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

Architettura di un Mediatore per un Sistema di Integrazione di Sorgenti Distribuite ed Autonome

Relatore

Chiar.mo Prof. Sonia Bergamaschi

Correlatore

Dott. Ing. Maurizio Vincini

Tesi di Laurea di

Alberta Rabitti

Anno Accademico 1997 - 1998

Parole chiave:
Database Eterogenei
Intelligent Information Integration
Integrazione Semantica
Mediatore
Query Manager

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi per l'aiuto fornito alla realizzazione della presente tesi e l' Ing. Domenico Beneventano per le conversazioni risolutrici.
Un ringraziamento particolare va inoltre all'Ing. Maurizio Vincini per la preziosa collaborazione e la costante disponibilità ed all' Ing. Alberto Corni per i suggerimenti pratici.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

L'evoluzione avuta negli ultimi anni dalle reti di calcolatori e la crescente presenza di sempre nuove sorgenti di informazioni fanno aumentare non solo la quantità ed il genere di dati accessibili, ma anche la velocità con cui questi possono essere scambiati. A dispetto di questi vantaggi offerti dal panorama informatico, occorre considerare attentamente le difficoltà implicite in questo contesto. Poter accedere a moli di dati ingenti ma autonomi non aumenta la qualità finale dell'informazione percepita dall'utente che si deve destreggiare fra problemi di varia natura: dalla selezione delle sorgenti potenzialmente interessanti, alla analisi e sintesi dei dati reperiti, spesso duplicati, fra loro inconsistenti e di varia natura (testi, video, suoni, etc...). Unire ed integrare dati consente di aumentare il valore dell'informazione ad essi collegata purchè essi vengano prima selezionati, filtrati, collegati e sintetizzati [1].

La possibilità di gestire e sviluppare applicazioni intersorgenti capaci di combinare ed utilizzare questi dati è un tema di grande interesse non solo teorico ma anche applicativo, come dimostra la sempre maggiore rilevanza assunta da sistemi commerciali quali i *Datawarehouse*, i *Dataminer*, i *Sistemi di Workflow*, *le Federazioni di Database*, etc... . I fruitori dei servizi informatici, siano essi utilizzatori esperti o meno, incominciano a richiedere elaborazioni sofisticate finalizzate alla consultazione ed all'esecuzione di applicazioni capaci di sintetizzare informazioni da grandi volumi di dati distribuiti sulle reti. I domini di impiego per tali servizi sono innumerevoli. Ad esempio, in ambito ospedaliero, è molto utile poter consultare contestualmente tutte le informazioni sanitarie relative ad una cartella clinica: ECG, lastre, esiti degli esami del sangue, informazioni sui ricoveri, etc... . Altri campi in cui l'integrazione è già stata parzialmente sperimentata sono quelli militare ed aziendale: grazie all'impiego dei DSS, l'organizzazione delle attività produttive e l'effettuazione delle scelte strategiche sono avvantaggiate dalla consultazione integrata sia delle informazioni manufatturiere, quanto di quelle logistiche e commerciali. Un altro dominio d'applicabilità è quello inerente le pubblicazioni: è molto più comoda la consultazione di una vista

sempre aggiornata che sintetizzi intelligentemente informazioni su articoli, riviste, libri, etc... , appartenenti ad enti fra loro indipendenti (librerie, biblioteche, fonti universitarie, etc...), piuttosto che disporre di un'unione non ragionata delle stesse informazioni (che sarebbero quindi facilmente replicate e non sempre perfettamente consistenti tra loro a causa dei diversi momenti che nelle sorgenti sono dedicati agli aggiornamenti), o addirittura che dover accedere separatamente ad ognuna delle sorgenti, limitando così la ricerca alle sole fonti note.

Il progetto che è stato realizzato con questa tesi rientra in un lavoro maggiore per la realizzazione di un integratore per sorgenti strutturate ed autonome. Le caratteristiche salienti di quest'architettura sono studiate per risolvere i conflitti struuturali ed integrare le differenze ontologiche fra le sorgenti. In particolare l'obiettivo della presente tesi è stato quindi arricchire il progetto iniziale del Mediatore consentendo non solo l'integrazione degli schemi ma anche la fusione delle informazioni senza però dover ricorrere alla loro materializzazione. Si osservi che l'integrazione degli schemi (col termine *schema* si intende l'insieme di metadati che descrive un deposito di dati), sfruttando le informazioni semantiche, realizza in modo semiautomatico la risoluzione dei conflitti che scaturiscono dalla struttura dei dati e dai modelli utilizzati per rappresentarli, l'integrazione delle informazioni invece richiede di riconoscere, sulla base di una singola interrogazione globale, le sorgenti da interrogare e le istanze da recuperare per fonderle in modo da ottenere una risposta completa, corretta e priva di duplicati.

Questa tesi si inserisce dunque nell' ambito del **Progetto MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [2, 3, 4], nato dalla cooperazione tra il gruppo di lavoro della Professoressa Bergamaschi e lo staff della Dottoressa Silvana Castano (del Dipartimento di Scienze dell'Informazione dell'Università di Milano) con lo scopo di sviluppare un sistema di integrazione di sorgenti di informazione strutturate ma eterogenee. Il modulo **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) esistente genera una *vista* degli schemi delle sorgenti. Il progetto proposto arricchisce la fase di generazione della vista attraverso l'introduzione di **assiomi estensionali**, che aggiungono conoscenza non derivabile nè dalla relazioni terminologiche ne da quelle strutturali. Inoltre è stato pensata un'architettura che, sfruttando la conoscenza estensionale ed intensionale, ottimizzi il processo di *Query Decomposition*, permettendo di interrogare tutte e sole le sorgenti contenenti informazioni rilevanti, e consenta la fusione delle istanze onde generare la vera integrazione delle informazioni.

Elemento caratterizzante del progetto è stato l'utilizzo delle Logiche Descrittive derivate dalla Logica del Primo Ordine studiate nell'ambito dei sisremi di Intelligenza Artificiale[5]. Queste sono risultate indispensabili sia in

fase di manipolazioni degli assiomi estensionali sia per la realizzazione dell'architettura del Query Manager (oltre che per la parte precedente di integrazione degli schemi). In particolare è stato molto vantaggioso poter disporre di ODB_Tools, un ambiente Sw realizzato presso l'Università di Modena, che realizza la validazione degli schemi ad oggetti e l'espansione semantica delle query. Per la risoluzione dei conflitti intensionali e terminologici ci si è ispirati anche alle tecniche di Analisi degli Schemi, sviluppate presso il Dipartimento di Scienze dell'Informazione del Politecnico di Milano, mentre per la parte inerente la fusione delle istanze è stato fatto riferimento ad una metodologia descritta in [6], che è stata ripresa e, dove necessario, ampliata.

Il mediatore costituisce un notevole avanzamento rispetto ai sistemi preesistenti proposti in letteratura[7, 8, 9, 10, 3, 4], in quanto include alcune fasi significative del processo di integrazione realizzate in maniera automatica, inoltre definisce e manipola conoscenza estensionale, aspetto del tutto nuovo rispetto agli altri sistemi.

