiti nelle sorgenti locali e, in risposta ad una query, sono visualizzati i dati nei formati specifici delle sorgenti locali.

Tra gli sviluppi futuri del sistema, seguendo quanto suggerito in [71, 64], un nostro obiettivo è quello di sviluppare una libreria aperta di funzioni che contiene le principali operazioni di mapping e che il progettista può eseguire, può personalizzare ed estendere secondo le proprie necessità.

La descrizione in linguaggio $ODL_{I^3}$ della classe globale *global_class*$_i$ viene tradotta in una mapping table persistente. Come esempio, la mapping table per la classe `Hospital_Patient` è mostrata in Fig. 8.13.

## 8.10 Query Processing e Optimization

In questa sezione verrà descritto il processo che, sfruttando le informazioni memorizzate nelle mapping table, realizza la Query Reformulation. Quando un utente generico invia una query al sistema MOMIS, il Query Manager produce un insieme di sottoquery che dovranno essere spedite alle diverse

```
interface Hospital_Patient
{ attribute name
    mapping_rule (ID.Patient.first_name and
                 ID.Patient.last_name),
                 CD.Patient.name;
     ...
  attribute physician
    mapping_rule CD.Patient.physician,
                 Id.Patient.doctor_id
  attribute dept
    mapping_rule CD.Patient = 'Cardiology',
                 ID.Patient = 'Intensive Care',
                 ID.Dis_Patient = 'Intensive Care'
}
```

Figura 8.12: Esempio di classi globali in $ODL_{I^3}$

| Hospital_Patient | code | name | exam | ... | physician | dept |
|---|---|---|---|---|---|---|
| CD.Patient | null | name | exam | ... | physician | 'Cardiology' |
| ID.Patient | code | first_name *and* last_name | test | ... | doctor_id | 'Intensive Care' |
| ID.Dis_Patient | code | null | null | ... | ID.Patient. doctor_id | 'Intensive Care' |

Figura 8.13: `Hospital_Patient` mapping table

fonti di informazione per dare risposta alla query originale.

Seguendo l'impostazione simile ad altri approcci *semantici* [7, 8], questo processo consiste di due fasi distinte:

- semantic optimization
- query plan formulation

### 8.10.1 Semantic Optimization

In questa fase il Query Manager processa la query data utilizzando le tecniche di ottimizzazione semantica supportate da ODB-Tools [20, 21, 18, 2] allo scopo di ridurre il costo del piano di accesso della query. La query viene sostituita da una nuova che incorpora ogni possibile restrizione che non è presente nella query originale ma è logicamente implicata dalla query basata sullo schema globale. La trasformazione è basata sulle inferenze logiche ba-

sate sulla conoscenza del contesto (in particolare sugli integrity constraint) dello schema globale di mediatore.

Consideriamo, ad esempio, la query: "Seleziona i nomi dei pazienti il cui esame ha come risultato 'Heart risk' ".

```
Q1:
select name
from   Hospital_Patient
where  exam.result = 'Heart risk'
```

Supponiamo ora che nel dominio `Hospital` esista una relazione tra il risultato di un esame e il dipartimento in cui il paziente viene ricoverato; ad esempio che tutti i pazienti il cui esame ha il valore 'Heart risk' sono ricoverati nel dipartimento 'Cardiology'. In base a questo requisito il progettista può definire il seguente vincolo di integrità relativo alla classe `Hospital_Patient` dello schema globale:

```
rule R1 forall X in Hospital_Patient:  (X.exam.result = 'Heart
risk') then X.dept = ''Cardiology';
```

Il Query Manager, utilizzando il query optimizer di ODB-Tools, esegue l'espansione semantica della query applicando la rule `R1`. La query risultante è la seguente:

```
Q1:
select name
from   Hospital_Patient
where  exam.result = 'Heart risk'
and    dept = 'Cardiology'
```

L'espansione semantica è compiuta al fine di aggiungere più fattori booleani possibile nella "where clause": questo processo rende il query plan formulation più costoso (poichè deve essere tradotta una query più pesante per ciascuna sorgente di interesse), ma il recupero dei dati sulla singola sorgente può essere enormemente ridotto se sono presenti indici sui predicati aggiunti in fase di ottimizzazione.

### 8.10.2 Query Plan Formulation

Dopo aver prodotto la query ottimizzata, il Query Manager genera un insieme di sottoquery destinate alle sorgenti locali. Per ciascuna sorgente informativa coinvolta, il Query Manager deve tradurre la query ottimizzata in termini dei corrispondenti schemi locali, utilizzando la mapping table

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

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

Figura 8.14: Algoritmo di Controllo ed Eliminazione

associata a ciascuna classe globale coinvolta nella query. Allo scopo di ottenere ciascuna sottoquery, il Query Manager controlla e traduce ciascun fattore booleano nella clausola `where`. In particolare, l'algoritmo di *Check* (mostrato in Fig. 8.14) è stato pensato per l'eliminazione dalla lista di classi da interrogare quelle nelle quali si è già sicuri, basandosi sulle informazioni memorizzate nella mapping table, di non trovare dati *utili* o comunque *verificabili*. Per eliminare le fonti che fornirebbero dati non *utili*, ci si basa sui valori di default della mapping table: se il valore definito di default per un attributo in una classe locale è diverso dal valore specificato (in **and**) nella clausola `where`, la classe non sarà interrogata.
Per eliminare invece le sorgenti che fornirebbero dati non *verificabili* ci si basa sui valori nulli presenti nella mapping table: in particolare, se è stata specificata una condizione nella clausola `where` (sempre in **and** con le altre condizioni espresse nella clausola) su un attributo globale per il quale non esiste un attributo locale corrispettivo nella classe che si sta analizzando, si è scelto di non procedere all'interrogazione della classe, non potendo essere verificata quella condizione. È stata quindi fatta una scelta drastica, per mantenere la sicurezza di ricevere solamente dati completamente verificati, ma sarebbero comunque possibili scelte differenti: tra queste, si potrebbe ad esempio ipotizzare l'esistenza di un livello di *credibilità* delle risposte, che l'utente dovrebbe scegliere, che esprima la percentuale delle condizioni che si vogliono siano assolutamente verificate dai dati ricevuti in risposta all'interrogazione.

Con riferimento al nostro esempio, lo step 1 dell'algoritmo di *Check* esclude la classe `Dis_Patient` e lo step 2 la classe `Patient` della sorgente `ID`.

Conseguentemente, solo la seguente sottoquery per la sorgente CD viene inviata:

```
select R.name
from   Patient R
where  exists X in R.exam : X.outcome = 'Heart risk'
```

In tal modo, una effettiva ottimizzazione viene compiuta poichè solamente una sorgente locale deve essere acceduta, anzichè due.
Se gli integrity constraint sono forniti anche per le sorgenti locali, è possibile iterare il procedimento eseguendo l'ottimizzazione semantica della sottoquery prima di inviarla alla sorgente locale. Per esempio, se la seguente rule viene definita per la sorgente CD:

```
rule R2 forall X in Patient:  (exists X.exam.outcome = 'Heart
risk') then X.room > 20 ;
```

R2 può essere utilizzata per ottimizzare la sottoquery, dando origine alla seguente query:

```
select R.name
from   Patient R
where  exists X in R.exam : X.outcome = 'Heart risk'
and    R.room > 20
```

Questa trasformazione può essere utile nel caso in cui sia disponibile un indice per l'attributo `room`.