È stato sviluppato il modulo **IHB** (**I**nheritance **H**ierarchy **B**uilder), che genera, a partire da ogni classe della vista globale, sulla base della conoscenza intensionale ed estensionale, una gerarchia ideale. Questa gerarchia rappresenta i possibili tipi di istanze ottenute fondendo opportunamente i dati provenienti dalle sorgenti.

In particolare, per realizzare il mediatore, è stato definito un linguaggio dichiarativo (denominato ODL_{I^3}) che permetta l'inserimento di relazioni terminologiche che legano le diverse sorgenti, nonché la definizione di regole di *mapping* che favoriscano il processo di integrazione stesso[2, 3, 4]), mentre per la definizione degli assiomi estensionali si è potuto utilizzare la medesima sintassi sviluppata per le rule avendo lo stessa semantica.

Gli argomenti trattati nel seguito sono così organizzati.
Nel Capitolo 1, considerata la relativa *giovinezza* di questa area di ricerca, si è pensato fosse utile presentare un'introduzione alle problematiche dell'Integrazione di Informazioni, nonché una breve rassegna delle soluzioni più interessanti già proposte in letteratura (Capitolo 2). Nel Capitolo 3 viene data una descrizione sia degli ODB-Tools, sia della tecnica di Fusione delle Gerarchie per mettere in evidenza, successivamente nel Capitolo 5, gli arricchimenti che è stato necessario apportarvi per utilizzarla all'interno del progetto. È brevemente illustrata anche la metodologia di Analisi degli Schemi, utilizzata nello sviluppo di MOMIS. Il Capitolo 4 riporta la descrizione dell'intero progetto MOMIS, descrivendo, con l'aiuto di un esempio di riferimento, tutti i passaggi che portano alla unificazione di un insieme di sorgenti eterogenee. I Capitoli 5 e 6 descrivono il contributo originale della tesi nell' ambito del progetto MOMIS.

Capitolo 1

L'Integrazione delle Informazioni

Integrando informazioni i primi aspetti teorici da considerare sono legati alla natura dei dati (testi, immagini, suoni, record, ...) ed ai diversi tipi di sorgenti: esse possono essere pagine HTML, DBMS relazionali o ad oggetti, file system, etc... . Gli standard esistenti (TCP/IP, ODBC, OLE, CORBA, SQL, etc...) solo parzialmente risolvono i problemi legati alle diversità Hw, Sw, dei protocolli di rete e delle comunicazioni fra i moduli, mentre rimangono irrisolti quelli specifici della modellazione delle informazioni. I modelli dei dati e gli schemi si differenziano infatti gli uni dagli altri in modo da dare una struttura logica ai numerosi generi di dati da immagazzinare e questo crea una eterogeneità *semantica* (o *logica*) non risolvibile da questi standard. Inoltre si devono trovare le soluzioni a problemi quali l'*overload* delle informazioni, la riduzione dei tempi di accesso, la salvaguardia della sicurezza e della consistenza, gli elevati costi di mantenimento da affrontare per modificare, eliminare od introdurre una nuova sorgente.

Tutti i problemi elencati sottolineano la complessità degli aspetti che le architetture dedicate all'integrazione devono comprendere. Per facilitare il processo di progettazione e realizzazione di tali moduli dedicati, che siano affidabili, flessibili e tali da far fronte efficacemente ad un panorama in continua evoluzione, si cerca di sviluppare un'architettura di moduli che assicuri il riuso e la capacità di interagire con altri sistemi esistenti.

Gli approcci all'integrazione, descritti in letteratura o effettivamente realizzati, presentano diverse metodologie che vanno dalla reingegnerizzazione delle sorgenti mediante standardizzazione degli schemi e creazione di un database distribuito, all'approccio complementare denominato *repository independence*, che prevede di isolare al di sotto di una vista integrata le applicazioni ed i dati integrati dalle sorgenti, consentendo la massima autonomia

e nascondendo al contempo le differenze esistenti. Nel seguito si descrive la proposta dell'ARPA (Advanced Research Projects Agency)[11] per un'architettura che favorisca l'autonomia delle sorgenti, assicurando flessibilità e riusabilità e si presenta l'approccio seguito per il progetto MOMIS.

1.1 L'Integrazione Intelligente delle Informazioni

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

1.1.1 Il Programma I^3

Un'ambiziosa ricerca, finalizzata ad indicare un'architettura di riferimento che realizzi l'integrazione di sorgenti eterogenee in maniera automatica, è quella sviluppata dalla ARPA nel programma I^3. L'integrazione aumenta il valore dell'informazione ma richiede una forte adattabilità realizzativa: si devono poter gestire i casi di aggiornamento e sostituzione delle sorgenti, dei loro ambienti e/o piattaforme, della loro ontologia e della semantica. Le tecniche sviluppate dall'*Intelligenza Artificiale*, potendo efficacemente dedurre informazioni utili dagli schemi delle sorgenti, diventano uno strumento prezioso per la costruzione automatica di soluzioni integrate flessibili e riusabili.

Costruire supersistemi ad hoc che interessino una gran quantità di sorgenti non correlate semanticamente in modo premeditato, è faticoso ed il risultato è un sistema scarsamente manutenibile, adattabile e strettamente finalizzato alla risoluzione dei problemi per cui è stato implementato. Secondo il programma I^3, una soluzione a questi problemi deriva dall'introduzione di architetture modulari sviluppabili secondo i principi posti da uno standard che ponga le basi dei servizi che devono essere soddisfatti dall'integrazione ed abbassi i costi di sviluppo e di mantenimento. Costruire nuovi sistemi

risulta realizzabile con minor difficoltà e minor tempo (e costi) se si riesce a supportare lo sviluppo delle applicazioni riutilizzando la tecnologia già sviluppata. Per potere riusare è fondamentale l'esistenza di interfacce ed architetture standard. Il paradigma suggerito[1] per la suddivisione dei servizi e delle risorse nei diversi moduli si basa su due partizionamenti:

- quello orizzontale distingue tre livelli in database sorgenti, basi di conoscenza intermedi, applicazioni utente.
- quello verticale distingue i domini in cui raggruppare le sorgenti il cui numero si deve aggirare fra cinque e nove.

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

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

Motivazioni

Per capire come il programma I^3 sia arrivato a suggerire una certa architettura occorre mettere in evidenza i problemi che essa si deve prefiggere di risolvere:

- *L'eterogeneità delle sorgenti:* si manifesta a tutti i livelli di rappresentazione. Distingue i dati per tipi, schemi logici, lessico utilizzato per descriverli, meccanismi di accesso (interfacce di programmazione, linguaggi di interrogazione, protocolli per la gestione delle transazioni), rendendo necessaria la creazione di un modello globale per i dati da integrare.

[1] Le specifiche architetturali del programma I^3 non sono ancora state messe a punto in maniera definitiva, ma è stato sviluppato un Glossario comune (in appendice).