Come altro esempio della query formulation sullo schema globale, consideriamo il caso di rappresentazione eterogenea dell'attributo address e supponiamo di ricercare il nome di tutti i pazienti con un determinato indirizzo. La query risultante è la Q2:

```
Q2: select name
from Hospital_Patient
where address.street = 'Army Street'
```

Il processo di reformulation della query Q2 porta alla generazione di due sottoquery per la sorgente CD (in quanto occorre tener conto del costruttore di union sulll'attributo address) e una sottoquery per il dipartimento `Intensive Care`:

```
CD:
select R.name
from   Patient R
```

```
where  R.address = 'Army Street'

CD:
select R.name
from   Patient R
where  R.address.street = 'Army Street'

ID:
select P.first_name, P.last_name
from   Patient P
where  P.address = 'Army Street'
```

## 8.11 Confronto con altri lavori

Lavori correlati al progetto MOMIS sono identificati nell'area dei semistructured data e dell'integrazione di informazioni eterogenee.

**Semistructured data.** La problematica della modellizzazione dei semistructured data è stata particolarmente investigata in letteratura. In particolare, una survey inerente ai problemi del semistructured data modeling querying è stata presentata in [42]. Due modelli simili per semistructured data sono stati proposti in [44, 108], e sono basati un grafo etichettato a radice, in cui i nodi sono gli oggetti (i dati) e gli archi le etichette. Nel modello presentato in [44], l'informazione risiede solamente nelle etichette, mentre nel modello "Object Exchange Model" (OEM) propopsto da Papakonstantinou et. al. in [108], l'informazione risiede anche nei nodi.

Nel primo modello, gli archi sono etichettati sia tramite i tipi base piú comuni quali integer, real, string, sia tramite nomi (chiamati *symbols*) che corrispondono ai nomi degli attributi delle relazioni o delle classi. Nel secondo modello abbiamo: *i)* i nodi foglia sono etichettati dai dati; *ii)* i nodi interni non sono etichettati con dati significativi; *iii)* gli archi sono etichettati solo da symbols (nomi di attributi). È possibile definire funzioni di mapping tra i due modelli: introducendo nuovi archi, un grafo OEM può essere convertito nel corrispondente modello proposto in [44].

L'altra problematica che è stata fortemente studiata riguarda il tentativo di aggiungere struttura a dati semistrutturati: questa prospettiva è vicina alla nostra proposta di object pattern. In particolare, in [73] viene proposta la nozione di *dataguide* come una "loose description of the structure of the data" memorizzati nella sorgente informativa. Una proposta per inferire struttura in dati semistrutturati è stata presentata in [87], dove gli autori utilizzano un modello dei dati basato su grafi derivato da [44, 108]. In questo

modelli, la struttura dei dati è una gerarchia di tipi che viene ottenuta tramite un particolare algoritmo che misura la distanza (*jump*) tra i tipi, basandosi sull'importanza relativa di alcuni attributi rispetto agli altri in un insieme elevato di dati. In questo modo vine costruita una "reasonably small approximation" dei tipi che rappresentano la vasta e irregolare casistica dei dati presenti nella sorgente, riuscendo a distinguere gli attributi principali ("core" attribute) dagli altri meno significativi. In [43] viene proposta una nuova nozione di schema a grafo appropriata per rappresentazioni database tramite grafi etichettati con radice. Il principale utilizzo della struttura estratta da una sorgente semistrutturata è stato presentato per la query optimization. Infatti, l'esistenza di un cammino nella struttura semplifica la valutazione della query limitando la ricerca sui soli dati rilevanti. La nostra attenzione verso i dati semistrutturati è stata rivolta verso l'utilizzo di una struttura nella forma di object patterns per la rappresentazione di dati semistrutturati per utilizzarla come supporto all'integrazione di dati.

**Heterogeneous information integration.** In quest'area sono stati sviluppati molti progetti basati sull'architettura a mediatore [7, 8, 38, 60, 62, 85, 94].
MOMIS è basato su di un'architettura a mediatore e segue il 'semantic approach'.

Seguento la classificazione dei sistemi di integrazione proposti da Hull [75], MOMIS si pone nel gruppo di quelli che seguono il "virtual approach". Virtual approach è stato proposto all'inizio degli anni '80 nei modelli multidatabase [83, 66]. Più recentemente, alcuni sistemi sono stati sviluppati basandosi sulla description logics [7, 82], tra i quali CLASSIC [35]. Tutti gli approcci virtuali sono basati sul modello di query decomposition, inviando sottoquery ai database sorgenti e unificando le risposte sui dati selezionati. Sistemi più recenti basati sulla description logic sono focalizzati fondamentalmente sulle query congiuntive (i.e. quelle esprimibili utilizzando la selezione, proiezione e join), beneficiando della *Open World Assumption.* Riguardo allo schema, viene utilizzato un approccio "top-down": viene creato uno schema globale che comprende tutte le informazioni rilevanti delle singole sorgenti, ed i dati delle sorgenti sono espressi come viste dello schema globale [101].

Riferendoci sempre alla classificazione proposta da Hull, MOMIS può essere posizionato nella categoria delle "read-only views", e cioè di quei sistemi il cui compito è quello di supportare una vista integrata dei dati presenti in database diversi con accessi di tipo read-only. I progetti più simili al nostro sono: GARLIC, SIMS, Information Manifold e Infomaster.

Il progetto GARLIC [85, 94] fornisce un'architettura con wrapper complessi che forniscono la descrizione delle sorgenti in linguaggio OO (chiamato

GDL) e poi viene definito manualmente uno schema globale che unifica la visione delle sorgenti locali tramite gli oggetti Garlic Complex Objects.
Il progetto SIMS [7, 8] propone di creare la definizione dello schema globale utilizzando la description logic (i.e. il linguaggio LOOM) per descrivere le sorgenti informative. L'uso dello schema globale permette ad entrambi i progetti GARLIC e SIMS di supportare query di qualsivolgia natura basate sullo schema invece di quelle predefinite dal sistema.

Information Manifold Systems [79, 82], alla stregua del progetto MOMIS, fornisce un mediatore e una query manager independente dalla sorgente. Lo schema di input di Information Manifold System è dato da un'insieme di descrizione delle sorgenti; così, data una query, il sistema è in grado di creare un query plan in grado di rispondere alla query utilizzando le sorgenti locali. Gli algoritmi che descrivono le tecniche di decisione per la scelta delle informazioni utili e per generare il query plan sono forniti in [82, 81]. Lo schema globale di input è completamente modellato dall'utente, a differenza del nostro approccio in cui l'integrazione viene guidata dal sistema.