- *La semantica nascosta:* molti sistemi di grandi dimensioni consentono di accedere ai propri dati attraverso interfacce predefinite: in questi casi diventa spesso molto complicato dedurre le regole che consentono di interpretare ed elaborare i dati da integrare.
- *La costruzione di applicazioni indipendenti dai diversi gradi di dettaglio delle sorgenti*: spesso integrare grandi sistemi comporta una scelta sui segmenti di dati accessibili e su come farne uso: se renderli semplicemente accedibili o viceversa mantenerli trasparenti ma tradurli in informazioni implicite utili ai servizi di integrazione.
- *Le dimensioni:* non solo le sorgenti da integrare sono sempre più spesso di grandi dimensioni, ma anche il numero di sorgenti appartenenti ad uno stesso dominio è spesso molto elevato.
- *La necessità di sviluppare sistemi modulari riusabili:* è importante disporre di strumenti che facilitno e standardizzino le operazioni necessarie all'integrazione in modo da ridurre tempi e costi di sviluppo.
- *La continua evoluzione delle sorgenti:* non solo le sorgenti migliorano le piattaforme, i programmi e le rappresentazioni dei dati, ma si aggiungono sempre nuove e diverse sorgenti, perciò i sistemi progettati devono essere in grado di adattarsi a questi cambiamenti con flessibilitá e velocitá.

L'Architettura di Riferimento

In questa prima fase, il progetto proposto dall'ARPA intende dare una visione esauriente dei servizi necessari all'integrazione. In particolare individua cinque famiglie di attività omogenee, la cui reciproca interazione consente di eseguire le operazioni di comunicazione, traduzione ed integrazione dei dati nelle sorgenti. In Figura 1.1 si illustrano i principali legami fra le famiglie di servizi. I sevizi principali sono quelli di Coordinamento: essi usufruiscono delle funzionalità messe a loro disposizione dalle altre famiglie e coordinano le operazioni da eseguire sia in fase di progettazione dei "link" fra i livelli ed i domini informativi che di esecuzione in tempo reale di una particolare richiesta dell'utente. Per svolgere quest'importanti attività si devono individuare le sorgenti informative ed i diversi servizi messi a disposizione dagli altri moduli, e generare ed invocare i sistemi run-time in grado di eseguire le applicazioni stesse.

Alcune parole chiave per capire il fitto intreccio di interazioni fra queste famiglie sono qui definite:

- *Ambiente(Environment):* l'insieme dei servizi di Coordinamento e delle sorgenti e degli altri servizi I^3 che possono essere trovati per invocare un certo insieme di attività (task). Esso fornisce un accesso intelligente ad un sottoinsieme dei dati e degli applicativi messi a disposizione dall'architettura complessiva in modo finalizzato.

- *Configurazione(Configuration):* sono i sistemi run-time incaricati di eseguire uno specifico task, in particolare sono cosituiti da un insieme di tool e sorgenti interconnesse. Sono costruite sia a run che a compile time. Un ambiente può invocare più configurazioni che collaborino per raggiungere un determinato risultato.

- *Strumento(Tool):* sono incaricati di eseguire le attività, indipendentemente dalla famiglia di servizi che li invoca.

- *Schema di comportamento(Template):* costituisce la codifica di una configurazione (o di parte di essa): generalmente si tratta di strutture dati che specificano le sorgenti, i tool ed il reciproco ordine di interazione.

L'analisi del diagramma in Figura 1.1 bene evidenzia la dimensione orizzontale citata precedentemente. L'asse centrale rappresenta il flusso informativo di un dominio (o interdomini) fra le sorgenti e l'applicazioni finali. I tre livelli orizzontali rappresentano i livelli di gestione e manipolazione dei dati: quello delle sorgenti, rappresentato dai Wrapper, traduce l'informazione in un'interfaccia comune, quello intermedio analizza e sintetizza le informazioni provenienti dai diversi domini e quello di vertice mantiene riferimento dei domini e dei servizi disponibili e li invoca. Illustriamo brevemente le cinque famiglie di servizi.

Servizi di Coordinamento Questi servizi forniscono supporto sia in fase di progettazione e programmazione di nuove Configurazioni che di esecuzione a run-time delle stesse. Per poter fare questo devono disporre di conoscenza sulle sorgenti (metadati ed ontologie) e sulla natura e la struttura dei servizi esistenti: queste informazioni sono gestite dai servizi di Amministrazione. Il Coordinamento costituisce l'interfaccia con l'utente dandogli l'impressione di trattare con un sistema omogeneo: ogni richiesta dell'utente non viene gestita direttamente da questi servizi, ma si inserisce in un certo Ambiente ed origina o invoca una Configurazione: a questo punto il controllo passa ai servizi di Amministrazione che trovano le sorgenti ed i tool preposti e ne controllano lo scheduling delle attività. Illustriamo sinteticamente i principali moduli di Coordinamento.

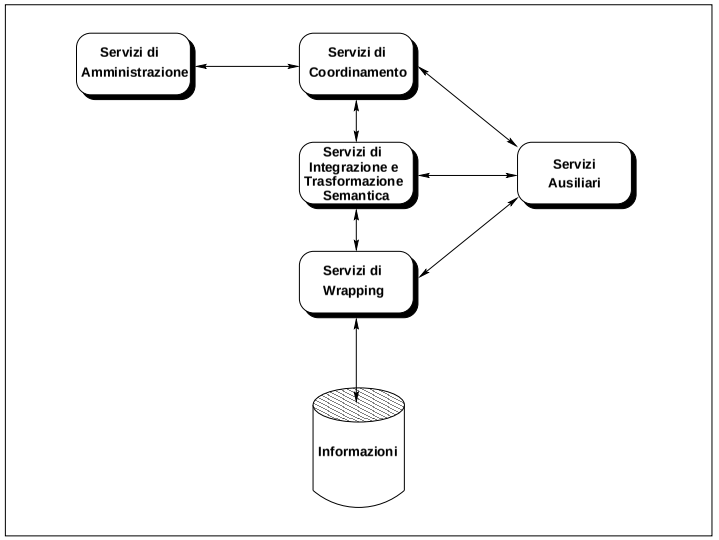


Figura 1.1: Diagramma dei servizi I^3

- **Broker:** si occupa di selezionare ed invocare i Tool capaci di soddisfare una richiesta: in questo senso interagisce coi sevizi di Amministrazione che trovano le risorse adeguate e coordinano le richieste e le risposte fra i tool che servono la richiesta specifica.

- **Facilitator:** gestisce le richieste più complesse: le traduce, le scompone, le instrada verso gli opportuni Tool che le soddisfano in maniera autonoma ed infine compone le risposte ottenute: in altre parole costruisce dinamicamente le Configurazioni. Trattandosi di un Broker con funzionalità più sofisticate oltre ai servizi di Amministrazione già citati, interagisce con sevizi Ausiliari di inferenza e con quelli di Tasformazione ed Integrazione Semantica per riuscire nel processo di scomposizione delle query.

- **Matchmaker:** appronta configurazioni dedicate a certi servizi: i componenti e le loro comunicazioni sono settate una volta per tutte, e non calcolate dinamicamente.

Servizi di Amministrazione Forniscono le informazioni relative alle sorgenti, agli strumenti disponibili per soddisfare una certa attività ed ai passi in da eseguire per soddisfare una richiesta, sono quindi indispensabili ai servizi di Coordinamento nella costruzione (statica o dinamica) delle configurazioni e dei relativi template. Inoltre una volta creati i template sono questi servizi a controllarne e schedularne l'attività. I moduli più interessanti sono:

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

- **Browser:** servono per mostrare all'utente gli schemi integrati e quelli delle sorgenti singole, in questo senso è indispensabile l'interazione coi sevizi di Wrapping, che recuperano e traducono in uno standard dichiarativo queste informazioni, e di Trasformazione Semantica per le informazioni relative all'ontologia ed al vocabolario per il mappaggio fra lo schema integrato e quelli sorgenti.

- **Iterative Query Formulation:** sono di supporto all'utente in fase di formulazione di query particolarmente complesse sullo schema integrato, specialmente qualora la query formulata non abbia prodotto informazioni interessanti, suggeriscono quasli condizioni rilasciare, come rendere più specifica o più generale la query.

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

Servizi di Integrazione e Trasformazione Semantica Sono preposti alla manipolazione semantica delle informazioni e delle applicazioni delle sorgenti: hanno come input il risultato dei servizi di Traduzione e da questi generano una vista integrata diversa dalla semplice unione delle informazioni delle sorgenti, oppure una applicazione ottenuta riusando opportunamente applicazioni che operano in ambienti autonomi. I pricipali servizi sono:

- **Integrazione degli Schemi:** si occupano della creazione dei vocabolari e delle ontologie condivise delle sorgenti, integrano gli schemi in una vista globale, mantengono il mappaggio tra gli schemi globali e le sorgenti,

- **Integrazione delle Informazioni:** aggregano, riassumono ed astraggono i dati per fornire presentazioni analitiche significative.

- **Supporto:** forniscono le funzionalità per la scomposizione delle query su un certo schema integrato e per l'aggregazione dei risultati, generano la propagazione degli update (quando prevista) e risolvono i problemi di incongruenze fra i dati da integrare. Sono forniti anche servizi di supporto all'integrazione fisica: caching delle risposte, indexing, replicazione, etc... .

Servizi Ausiliari Sono servizi di varia natura che aumentano le funzionalità e le possibilità degli altri servizi. I servizi atti ad arricchire l'espressività semantica delle sorgenti sono utilizzati maggiormente da quelli di Coordinamento, di Integrazione e di Wrapping. Altri servizi forniti sono quelli di monitoraggio e di mantenimento della consistenza fra le sorgenti.

Servizi di Traduzione Realizzano il primo passo verso l'integrazione rendendo tutte le informazioni (dati ed applicazioni) omogenee grazie alla traduzione delle stesse in un linguaggio standard. Costituiscono un'interfaccia fra le sorgenti e gli altri servizi, dunque un grande vantaggio si ha qualora questi moduli siano riusabili e coerenti con gli standard giá definiti (SQL, CORBA, OLE) e se se ne rendende il processo di costruzione automatico attraverso la costruzione di una libreria di servizi addetti.

1.1.2 OMG versus I^3

L'architettura proposta dall' ARPA non deve essere considerata indipendente dalle tecnologie che altri gruppi stanno sviluppando, anzi deve essere collocata all' interno del panorama informatico: il rispetto reciproco del lavoro realizzato dalle diverse comunità aumenta il valore delle tecnologie e della loro interazione. In particolare evidenziamo il rapporto fra l'architettura realizzata dall' OMG (Object Management Group) [12], i cui obiettivi di fondo sono lo sviluppo di uno standard e di interfacce per le comunicazioni fra gli oggetti.

Il primo obiettivo dell'ARPA è stato identificare i servizi e le loro relazioni, il successivo deve affrontare la scelta dei paradigmi di supporto all'interazione fra i vari I^3 Tool, come ad esempio il linguaggio per le comunicazioni, lo standard per le interfacce e per comporre servizi complessi. Per realizzare questa seconda fase di definizione delle interazioni fra i vari servizi dell'architettura, il confronto con la tecnologia OMG, sviluppata per supportare la

produzione di Sw riusabile, scalabile e portabile per sistemi aperti, distribuiti ed interagenti, è basilare.

L'architettura proposta dall'OMG è più generale[2] di quella I^3: mentre la seconda si occupa di descrivere e coordinare solo i servizi necessari all'integrazione, la prima si rivolge a tutti quei servizi che possono essere trattati come oggetti, indipendentemente dal loro scopo. Questo non significa però che l'approccio I^3 sia inutile, anzi esso può supportare l'architettura OMG in relazione ai compiti di integrazione, mentre OMG riesce a fornire lo standard per implementare i servizi I^3: il vantaggio di ricorrere ad essi deriva dal fatto che sono già implementati e provati ed assicurano un continuo sviluppo delle loro funzionalità.

Un limite offerto dallo standard OMG alla gestione dell'architettura I^3 è rappresentato dal linguaggio IDL. Esso è descrittivo, indipendente dal codice, orientato agli oggetti e adatto a descrivere importanti concetti del mondo dei DB, ma non è semanticamente ricco e quindi non consente di trattare adeguatamente informazioni relative alla *conoscenza*, come rule e metadati, indispensabili ai servizi di Integrazione Semantica, Ausiliari e di Amministazione. Altri linguaggi, sviluppati nell'area dei paradigmi ad Agenti, sembrano invece maggiormente adatti a favorire la comunicazione fra quei sevizi che devono trattare principalmente con la conoscenza. Al momento si sta ancora cercando di capire quale dei due protocolli sia maggiormente vantaggioso per implementare gli I^3 Tools o se sia possibile realizzare una combinazione nell'uso di entrambi.

1.2 L'Approccio Semantico all'Integrazione

Il cuore dell'architettura I^3 è rappresentato dai Servizi di Trasformazione ed Integrazione Semantica. Essi si occupano dell'integrazione vera e propria delle informazioni cercando di risolvere le differenze fra gli schemi, i modelli ed i tipi di dati, mentre le differenze dalla prospettiva delle piattaforme (Hw, DBMS, API) sono risolte dai protocolli di rete e dagli standard: SQL, OLE (Object Linking Embedded), per legare oggetti in applicazioni contenitori, ODBC (Object Database Connectivity): per la connessione fra DB eterogenee, ODMG, che fornisce un modello dei dati ed i linguaggi di descrizione, manipolazione ed interrogazione per basi di dati orientate agli oggetti, CORBA, per lo scambio degli oggetti fra sorgenti eterogenee. Un

[2]Alcuni servizi sono forniti da entrambe le architetture: ad esempio l'elaborazione di query distribuite e di trasformazione dei dati, mantenimento di informazioni sulle sorgenti, servizi di inferenza per le rule, gestione dellle modifiche delle configurazioni e degli schemi delle sorgenti

secondo problema cui si dedica la Semantica riguarda il potere espressivo delle interrogazioni: molti sistemi, specialmente quelli accessibili via WWW, supportano solo un ristretto numero di query predefinite, aggiungendo conoscenza semantica si riesce a ricondurre le query più complesse in quelle predefinite o in un loro sovrainsieme.