Infomaster System [71, 64] fornisce l'accesso integrato a sorgenti informative eterogenee e distribuite, mostrando all'utente il sistema come fosse centralizzato ed omogeneo. Il sistema è basato su di uno schema globale, modellato completamente dall'utente e dal modulo di query processing che determina dinamicamente ed efficientemente il piano di accesso utilizzando regole di traduzione che armonizzano le sorgenti eterogenee.

D'altra parte, molti progetti sono basati sull'approccio 'structural' [38, 60, 62]. Il progetto TSIMMIS [60, 70] segue l'approccio 'structural' ed utilizza un modello self-describing (OEM) per rappresentare gli oggetti ed una tecnica di pattern matching per eseguire un insieme predefinito di query basate su query template. La conoscenza semantica viene codificata nelle regole di mediazione scritte in linguaggio MSL (Mediator Specification Language) che permettono l'integrazione delle sorgenti a livello di mediatore. Sebbene la generalità e l'immediatezza di OEM e MSL rendono questo approccio un buon candidato per l'integrazione di sorgenti informative eterogenee e semistrutturate di notevoli dimensioni, si ha un aumento notevole dei costi di sviluppo nel caso di inserimento dinamico di altre sorgenti nel sistema. Infatti, in TSIMMIS, le nuove sorgenti non solo sono collegate allo specifico wrapper (che quindi deve essere specificato e/o sviluppato) ma il mediatore deve ridefinire le regole MSL e ricompilarle; l'amministratore del sistema deve infine specificare come il sistema utilizza la nuova sorgente [101]. In MOMIS questa attività viene assistita dal sistema.

Un'altra area di ricerca è focalizzata sulla identificazione e risoluzione delle inconsistenze semantiche tra gli schemi, basandosi sulla definizione di corrispondenza tra elementi di schemi. In [39] viene presentato il modello

Summary Schema Model (SSM) per fornire il supporto all'idenficazione delle similarità semantiche. Utilizzando le tassonomie presenti, SSM viene usato come descrizione di alto livello e coincisa di tutti i dati disponibili nella sorgente. Due entità sono semanticamente correlate se sono mappate nello stesso concetto della tassonomia.

In [68] viene definita una tecnica basata sulla fuzzy logic per derivare le terminological relationships tra gli elementi dello schema.

Altri approcci considerano gli attributi ed i domini degli elementi dello schema [33, 61]. In [33], relazioni, attributi, ed integrity constraints presenti nello schema sorgente sono espressi utilizzando un linguaggio di tipo logico per unificare la conoscenza di tutti gli elementi nella sorgente.

Infine, alcuni gruppi di ricerca hanno proposto di considerare le architetture federate per la risoluzione delle eterogeneità semantiche, utilizzando la definizione di schema federato data in [92, 97].

# Capitolo 9

# Conclusioni

La presente tesi si articola su due temi fondamentali. Il primo è relativo all'estensione delle Logiche Descrittive introdotte in ambito Intelligenza Artificiale per modellare basi di dati ad oggetti e sfruttare le tecniche di inferenza di tali logiche al fine di risolvere problemi fondamentali quali: il controllo di consistenza di schemi e l'ottimizzazione di interrogazioni.

Il secondo tema è relativo all'integrazione di informazioni testuali eterogenee e distribuite. Nell'affrontare questo tema, si è potuto verificare l'efficacia di logiche descrittive estese, quali quella introdotta nella prima parte per risolvere il problema della integrazione "intelligente" e quindi effettiva.

Entrambi i temi sono stati affrontati sia dal punto di vista teorico che realizzativo. Hanno portato infatti allo sviluppo di due strumenti software: ODB-Tools (per il primo tema) e MOMIS (per il secondo).

La presente tesi presenta un sistema a mediatore, MOMIS, il cui compito è fornire accesso integrato ad una molteplicità di sorgenti eterogenee di informazioni. Il progetto MOMIS si colloca all'interno di una vasta e recente area di ricerca, l'Integrazione Intelligente di Informazioni, che si pone come obiettivo l'utilizzo di tecniche di Intelligenza Artificiale nell'ambito dell'integrazione dei dati. Sono quindi state presentate tecniche di Intelligenza Artificiale sviluppate nell'area della Logica Descrittiva per guidare il progettista nella definizione dello schema di mediatore integrato. In particolare, la Logica Descrittiva OLCD è stata estesa per migliorare i servizi resi al problema dell'integrazione.

Confrontando il lavoro con le soluzioni già presenti in letteratura, si può affermare di aver raggiunto un risultato innovativo: i sistemi precedentemente sviluppati realizzano una buona (a volte eccellente) fase di "Query Processing" (ovvero di gestione delle interrogazioni), ma sono per la maggior parte dei casi rinunciatari nella fase di integrazione, lasciata alla responsabilità del progettista del sistema. Sotto questo punto di vista, si è allora sicuramente

raggiunto lo scopo che ci si era prefissi: con l'aiuto dell'operatore, attraverso un processo semi-automatico, MOMIS giunge non solo alla unificazione degli schemi delle sorgenti, ma pure alla creazione di uno schema globale vero e proprio, direttamente interrogabile dall'utente. Per arrivare a questo risultato, é stato sviluppato un sistema che, sfruttando le informazioni fornite dal progettista attraverso un linguaggio dichiarativo ($ODL_{I^3}$), giunge alla creazione di un *mediatore di informazioni* in grado di gestire la maggior parte delle problematiche che si possono incontrare in questo campo.

Importante è anche, a nostro avviso, la componente *intelligente* dell'intero processo: sebbene la presenza del progettista rimanga essenziale, le tecniche di Intelligenza Artificiale che erano a disposizione sono risultate essere una componente fondamentale per l'intero sistema, usate ampiamente e nelle diverse fasi dell'integrazione.

Da sottolineare inoltre è l'utilizzo delle tecniche di Inferenza proposto nella fase di "Query Reformulation", dove attraverso un algoritmo di ottimizzazione semantica si determina un effettivo risparmio sui costi di realizzazione delle interrogazioni.

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

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

# Appendice A

# Glossario $I^3$

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

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

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

## A.1 Architettura

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

- configurazione = istanza particolare di una architettura per una applicazione o un cliente.

- collante (glue) = software o regole che servono per per collegare i componenti o per interoperare attraverso i domini.

- strato = grossolana categorizzazione dei componenti e degli strumenti in una configurazione. L'architettura $I^3$ distingue tre strati, ognuno dei quali fornisce una diversa categoria di servizi:

    1. Servizi di Coordinamento = coprono le fasi di scoperta delle risorse, distribuzione delle risorse, invocazione, scheduling ...
    2. Servizi di Mediazione = coprono la fase di query processing e di trattamento dei risultati, nonché il filtraggio dei dati, la generazione di nuove informazioni, etc.
    3. Servizi di Wrapping = servono per l'utilizzo dei wrappers e degli altri strumenti simili utilizzati per adattarsi a standards di accesso ai dati e alle convenzioni adoperate per la mediazione e per il coordinamento.