L'integrazione degli schemi porta alla definizione di *viste* omogenee: queste rappresentano un superschema, virtuale nella maggior parte dei casi, degli schemi delle sorgenti integrate. Le viste possono essere *convenzionali* se rappresentano in maniera omogenea gli schemi delle sorgenti o parti di esse, altrimenti sono dette *integrate*: in questo caso gli schemi sono fusi e combinati e non è più facilmente riconoscibile il contributo portato dalle diverse sorgenti [13]. La fusione degli schemi può essere completata dalla materializzazione dei dati, come ad esempio si verifica nei data warehouse: in cui i dati sono duplicati e fusi per essere conservati presso l'integratore. Questa scelta presenta vantaggi in termini di velocità di risposta, specialmente quando la fusione dei dati richiede l'esecuzione di join su molti attributi, però diventa pesante il mantenimento della consistenza tra le sorgenti e la vista. In ragione di queste considerazioni si sta diffondendo sempre più l'approccio virtuale: esso consiste nel non replicare i dati, ma nell'eseguire a run-time la decomposizione della query globale, l'interrogazione delle sorgenti e la fusione delle informazioni.

1.2.1 Trattamento dei Metadati e delle Ontologie

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

Metadati Essi rappresentano informazioni relative agli schemi: significato delle classi (o relazioni) e delle loro proprietà (attributi), relazioni sintattiche e/o semantiche esistenti tra i modelli nelle sorgenti, caratteristiche delle istanze nelle sorgenti: tipi degli attributi, valori null. Ad esempio una proprietà di una classe in una sorgente, sia ad esempio l'attributo `impiego` della classe `Persone` ha lo stesso significato dell'insieme di tabelle `Dirigenti, Impiegati, Operai` in un'altra sorgente. Un altro caso di relazione fra le sorgenti si ha quando classi, attributi, metodi hanno lo stesso significato ma nomi diversi, o nomi uguali ma significati diversi.

Un'importante osservazione riguardo il trattamento dei metadati deve essere fatta in relazione alla crescente disponibilità di dati semistrutturati o i

cui schemi sono soggetti a frequenti cambiamenti nel tempo. L'integrazione di questi comporta ulteriori complicazioni in quanto non si possono stabilire dei mapping statici fra gli schemi: gli schemi integrati dati che saranno generati dalla fusione non sono conosciuti a priori e dunque non possono essere creati se non dinamicamente contemporaneamente alla generazione delle istanze. Inoltre l'interrogazione stessa di sorgenti destrutturate genera naturali difficoltà: in primo luogo perchè queste sorgenti consentono solo un ristretto genere di interrogazioni standard (template) e quindi complicano la formulazione delle interrogazioni e la conseguente soddisfazione della richiesta. Ad esempio un'interrogazione può essere formulata ponendo un vincolo su di un attributo che in realtà non è presente con quel nome. In considerazione di quest'ultima osservazione o del fatto che molti sistemi consentono solo alcune interrogazioni standard si può affermare che la deduzione dei metadati porta ad un arricchimento semantico. Ad esempio si potrebbe migliorare la definizione della tabella `Impiegati` inserendo la proprietà: `impiego` il cui valore è `"impiegati"` e viene assegnato di default.

Ontologie Si intende "l'insieme dei termini e delle relazioni esistenti in un dominio per denotare concetti ed oggetti". Un'ontologia relativa ad un insieme di sorgenti è costituita da tutti quei termini che identificano in maniera non ambigua lo stesso concetto o la stessa porzione di conoscenza. Un'ontologia può essere più o meno generale a seconda del livello cui ci si riferisce: se ci si riferisce ad una precisa applicazione l'ontologia sarà limitata dipendendo dal dominio e dall'obiettivo della stessa. Le ontologie di livello superiore sono più vaste riferendosi solo al dominio ma essendo indipendenti dall'obiettivo, quelle di livello ancora superiore sono indipendenti anche dal dominio e definiscono concetti molto generali. È lecito supporre che integrando sorgenti seppur autonome il livello dell'ontologia sia vincolato all'interno di un certo dominio; questa restrizione è congruente con l'idea di integrare i domini comuni di sorgenti che descrivono campi almeno parzialmente sovrapposti.

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

diversi e con fini del tutto autonomi.

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

1.3 Progetto del Mediatore MOMIS

1.3.1 Specifiche Architetturali

Si è già evidenziato come il servizio strategico delle architetture I^3 sia incaricato di risolvere l'eterogeneità semantica esistente fra le sorgenti per generare una vista integrata che sia interrogabile dagli utenti finali, nascondendo le singole sorgenti. L'approccio che è stato deciso di seguire si ispira ad altre architetture già studiate e sviluppate (ad esempio TSIMMIS di Stanford): in particolare si è ripresa l'idea dei *Wrapper*, sistemi incaricati di tradurre i dati delle sorgenti rendendoli omogenei ad un'interfaccia comune, e dei *Mediator*, che si occupano di combinare, integrare ed arricchire gli schemi ed i modelli delle sorgenti e di fornire facilitazioni e supporto per l'interrogazione e la generazione delle risposte.

I principali problemi nella realizzazione di queste architetture si riscontrano in relazione ai seguenti aspetti:

- nello sviluppo di generatori di Wrapper, che permettano di sviluppare velocemente Wrapper ad hoc per ogni tipo di sorgente (relazionale, ad oggetti, file system, semistrutturata, ...) in grado di interpretare il modello logico dei dati della sorgente e di convertirlo in un modello standard, di tradurre la query nel linguaggio di interrogazione della sorgente e di convertire i dati in risposta nel formato comune (ad esempio da tupla ad oggetto),
- nello sviluppo del Mediatore che a partire dal modello standard dei dati riesca a realizzare lo schema integrato in modo automatico o almeno semiautomatico secondo una procedura standardizzata che sia riutilizzabile,
- nella generazione di un modulo che fornisca le funzionalità di *Decomposizione delle Interrogazioni* e di *Fusione delle Risposte* che fornisca

risposte corrette e complete. La correttezza è legata alla fase di fusione delle istanze, la completezza all'ottenimento dell'insieme di oggetti strettamente necessari: si devono selezionare efficacemente fra le grandi moli di dati e di sorgenti quelle strettamente necessarie a dare una risposta esauriente evitando quelle inutili.

Questo lavoro è stato dedicato a completare la progettazione dell'architettura del modulo MOMIS, presentata in [2, 3, 4], migliorando il processo di integrazione degli schemi ed introducendo la fase di scomposizione delle query e di generazione dei risultati.

La base teorica dell'approccio proposto nell'architettura del Mediatore[3] si fonda sullo standard ODMG93, su ODB-Tools, un componente basato sulla logica descrittiva [14, 15] e sull' architettura presentata in TSIMMIS [7, 8]. In particolare ODM ed ODL di ODMG93sono presi come modello dei dati e linguaggio di descrizione degli schemi interni e quindi rappresentano lo standard di riferimento dei Wrapper. Inoltre ODL è stato esteso per renderlo in grado di esprimere conoscenza non solo intensionale ed ontologica ma anche estensionale e vincoli, attraverso relazioni terminologiche e rule. OQL è stato limitato in modo che le query siano traducibili anche su sorgenti non a oggetti assumendo il linguaggio proposto come standard dal progetto DARPA I^3 [11]. Il componente basato sulla logica descrittiva, ODB-Tools, è dotato di interfacce verso tutti questi linguaggi ed è in grado non solo di calcolare incoerenza e sussunzione e di eseguire l'ottimizzazione semantica delle query, ma di realizzare l'integrazione degli schemi.

In merito a quanto è stato considerato sul linguaggio interno di comunicazione fra i servizi si è potuto risolvere il problema evidenziato in 1.1.2 decidendo per un linguaggio con caratteristiche ibride. L'estensione di ODL consente di esprimere le rule che sono poi interpretate dal modulo basato sulla Logica Descrittiva (DL) con un approccio di mondo aperto (OWA). Il vantaggio duplice ricavato da questa soluzione risiede nel:

- disporre ad alto livello di un linguaggio descrittivo, indipendente dalle sorgenti, accettato come standard, orientato agli oggetti ed estremamente diffuso nel mondo delle basi di dati
- poter disporre degli strumenti di manipolazione della conoscenza tipici delle logiche descrittive ed in particolare delle tecniche di inferenza utili al trattamento dei metadati e delle ontologie

[3] Ovviamente esistono dei presupposti in assenza dei quali non si riesce a ricorrere a questa soluzione di tipo semantica. In particolare ogni sorgente deve essere caratterizzata da uno schema concettuale da cui inferire le ontologie ed i metadati e devono essre traducibili in un modello comune per poter eseguire l'unificazione o integrazione degli schemi.

In favore di questo approccio si hanno le seguenti argomentazioni:

- I modelli orientati agli oggetti sono sempre più utilizzati non solo nelle aree della programmazione e delle basi di dati ma anche dell'intelligenza artificiale perchè adatti sia alla descrizione degli aspetti intensionali ed estensionali della conoscenza (classi, gerarchie, composizione) sia di comportamento (metodi).
- La grande quantità di standard disponibili per la manipolazione e lo scambio degli oggetti fra sorgenti distinte (ODM, ODL, OQL, OLE, CORBA, ODBC, ...) è un patrimonio esistente e ben sperimentato che non deve andare perso e che risulta estremamente utile in tema di integrazione delle informazioni inquanto *integrare* significa scambiare e *informazioni* implica automaticamente le idee di modellazione, descrizione ed interrogazione.
- Le informazioni estensionali, semantiche ed ontologiche codificate in uno schema possono essere facilmente estratte ed organizzate grazie agli strumenti della logica descrittiva
- L'architettura sostenuta dai moduli dei Wrapper e del Mediatore consente la riusabilità dei moduli. Inoltre presenta versabilità ed adattabilità nei confronti dell'aggiunta delle sorgenti o di modifiche delle loro piattaforme o degli schemi.

1.3.2 Specifiche del Query Manager

L'architettura del Mediatore definita in [16] consente di risolvere i problemi di incompatibilità intensionale e di generare la vista utilizzabile dagli utenti finali per porre le interrogazioni. Il contributo innovativo proposto in questo lavoro stabilisce sulla base delle conoscenze estensionali di migliorare il processo di integrazione degli schemi e di ottimizzare la fase di query processing. Il miglioramento in fase di generazione della vista viene realizzato aggiungendo all'analisi intensionale e terminologica una fase di analisi estensionale basata sulla necessaria proposizione di assiomi estensionali. La modifica introdotta rende più completo il processo di integrazione perchè consente di interagire col processo di generazione dello schema globale introducendo informazioni sulle estensioni. Inoltre dá la possibilità di ottenere una risposta ad un'interrogazione il più possibile completa, oltre che corretta, inquanto consente di rimediare ad eventuali sviste create dall'analisi terminologica ed intensionale. Il processo di decomposizione e traduzione delle query risulta ottimizzato perchè consente di selezionare con precisione solo le sorgenti

che possono dare un significativo contributo alla risposta, risparmiando sia in tempi di accesso delle sorgenti, che in tempi di generazione dei risultati ottenendo un insieme molto più ristretto di istanze, ma sempre completo.

In particolare è stato studiato un procedimento per generare una gerarchia di classi relativa ad ogni vista dello schema globale. La gerarchia viene realizzata combinando le informazioni intensionali ed estensionali sulle sorgenti così come descritto in [6], una volta ottenuta questa gerarchia ed avendo a disposizione ODB-Tools, strumento in grado di espandere semanticamente un'interrogazione su di una gerarchia, si riesce a ricondursi all' insieme ottimo di sorgenti da interrogare ed a realizzare la fusione delle istanze. Il modulo realizzato IHB (Inheritance Hierarchy Builder) realizza l'interpretazione della conoscenza estensionale attraverso l'interazione con ODB-Tools, consentendo di evidenziare tutte le relazioni fra le estensioni delle classi, e poi genera la gerarchia.

Capitolo 2

Stato dell' Arte

Nonostante l' Integrazione Intelligente di Informazioni sia un campo di ricerca relativamente nuovo, esistono già in letteratura diversi sistemi che cercano di realizzare, in modo più o meno efficiente e flessibile, un modulo integratore (nei casi più riusciti) o un semplice modulo di ricerca di informazioni. Ancora prima di iniziare la fase di analisi del problema, si è ritenuto quindi utile esaminare alcuni di questi sistemi, alla ricerca delle soluzioni migliori. Per non appesantire questo capitolo, e per poter descrivere i sistemi presentati in modo sufficientemente esauriente, si è scelto di limitare a tre il numero di progetti presentati, cercando di selezionare i più significativi. Si rimanda comunque alla bibliografia per una analisi più approfondita sia di questi sistemi, sia di altri [17, 18, 19, 20].