- agente = strumento che realizza un servizio, sia per il suo proprietario, sia per un cliente del suo proprietario.

- facilitatore = componente che fornisce i servizi di coordinamento, come pure l'instradamento delle interrogazioni del cliente.

- mediatore = componente che fornisce i servizi di mediazione e che provvede a dare valore aggiunto alle informazioni che sono trasmesse al cliente in risposta ad una interrogazione.

- cliente (customer) = proprietario dell'applicazione che gestisce le interrogazioni, o utente finale, che usufruisce dei servizi.

- risorsa = base di dati accessibile, server ad oggetti, base di conoscenze ...

- contenuto = risultato informativo ricavato da una sorgente.

- servizio = funzione fornita da uno strumento in un componente e diretta ad un cliente, direttamente od indirettamente.

- strumento (tool) = programma software che realizza un servizio, tipicamente indipendentemente dal dominio.

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

## A.2 Servizi

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

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

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

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

## A.3 Risorse

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

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

- sistema di amministrazione delle regole = software indipendente dal dominio che raccoglie, seleziona ed agisce sulle regole.
- database attivo = database in grado di reagire a determinati eventi.
- dato virtuale = dato rappresentato attraverso referenze e procedure.
- stato = istanza o versione di una base di dati o informazioni.
- cambiamento di stato = stato successivo ad una azione di aggiornamento, inserimento o cancellazione.
- vista = sottoinsieme di un database, sottoposto a limiti, e ristrutturato.
- server di oggetti = fornisce dati oggetto.
- gerarchia = struttura di un modello che assegna ogni oggetto ad un livello, e definisce per ogni oggetto l'oggetto da cui deriva.
- network = struttura di un modello che fa uso di relazioni relativamente libere tra oggetti.
- ristrutturare = dare una struttura diversa ai dati seguendo un modello differente dall'originale.
- livello = categorizzazione concettuale , dove gli oggetti di un livello inferiore dipendono da un antenato di livello superiore.
- antenato (ancestor) = oggetto di livello superiore, dal quale derivano attributi ereditabili.
- oggetto root = oggetto da cui tutti gli altri derivano, all'interno di una gerarchia.
- datawarehouse = deposito di dati integrati provenienti da una molteplicitá di risorse.
- deposito di metadata = database che contiene metadata o metainformazioni.

## A.4 Ontologia

- Ontologia = descrizione particolareggiata di una concettualizzazione, i.e. l'insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti, spesso ambigui tra domini diversi.
- concetto = definisce una astrazione o una aggregazione di oggetti per il cliente.
- semantico = che si riferisce al significato di un termine, espresso come un insieme di relazioni.
- sintattico = che si riferisce al formato di un termine, espresso come un insieme di limitazioni.
- classe = definisce metaconoscenze come metodi, attributi, ereditarietá, per gli oggetti in essa istanziati.
- relazione = collegamento tra termini, come *is-a*, *part-of*, ...
- ontologia unita (merged) = ontologia creata combinando diverse ontologie, ottenuta mettendole in relazione tra loro (mapping).
- ontologia condivisa = sottoinsieme di diverse ontologie condiviso da una molteplicitá di utenti.
- comparatore di ontologie = strumento per determinare relazioni tra ontologie, utilizzato per determinare le regole necessarie per la loro integrazione.
- mapping tra ontologie = trasformazione dei termini tra le ontologie, attraverso regole di accoppiamento, utilizzato per collegare utenti e risorse.
- regole di accoppiamento (matching rules) = dichiarazioni per definire l'equivalenza tra termini di domini diversi.
- trasformazione dello schema = adattamento dello schema ad un'altra ontologia.
- editing = trattamneto di un testo per assicurarne la conformitá ad una ontologia.
- algebra dell'ontologia = insieme delle operazioni per definire relazioni tra ontologie.

- consistenza temporale = é raggiunta se tutti i dati si riferiscono alla stessa istanza temporale ed utilizzano la stessa granularitá temporale.

- specifico ad un dominio = relativo ad un singolo dominio, presuppone l'assenza di incompatibilitá semantiche.

- indipendente dal dominio = software, strumento o conoscenza globale applicabile ad una molteplicitá di domini.

# Appendice B

# Il linguaggio di descrizione ODL$_{I^3}$

L'appendice riporta la descrizione della BNF del linguaggio ODL$_{I^3}$. Viene descritta la parte si sintassi relativa alle estensioni compiute rispetto alla grammatica ODL dello standard ODMG-93, presentata nell'appendice C.

| | | |
|---|---|---|
| ⟨interface_dcl⟩ | ::= | ⟨interface_header⟩ **{**[⟨interface_body⟩] |
| | | [**union** ⟨interface_body⟩]**};** |
| ⟨interface_header⟩ | ::= | **interface** ⟨identifier⟩ |
| | | [⟨inheritance_spec⟩] |
| | | [⟨type_property_list⟩] |
| ⟨inheritance_spec⟩ | ::= | : ⟨scoped_name⟩ [,⟨inheritance_spec⟩] |

Parte relativa alla specificazione delle classi, in cui occorre specificare il tipo e il nome della sorgente informativa.

| | | |
|---|---|---|
| ⟨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** \| **semistructured** |
| ⟨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⟩) references ⟨identifier⟩[,⟨f_key_spec⟩] |
| ... | | |

Regole di definizione del mapping tra attributi della classe globale dello scshema di mediatore ed icorrispondenti nelle sorgenti locali.

⟨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⟩ | ⟨union_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and** ⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **and** ⟨and_list⟩
⟨union_expression⟩ ::= **(** ⟨local_attr_name⟩ **union** ⟨union_list⟩ **on** ⟨identifier⟩ **)**
⟨union_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **union** ⟨union_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩.⟨class_name⟩.⟨attribute_name⟩
...

Terminological relationships usate per la definizione del Common Thesaurus.

⟨relationships_list⟩ ::= ⟨relationship_dcl⟩; | ⟨relationship_dcl⟩; ⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
⟨relationship_type⟩ ::= **SYN** | **BT** | **NT** | **RT**
...

Definizione degli OLCD integrity constraint: le regole sono definite tramite la logica di *if then* e sono valide per ogni istanza del database.
Definizione delle regole di *or* e *union*.