2.1 TSIMMIS

TSIMMIS (**T**he **S**tanford- **I**BM **M**anager of **M**ultiple **I**nformation **S**ources) [8, 21, 7] 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

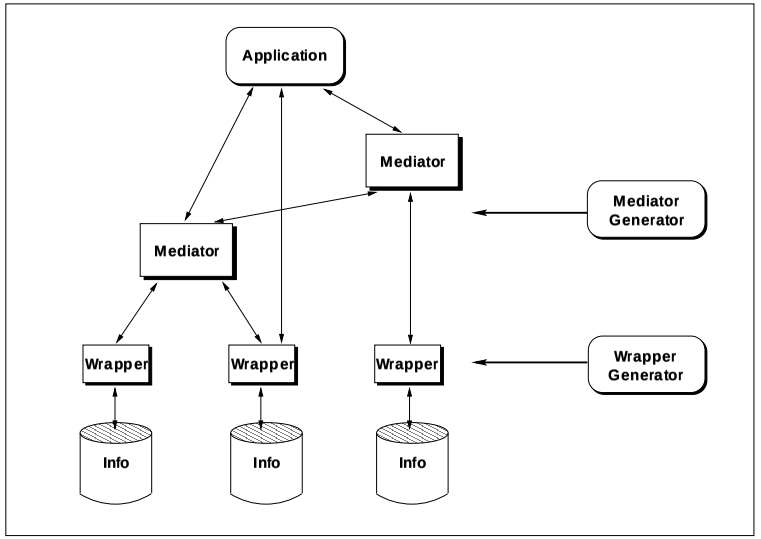


Figura 2.1: Architettura TSIMMIS

ai mediatori (ricevendo dati integrati da più sorgenti), sia direttamente ai wrapper (interrogando in questo modo un' unica fonte).

In Figura 2.1 è mostrata l'architettura: ad ogni sorgente corrisponde un 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) [22] 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 pro-

gettati 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.

2.1.1 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 semistrutturati, non dovendo predefinire le strutture che questi oggetti dovranno seguire, ed anzi accettando tutti gli oggetti con una determinata etichetta, qualunque schema seguano.

2.1.2 Il linguaggio MSL

MSL è il linguaggio utilizzato per definire, in modo dichiarativo, i mediatori: attraverso l' uso di *rule*, si può specificare il punto di vista del mediatore, ed in questo modo definire lo "schema globale" in modo manuale. 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 (si rimanda alla Sezione 5.2.2 la descrizione dettagliata di questo processo). 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 2.1.4

2.1.3 Il generatore di Wrapper

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

Per facilitare l'intero processo 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'.

2.1.4 Il generatore di Mediatori

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


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

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

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

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

Computer_Science, il cui schema è riportato di seguito:

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

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

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

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

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 ap-

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

Figura 2.4: Oggetti esportati da **MED**

partenga al dipartimento 'CS' (ovvero Computer Science), la risposta che si vorrebbe ottenere è rappresentata in Figura 2.4.

Per realizzare questa integrazione, TSIMMIS fa uso di un sistema di regole, MSL (Mediator Specification Language), che permettono di specificare la struttura dell'oggetto integrato, 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
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 il ruolo specificato nell'intestazione, e può valere '`employee`' o '`student`'), e

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

2.1.5 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) [23]. 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 attributi 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.

2.1.6 Pregi e difetti di TSIMMIS

Il punto di forza di questo progetto è sicuramente il modello comune OEM adottato: 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 2.1.4), rendendo possibile ciò che è negato in qualunque ambiente convenzionale orientato agli oggetti, che non aderisca ad una semantica di mondo aperto. Queste possibilità rendono l'intero sistema estremamente flessibile, permettendo l'integrazione di sorgenti il cui schema può essere sia parzialmente sconosciuto, sia variabile nel tempo.

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. Un vantaggio consiste 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.

2.2 GARLIC

L'obiettivo del progetto **GARLIC** [24, 25] (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.

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

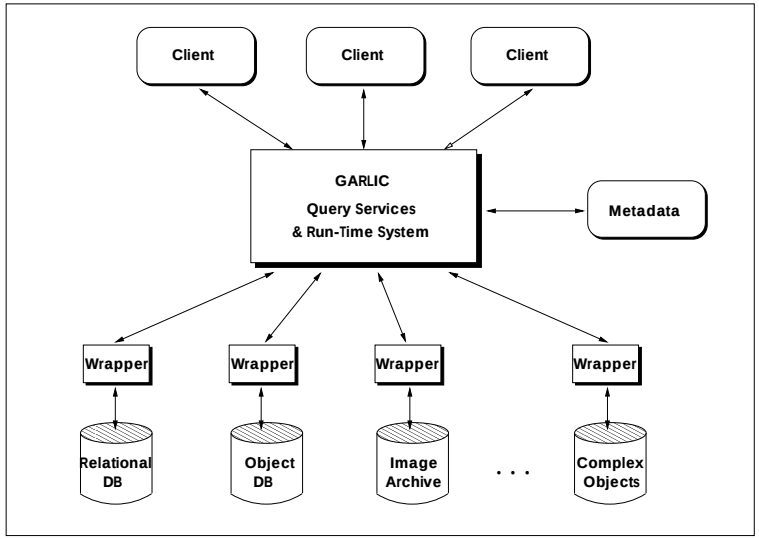


Figura 2.5: Architettura GARLIC

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:

Relational Repository Schema

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

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

Web Repository Schema

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

Image Server Repository Schema

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

Figura 2.6: GDL schema

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].

2.2.1 Il linguaggio GDL

Tra i vari compiti dei wrapper vi è quello di fornire una descrizione del contenuto della sorgente da loro servita, utilizzando il **G**arlic **D**ata **L**anguage, o **GDL**. GDL è un variante dell'ODMG[2] **O**bject **D**escription **L**anguage [26]: 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 2.6.

L' esempio considera una semplice applicazione per una agenzia di viaggio: l'agenzia gestisce informazioni sugli stati e sulle città per le quali organizza viaggi (in un db relazionale), nonché un sito web per la prenotazione

[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.

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

2.2.2 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 de-

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

2.2.3 Pregi e difetti 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 2.1.1), a cui è stato preferito l'utilizzo degli schemi locali (descritti attraverso il modello GDL): a fronte di una perdita di flessibilità dell'intero sistema (è ora praticamente impossibile gestire anche sorgenti 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 [24, 25].

Sostanzialmente diverse da progetti analoghi sono comunque le funzioni del wrapper: da semplice traduttore di linguaggi e protocolli, in GARLIC il wrapper include molte delle funzioni demandate in altri sistemi al mediatore vero e proprio. Se dal punto di vista architetturale questo potrebbe anche essere considerato un errore, risulta sorprendente (e forse poco credibile) la sostanziale differenza dei tempi di sviluppo dichiarata da TSIMMIS e da GARLIC: mentre in TSIMMIS si considera ragionevole impiegare circa 6 mesi per realizzare un semplice wrapper, nel progetto GARLIC può essere sviluppato in poche settimane...

2.3 SIMS

SIMS [9, 10], 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.

Si basa sulle seguente idee di fondo:

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

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

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 (vedi Capitolo 3) derivato dal KL-ONE (modello sviluppato da Brachman nel 1976 in [27]).

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.

2.3.1 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 2.7, che richiede la profondità del porto di SAN-DIEGO) usando solo la terminologia del *domain model*, essendo in questo modo esonerato dal dover conoscere tutte le sorgenti integrate nel sistema (benché possa pure interagire direttamente con alcune di esse, se ha particolare familiarità con i loro termini).