⟨rule_list⟩ ::= ⟨rule_dcl⟩; | ⟨rule_dcl⟩; ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_spec⟩
⟨rule_spec⟩ ::= ⟨rule_pre⟩ **then** ⟨rule_post⟩ | **{** ⟨case_dcl⟩ **}**
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨case_dcl⟩ ::= **case of** ⟨identifier⟩ **:** ⟨case_list⟩
⟨case_list⟩ ::= ⟨case_spec⟩ | ⟨case_spec⟩ ⟨case_list⟩
⟨case_spec⟩ ::= ⟨identifier⟩ **:** ⟨identifier⟩ **;**
⟨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

# Il linguaggio di descrizione ODL dello standard ODMG-93

L'appendice riporta la descrizione della BNF del linguaggio ODL dello standard ODMG-93 descritta in [57].

| | | |
|---|---|---|
| ⟨specification⟩ | ::= | ⟨definition⟩ |
| | | \| ⟨definition⟩ ⟨specification⟩ |
| ⟨definition⟩ | ::= | ⟨type_dcl⟩ **;** |
| | | \| ⟨const_dcl⟩ **;** |
| | | \| ⟨except_dcl⟩ **;** |
| | | \| ⟨interface⟩ **;** |
| | | \| ⟨module⟩ **;** |
| ⟨module⟩ | ::= | **module** ⟨identifier⟩ **{** ⟨specification⟩ **}** |
| ⟨interface⟩ | ::= | ⟨interface_dcl⟩ |
| | | \| ⟨forward_dcl⟩ |
| ⟨interface_dcl⟩ | ::= | ⟨interface_header⟩ |
| | | [**:**⟨persistence_dcl⟩] **{**[⟨interface_body⟩]**}** |
| ⟨persistence_dcl⟩ | ::= | **persistent** \| **transient** |
| ⟨forward_dcl⟩ | ::= | **interface** ⟨identifier⟩ |
| ⟨interface_header⟩ | ::= | **interface** ⟨identifier⟩ |
| | | [⟨inheritance_spec⟩] |
| | | [⟨type_property_list⟩] |
| ⟨type_property_list⟩ | ::= | **(** [⟨extent_spec⟩] [⟨key_spec⟩] **)** |
| ⟨extent_spec⟩ | ::= | **extent** ⟨string⟩ |
| ⟨key_spec⟩ | ::= | **key**[**s**] ⟨key_list⟩ |

```
⟨key_list⟩            ::=  ⟨key⟩ | ⟨key⟩ , ⟨key_list⟩
⟨key⟩                 ::=  ⟨property_name⟩ | (⟨property_list⟩)
⟨property_list⟩       ::=  ⟨property_name⟩
                           | ⟨property_name⟩ , ⟨property_list⟩
⟨property_name⟩       ::=  ⟨identifier⟩
⟨interface_body⟩      ::=  ⟨export⟩ | ⟨export⟩ ⟨interface_body⟩
⟨export⟩              ::=  ⟨type_dcl⟩ ;
                           | ⟨const_dcl⟩ ;
                           | ⟨except_dcl⟩ ;
                           | ⟨attr_dcl⟩ ;
                           | ⟨rel_dcl⟩ ;
                           | ⟨op_dcl⟩ ;
⟨inheritance_spec⟩    ::=  :⟨scoped_name⟩ [,⟨inheritance_spec⟩]
⟨scoped_name⟩         ::=  ⟨identifier⟩
                           | ::⟨identifier⟩
                           | ⟨scoped_name⟩::⟨identifier⟩
⟨const_dcl⟩           ::=  const ⟨const_type⟩ ⟨identifier⟩=⟨const_exp⟩
⟨const_type⟩          ::=  ⟨integer_type⟩
                           | ⟨char_type⟩
                           | ⟨boolean_type⟩
                           | ⟨floating_pt_type⟩
                           | ⟨string_type⟩
                           | ⟨scoped_name⟩
⟨const_exp⟩           ::=  ⟨or_expr⟩
⟨or_expr⟩             ::=  ⟨xor_expr⟩
                           | ⟨or_expr⟩ | ⟨xor_expr⟩
⟨xor_expr⟩            ::=  ⟨and_expr⟩
                           | ⟨xor_expr⟩ ^ ⟨and_expr⟩
⟨and_expr⟩            ::=  ⟨shift_expr⟩
                           | ⟨and_expr⟩ & ⟨shift_expr⟩
⟨shift_expr⟩          ::=  ⟨add_expr⟩
                           | ⟨shift_expr⟩ >> ⟨add_expr⟩
                           | ⟨shift_expr⟩ << ⟨add_expr⟩
⟨add_expr⟩            ::=  ⟨mult_expr⟩
                           | ⟨add_expr⟩ + ⟨mult_expr⟩
                           | ⟨add_expr⟩ - ⟨mult_expr⟩
⟨mult_expr⟩           ::=  ⟨unary_expr⟩
                           | ⟨mult_expr⟩ * ⟨unary_expr⟩
                           | ⟨mult_expr⟩ / ⟨unary_expr⟩
                           | ⟨mult_expr⟩ % ⟨unary_expr⟩
⟨unary_expr⟩          ::=  ⟨primary_expr⟩
                           | ⟨unary_operator⟩ | ⟨primary_expr⟩
⟨unary_operator⟩      ::=  - | + | ~
```

⟨primary_expr⟩ ::= ⟨scoped_name⟩
| ⟨literal⟩
| **(**| ⟨const_exp⟩**)**
⟨literal⟩ ::= ⟨integer_literal⟩
| ⟨string_literal⟩
| ⟨character_literal⟩
| ⟨floating_pt_literal⟩
| ⟨boolean_literal⟩
⟨boolean_literal⟩ ::= **TRUE** | **FALSE**
⟨positive_int_const⟩ ::= ⟨const_exp⟩
⟨type_dcl⟩ ::= **typedef** ⟨type_declarator⟩
| ⟨struct_type⟩
| ⟨union_type⟩
| ⟨enum_type⟩
⟨type_declarator⟩ ::= ⟨type_spec⟩ | ⟨declarators⟩
⟨type_spec⟩ ::= ⟨simple_type_spec⟩
| ⟨constr_type_spec⟩
⟨simple_type_spec⟩ ::= ⟨base_type_spec⟩
| ⟨template_type_spec⟩
| ⟨scoped_name⟩
⟨base_type_spec⟩ ::= ⟨floating_pt_type⟩
| ⟨integer_type⟩
| ⟨char_type⟩
| ⟨boolean_type⟩
| ⟨octet_type⟩
| ⟨any_type⟩
⟨template_type_spec⟩ ::= ⟨array_type⟩
| ⟨string_type⟩
| ⟨coll_type⟩
⟨coll_type⟩ ::= ⟨coll_spec⟩ <⟨simple_type_spec⟩>
⟨coll_spec⟩ ::= **set** | **list** | **bag**
⟨constr_type_spec⟩ ::= ⟨struct_type⟩
| ⟨union_type⟩
| ⟨enum_type⟩
⟨declarators⟩ ::= ⟨declarator⟩
| ⟨declarator⟩ **,** ⟨declarators⟩
⟨declarator⟩ ::= ⟨simple_declarator⟩
| ⟨complex_declarator⟩
⟨simple_declarator⟩ ::= ⟨identifier⟩
⟨complex_declarator⟩ ::= ⟨array_declarator⟩
⟨floating_pt_type⟩ ::= **float** | **double**

<table>
<tr><td>⟨integer_type⟩</td><td>::=</td><td>⟨signed_int⟩<br>⟨unsigned_int⟩</td></tr>
<tr><td>⟨signed_int⟩</td><td>::=</td><td>⟨signed_long_int⟩<br>⟨signed_short_int⟩</td></tr>
<tr><td>⟨signed_long_int⟩</td><td>::=</td><td>**long**</td></tr>
<tr><td>⟨signed_short_int⟩</td><td>::=</td><td>**short**</td></tr>
<tr><td>⟨unsigned_int⟩</td><td>::=</td><td>⟨unsigned_long_int⟩<br>⟨unsigned_short_int⟩</td></tr>
<tr><td>⟨unsigned_long_int⟩</td><td>::=</td><td>**unsigned long**</td></tr>
<tr><td>⟨unsigned_short_int⟩</td><td>::=</td><td>**unsigned short**</td></tr>
<tr><td>⟨char_type⟩</td><td>::=</td><td>**char**</td></tr>
<tr><td>⟨boolean_type⟩</td><td>::=</td><td>**boolean**</td></tr>
<tr><td>⟨octet_type⟩</td><td>::=</td><td>**octet**</td></tr>
<tr><td>⟨any_type⟩</td><td>::=</td><td>**any**</td></tr>
<tr><td>⟨struct_type⟩</td><td>::=</td><td>**struct** ⟨identifier⟩ **{**⟨member_list⟩**}**</td></tr>
<tr><td>⟨member_list⟩</td><td>::=</td><td>⟨member⟩<br>| ⟨member⟩ ⟨member_list⟩</td></tr>
<tr><td>⟨member⟩</td><td>::=</td><td>⟨type_spec⟩ ⟨declarators⟩ **;**</td></tr>
<tr><td>⟨union_type⟩</td><td>::=</td><td>**union** ⟨identifier⟩ **switch**<br>**(**⟨switch_type_spec⟩**)** **{**⟨switch_body⟩**}**</td></tr>
<tr><td>⟨switch_type_spec⟩</td><td>::=</td><td>⟨integer_type⟩<br>| ⟨char_type⟩<br>| ⟨boolean_type⟩<br>| ⟨enum_type⟩<br>| ⟨scoped_name⟩</td></tr>
<tr><td>⟨switch_body⟩</td><td>::=</td><td>⟨case⟩ | ⟨case⟩ ⟨switch_body⟩</td></tr>
<tr><td>⟨case⟩</td><td>::=</td><td>⟨case_label_list⟩ ⟨element_spec⟩ **;**</td></tr>
<tr><td>⟨case_label_list⟩</td><td>::=</td><td>⟨case_label⟩<br>| ⟨case_label⟩ ⟨case_label_list⟩</td></tr>
<tr><td>⟨case_label⟩</td><td>::=</td><td>**case** ⟨const_exp⟩**:**<br>| **default:**</td></tr>
<tr><td>⟨element_spec⟩</td><td>::=</td><td>⟨type_spec⟩ ⟨declarator⟩</td></tr>
<tr><td>⟨enum_type⟩</td><td>::=</td><td>**enum** ⟨identifier⟩ **{**⟨enumerator_list⟩**}**</td></tr>
<tr><td>⟨enumerator_list⟩</td><td>::=</td><td>⟨enumerator⟩<br>| ⟨enumerator⟩**,**⟨enumerator_list⟩</td></tr>
<tr><td>⟨enumerator⟩</td><td>::=</td><td>⟨identifier⟩</td></tr>
<tr><td>⟨array_type⟩</td><td>::=</td><td>⟨array_spec⟩<br><⟨simple_type_spec⟩**,**⟨positive_int_const⟩><br>| ⟨array_spec⟩ <⟨simple_type_spec⟩></td></tr>
<tr><td>⟨array_spec⟩</td><td>::=</td><td>**array** | **sequence**</td></tr>
<tr><td>⟨string_type⟩</td><td>::=</td><td>**string** | **string** ⟨positive_int_const⟩</td></tr>
</table>

⟨array_declarator⟩ ::= ⟨identifier⟩ ⟨array_size_list⟩
⟨array_size_list⟩ ::= ⟨fixed_array_size⟩
  | ⟨fixed_array_size⟩ ⟨array_size_list⟩
⟨fixed_array_size⟩ ::= [ ⟨positive_int_const⟩ ]
⟨attr_dcl⟩ ::= [**readonly**] **attribute** ⟨domain_type⟩
  ⟨attribute_name⟩ [⟨fixed_array_size⟩]
⟨domain_type⟩ ::= ⟨simple_type_spec⟩
  | ⟨struct_type⟩
  | ⟨enum_type⟩
⟨rel_dcl⟩ ::= **relationship** ⟨target_of_path⟩ ⟨identifier⟩
  **inverse** ⟨inverse_trasversal_path⟩
⟨target_of_path⟩ ::= ⟨identifier⟩
  | ⟨rel_collection_type⟩ <⟨identifier⟩>
⟨inverse_trasversal_path⟩ ::= ⟨identifier⟩ **::** ⟨identifier⟩
⟨attribute_list⟩ ::= ⟨scoped_name⟩
  | ⟨scoped_name⟩**,**⟨attribute_list⟩
⟨rel_collection_type⟩ ::= **set** | **list** | **bag** | **array**
⟨except_dcl⟩ ::= **exception** ⟨identifier⟩ **{**[⟨member_list⟩]**}**
⟨op_dcl⟩ ::= [⟨op_attribute⟩] ⟨op_type_spec⟩ ⟨identifier⟩
  ⟨parameter_dcls⟩ [⟨raises_expr⟩] [⟨context_expr⟩]
⟨op_attribute⟩ ::= **oneway**
⟨op_type_spec⟩ ::= ⟨simple_type_spec⟩ | **void**
⟨parameter_dcls⟩ ::= **(**[⟨param_dcl_list⟩]**)**
⟨param_dcl_list⟩ ::= ⟨param_dcl⟩
  | ⟨param_dcl⟩**,**⟨param_dcl_list⟩
⟨param_dcl⟩ ::= ⟨param_attribute⟩ ⟨simple_type_spec⟩ ⟨declarator⟩
⟨param_attribute⟩ ::= **in** | **out** | **inout**
⟨raises_expr⟩ ::= **raises** **(**⟨scoped_name_list⟩**)**
⟨scoped_name_list⟩ ::= ⟨scoped_name⟩
  | ⟨scoped_name⟩**,**⟨scoped_name_list⟩
⟨context_expr⟩ ::= **context** **(**⟨string_literal_list⟩**)**
⟨string_literal_list⟩ ::= ⟨string_literal⟩
  | ⟨string_literal⟩**,**⟨string_literal_list⟩

# Appendice D

# Sorgenti di esempio in linguaggio $\text{ODL}_{I^3}$

Di seguito é riportata la descrizione, attraverso il linguaggio $\text{ODL}_{I^3}$, dell'esempio di riferimento di Sezione 8.3.

```
Cardiology_Department (CD):

interface Patient
 ( source semistructured
      Cardiology_Department )
{  attribute string         name;
   attribute string         address;
   attribute set<Exam>      exam*;
   attribute integer        room;
   attribute integer        bed;
   attribute string         therapy*;
   attribute set<Physician> physician*; };

interface Physician
 ( source semistructured
      Cardiology_Department )
{  attribute string  name;
   attribute string  address;
   attribute integer phone;
   attribute string specialization;};

interface Nurse
 ( source semistructured
      Cardiology_Department )
{  attribute string       name;
   attribute string       address;
   attribute integer      level;
   attribute set<Patient> patient; };

interface Exam
 ( source semistructured
      Cardiology_Department )
{  attribute integer date;
   attribute string  type;
   attribute string  outcome; };
```

```
Intensive_Care_Department (ID):

interface Patient
( source relational Intensive_Care
  extent Patients
  key  code
  foreign_key(test) references Test,
  foreign_key(doctor_id) references
  Medical_Staff )
{ attribute string  code;
  attribute string  first_name;
  attribute string  last_name;
  attribute string  address;
  attribute string  test;
  attribute string  doctor_id; };

interface Medical_Staff
( source relational Intensive_Care
  extent Medical_Staffers
  key id )
{ attribute string  id;
  attribute string  first_name;
  attribute string  last_name;
  attribute integer phone;
  attribute string  address;
  attribute string  availability;
  attribute string  position; };

interface Test
( source relational Intensive_Care
  extent Tests
  key number )
{ attribute integer number;
  attribute string  type;
  attribute integer date;
  attribute string  laboratory;
  attribute string  result; };

interface Dis_Patient
( source relational Intensive_Care
  extent Dis_Patients
  key code
  foreign_key(code) references Patient)
{ attribute string  code;
  attribute integer date;
  attribute string  note; };
```

# Bibliografia


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

[2] Odb-tools Project. Available at http://sparc20.dsi.unimo.it/index.html.

[3] S. Abiteboul. Querying semi-structured data. In *Proceedings of the International Conference on Database Theory, ICDT97*, pages 1–18, Athens, Greece, 1997.

[4] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1), 1996.

[5] A. Albano, L. Cardelli, and R. Orsini. Galileo: a strongly typed, interactive conceptual language. *ACM Trans. on Database Systems*, 10(2):230–260, 1985.

[6] Pierre America and Frank van der Linden. A parallel object-oriented language with inheritance and subtyping. *ACM SIGPLAN Notices*, 25(10):161–168, October 1990. *OOPSLA ECOOP '90 Proceedings*, N. Meyrowitz (editor).

[7] 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.

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

[9] J. P. Ballerini, D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. A semantics driven query optimizer for OODBs. In A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, editors, *DL95 - Intern. Workshop on Description Logics*, volume 07.95 of *Dip. di Informatica*

*e Sistemistica - Univ. di Roma "La Sapienza" - Rapp. Tecnico*, pages 59–62, Roma, June 1995.

[10] F. Bancilhon, C. Delobel, and P. Kanellakis. *building an Object-Oriented Database System, the story of O_2*. Morgan Kaufmann Publishers, Inc., 1996.

[11] François Bancilhon, Claude Delobel, and Paris Kanellakis (eds.). *Implementing an Object-Oriented database system: The story of $O_2$*. Morgan Kaufmann, 1992.

[12] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE TSE*, 10(6):650–664, 1984.

[13] H. W. Beck, S. K. Gala, and S. B. Navathe. Classification as a query processing technique in the CANDIDE data model. In *5th Int. Conf. on Data Engineering*, pages 572–581, Los Angeles, CA, 1989.

[14] D. Beneventano, S. Bergamaschi, A. Garuti, C. Sartori, and M. Vincini. ODB-QOptimizer: un Ottimizzatore Semantico di interrogazioni per OODB. In *Terzo Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD95, Salerno*, 1996.

[15] D. Beneventano, S. Bergamaschi, A. Garuti, C. Sartori, and M. Vincini. ODB-reasoner: un ambiente per la verifica di schemi e l'ottimizzazione di interrogazioni in OODB. In *Quarto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD96, San Miniato*, pages 181–200, 1996.

[16] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. *IEEE Transactions on Knowledge and Data Engineering*, 10:576–598, July/August 1998.

[17] D. Beneventano, S. Bergamaschi, and C. Sartori. Taxonomic reasoning with cycles in LOGIDATA$^+$. In P. Atzeni, editor, *LOGIDATA$^+$: Deductive Databases with Complex Objects*, volume 701 of *Lecture Notes in Computer Science*, pages 105–128. Springer-Verlag, Heidelberg - Germany, 1993.

[18] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. ODL-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Sesto Convegno AIIA - Roma*, 1997.

[19] D. Beneventano, A. Corni, S. Lodi, and M. Vincini. ODB: validazione di schemi e ottimizzazione semantica on-line per basi di dati object oriented. In *Quinto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD97, Verona*, pages 208–225, 1997.

[20] Domenico Beneventano, Sonia Bergamaschi, and Claudio Sartori. Semantic query optimization by subsumption in OODB. In H. Christiansen, H. L. Larsen, and T. Andreasen, editors, *Flexible Query Answering Systems*, volume 62 of *Datalogiske Skrifter - ISSN 0109-9799*, Roskilde, Denmark, 1996.

[21] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. ODB-QOPTIMIZER: A tool for semantic query optimization in oodb. In *Int. Conference on Data Engineering - ICDE97*, 1997.

[22] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. Exploiting schema knowledge for the integration of heterogeneous sources. In *Sesto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD98, Ancona*, pages 103–122, 1998.

[23] 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.

[24] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. A semantic approach to information integration: the momis project. In *Sesto Convegno della Associazione Italiana per l'Intelligenza Artificiale AI*IA98, Padova*, pages 78–82, 1998.

[25] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[26] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[27] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Trans. on Database Systems*, 17(3):385–422, September 1992.

[28] S. Bergamaschi, C. Sartori, and M. Vincini. DL techniques for intensional query answering in OODBSs. In L. Dreschler-Fischer and S. Pribbenow, editors, *KI-95 Activities: Workshops, Posters, Demos*, pages 19–20, Bielefeld, Germany, 1995.

[29] S. Bergamaschi and M. Vincini. Exploiting odb-tools as a tool for the design of electrical systems. *AI*IA Notizie*, 1998.

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

[31] E. Bertino and L. Martino. *Object Oriented Database Systems*. Addison-Wesley, 1993.

[32] E. Bertino and D. Musto. Query optimization by using knowledge about data semantics. *Data and Knowledge Engineering*, 9(2):121–155, December 1992.

[33] J.M. Blanco, A. Illarramendi, and A. Goni. Building a federated relational database system: An approach using a knowledge-based system. *Int. Journal of Intelligent and Cooperative Information Systems*, 3(4):415–455, 1994.

[34] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.

[35] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.

[36] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *AAAI*, pages 34–37, 1984.

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

[38] Y.J. Breibart et al. Database integration in a distributed heterogeneous database system. In *Proc. 2nd Intl IEEE Conf. on Data Engineering, Los Angeles, CA*, 1986.

[39] M.W. Bright, A.R. Hurson, and S. Pakzad. Automated resolution of semantic heterogeneity in multidatabases. *ACM Transactions on Database Systems*, 19(2):212–253, 1994.

[40] Kim B. Bruce and P. Wegner. An algebraic model of subtypes in object-oriented languages. In *SIGPLAN Notices*, pages 163–172, 1986.

[41] M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.

[42] P. Buneman. Semistructured data. In *Proc. of 1997 Symposium on Principles of Database Systems (PODS97)*, pages 117–121, Tucson, Arizona, 1997.

[43] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proc. of ICDT 1997*, pages 336–350, Delphi, Greece, 1997.

[44] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. of the ACM SIGMOD International Conference*, pages 505–516. ACM Press, 1996.

[45] P. Buneman, L. Raschid, and J. Ullman. Mediator languages - a proposal for a standard. Technical report, University of Maryland, 1996. ftp://ftp.umiacs.umd.edu /pub/ONRrept/ medmodel96.ps.

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

[47] D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of Int. Conference on Deductive and Object-Oriented Databases*, 1995.

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

[49] Luca Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76(2/3):138–164, February/March 1988.

[50] Giuseppe Castagna. Covariance and contravariance: Conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, May 1995.

[51] 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.

[52] S. Castano and V. De Antonellis. Engineering a library of reusable conceptual components. *Information and Software Technology*, 39:65–76, 1997.

[53] 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.

[54] S. Castano, V. De Antonellis, M.G. Fugini, and B. Pernici. Conceptual schema analysis: Techniques and applications. *accepted for publication on ACM Transactions on Database Systems*, 1997.

[55] 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.

[56] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[57] R. G. G. Cattell, editor. *The Object Database Standard: ODMG93*. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[58] Edward P. F. Chan. Containment and minimization of positive conjunctive queries in OODB's. In *Principles of Database Systems*, pages 202–11. ACM, 1992.

[59] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1), 1997.

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

[61] C. Clifton, E. Housman, and A. Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *Proc. of IFIP DS-7 Data Semantics Conf.*, 1997.

[62] U. Dayal and H. Hwuang. View definition and generalization for databse integration in a multidatabase system. In *Proc. IEEE Workshop on Object-Oriented DBMS - Asilomar, CA*, 1986.

[63] D.Quass, A.Rajaraman, Y.Sagiv, J.Ullman, and J.Widom. Querying semistructured heterogeneous informations. Technical report, Stanford University, 1996.

[64] O. M. Duschka and M. R. Genesereth. Infomaster - an information integration tool. In *Proceedings of the International Workshop "Intelligent Information Integration" during the 21st German Annual Conference on Artificial Intelligence, KI-97*, 1997.

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

[66] Thomas et al. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22(3):237–266, 1990.

[67] B. Everitt. *Computer-Aided Database Design: the DATAID Project.* Heinemann Educational Books Ltd, Social Science Research Council, 1974.

[68] P. Fankhauser, M. Kracker, and E.J. Neuhold. Semantic vs. structural resemblance of classes. *SIGMOD RECORD*, 20(4):59–63, 1991.

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

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

[71] M. R. Genesereth, A. M. Keller, and O. Duschka. Infomaster: An information integration system. In *Proceedings of 1997 ACM SIGMOD Conference*, 1997.

[72] J. Gilarranz, J. Gonzalo, and F. Verdejo. Using the eurowordnet multilingual semantic database. In *Proc. of AAAI-96 Spring Symposium Cross-Language Text and Speech Retrieval*, 1996.

[73] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured data. In *23rd VLDB Int. Conf.*, 1997.

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

[75] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *ACM Symp. on Principles of Database Systems*, pages 51–61, 1997.

[76] M. Jeusfeld and M. Staudt. Query optimization in deductive object bases. In Freytag, Maier, and Vossen, editors, *Query Processing for Advanced Database System*. Morgan Kaufmann Publishers, Inc., June 1993.

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

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

[79] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The information manifold. In *In Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous*, 1995.

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

[81] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-answering algorithms for information agents. In *AAAI/IAAI*, volume 1, pages 40–47, 1996.

[82] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB 1996*, pages 251–262, 1996.

[83] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22:267–293, 1990.

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

[85] 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.

[86] B. Nebel. Terminological cycles: Semantics and computational properties. In J. F. Sowa, editor, *Principles of Semantic Networks*, chapter 11, pages 331–362. Morgan Kaufmann Publishers, Inc., San Mateo, Cal. USA, 1991.

[87] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. *SIGMOD Record*, 26(4):39–43, 1997.

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

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

[90] H. H. Pang, H. Lu, and B. C. Ooi. An efficient semantic query optimization algorithm. In *Int. Conf. on Data Engineering*, pages 326–335, 1991.

[91] 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.

[92] M.P. Reddy, B.E. Prasad, P.G. Reddy, and A. Gupta. A methodology for integration of heterogeneous databases. *IEEE Trans. on Knowledge and Data Engineering*, 6(6):920–933, 1994.

[93] S. Riccio. Elet-designer: uno strumento intelligente orientato agli oggetti per la progettazione di impianti elettrici industriali. Tesi di Laurea, Dipartimento di Scienze dell'Ingegneria, Università degli Studi di Modena, 1998.

[94] 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.

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

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

[97] A.P. Sheth and J.P. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3), 1990.

[98] P. C. Sheu, R. L. Kashyap, and S. Yoo. Query optimization in object-oriented knowledge bases. *Data & Knowledge Engineering*, 3:285–302, 1989.

[99] D. D. Straube and T. Özsu. Queries and query processing in object-oriented database systems. *ACM Trans. on Information Systems*, 8(4):387–430, October 1990.

[100] J. D Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, Maryland - USA, 1989.

[101] J. D. Ullman. Information integration using logical views. In *Intl. Conf on Database Theory*, pages 19–40, 1997.

[102] AA. VV. The common object request broker: Architecture and specification. Technical report, Object Request Broker Task Force, 1993. Revision 1.2, Draft 29, December.

[103] J. Widom. Research problems in data warehousing. In *Proc. of CIKM95*, 1995.

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

[105] W. A. Woods and J. G. Schmolze. The KL-ONE family. In F. W. Lehman, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Pubblished as a Special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2-9.

[106] Y.Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB Int. Conf.*, Bombay, India, September 1996.

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

[108] Y.Papakonstantinou, H.Garcia-Molina, and J.Widom. Object exchange across heterogeneous information sources. In *Proc. of ICDE95*, Taipei, Taiwan, 1995.