La parte critica dell'intero processo è invece la fase di integrazione delle sorgenti, ovvero il collegare lo schema globale alle varie sorgenti locali. Questo consiste nel descrivere tutti i concetti e le relazioni di una sorgente attraverso i termini del *domain model*. Questo mapping deve essere fatto per tutte le sorgenti, ed in modo particolarmente accurato: un anello di mapping tra un concetto globale ed uno locale significa che i due concetti rappresentano la stessa classe di informazioni (ed analogamente per gli attributi). Un esempio qualitativo di questo mapping è riportato in Figura 2.8: se l'utente richiede

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

Figura 2.7: Esempio di query SIMS

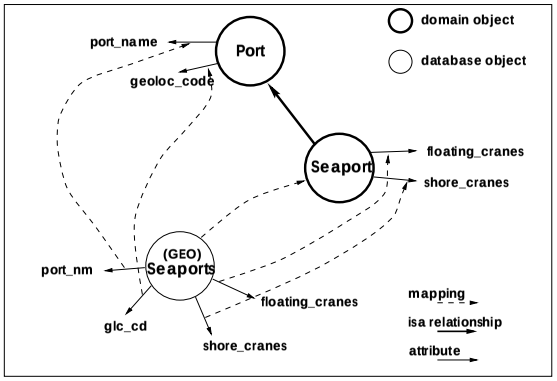


Figura 2.8: Mapping tra *domain model* e modello locale

tutti gli elementi della classe globale `Seaport`, andrà interrogata la classe `Seaports` della sorgente `GEO`.

2.3.2 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 2.9.

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

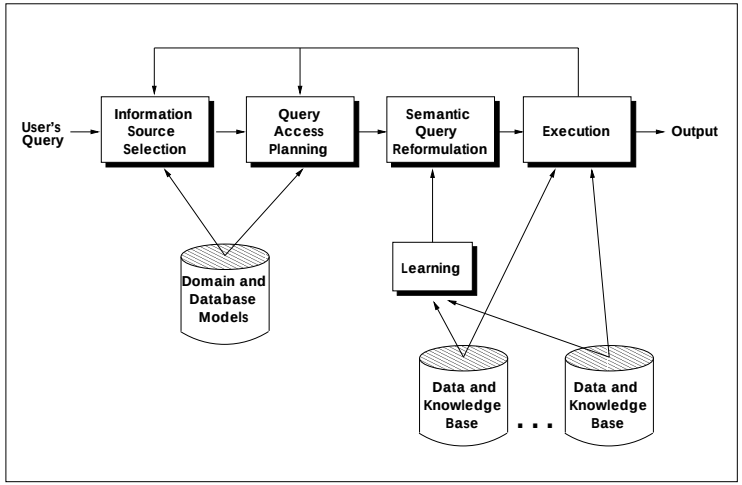


Figura 2.9: Query Processing

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.

2.3.3 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, la automazione della fase di integrazione degli schemi locali, realizzata dal nostro sistema, mentre risulta possibile usufruire della conoscenza semantica degli schemi sorgenti, cosa che negli altri approcci è stata trascurata.

2.4 Osservazioni

È stato scelto di illustrare questi tre sistemi perchè presentano approcci complementari al vasto problema dell' integrazione delle informazioni:

- TSIMMIS realizza l' integrazione di sorgenti semistrutturate, trascurando necessariamente la possibilità di ottimizzazione delle fasi di query processing possibili usufruendo dei mapping fra gli schemi

- GARLIC, viceversa, mostra un approccio più orientato ad una visione DBMS e commerciale, però trascura di trattare aspetti più profondi impliciti nelle ontologie delle sorgenti autonome

- SIMS coglie l' importanza della conoscenza semantica delle sorgenti e la utilizza per completare la fase di integrazione nel processo di ottimizzazione delle query, però non supporta la fase di integrazione degli schemi, né estende la possibilità di esprimere la conoscenza inter sorgenti, oltre che fra le sorgenti e lo schema globale

Il progetto che si presenterà nel seguito di questo lavoro fornisce un' ulteriore avanzamento rispetto all' idea si SIMS, in quanto consente di rendere semi_automatica la fase di integrazione degli schemi e di esprimere e legare conoscenza non solo fra le sorgenti e lo schema globale, ma anche fra le sorgenti stesse Quest' ultima funzionalità rende effettivamente traducibile il concetto di ottimizzazione semantica di un' interrogazione anche all' ambiente distribuito ed autonomo: la *riformulazione* delle query porta a trovare l' insieme minimo e meno *costoso* di query da inviare alle sorgenti.

Capitolo 3

Strumenti Utilizzati

L' obiettivo di questa tesi è completare il Mediatore MOMIS attraverso lo sviluppo di un modulo in grado di interpretare la conoscenza estensionale, automatizzando il processo di integrazione delle sorgenti di informazione.

L' interpretazione della conoscenza estensionale, così come definita nel Capitolo 1, comporta una forma di *comportamento intelligente*. Progettare un modulo in grado di manifestare una tale caratteristica richiede di utilizzare tecniche di Intelligenza Artificiale. La comunità che fa capo alle tecniche dell' AI, utilizzando la logica del primo ordine, riconduce tutti i problemi di conoscenza esprimibili, in problemi di ricerca, facilmente traducibili in un algoritmo per calcolatore [28]. In particolare, in questa tesi, si è ricorso all' utilizzo di algoritmi di inferenza, basati sulla logica descrittiva `ocdl` [29], che vanno sotto il nome di ODB-Tools, sviluppati presso il Dipartimento di Ingegneria dell'Università di Modena.

La descrizione di questi strumenti è stata riportata per sottolineare, nei prossimi capitoli, le estensioni, teoriche e applicative, che è stato necessario apportarvi per arrivare a definire il modulo IHB che realizza la fase di interpretazione e manipolazione degli assiomi estensionali.

3.1 ODB-Tools

ODB-Tools è un sistema per l'acquisizione e la verifica di consistenza di schemi di basi di dati e per l'ottimizzazione semantica di interrogazioni nelle basi di dati orientate agli oggetti (OODB). È stato sviluppato presso il Dipartimento di Scienze dell'Ingegneria dell'Università di Modena [14, 15].

L' impianto teorico di ODB-Tools è costituito da:

1. `ocdl` (**O**bject **C**onstraint **D**escription **L**ogics), un formalismo capace di esprimere:
   - descrizioni di classi,
   - vincoli di integrità,
   - interrogazioni

   ed in grado di eseguire inferenze basate sul calcolo della sussunzione;
2. **espansione semantica** di un tipo, realizzata attraverso l'algoritmo di sussunzione. Questa consente operazioni vantaggiose per le fasi di progettazione di schemi ad oggetti rendendo implementabile la verifica di coerenza; inoltre, in fase di Query Processing, supporta il meccanismo di ottimizzazione.

Analizziamo in maniera più approfondita gli elementi che sono di supporto allo sviluppo dei moduli degli ODB-Tools, definendo le logiche descrittive (DLs) ed in particolare `ocdl`. Illustriamo infine l' architettura di questo sistema.

3.1.1 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 si introducono i concetti fondamentali che stanno alla base di qualsiasi Sistema a Base di Conoscenza (KBRS): questi concetti sono stati volontariamente semplificati e riassunti per cercare di comunicarne gli aspetti essenziali, senza creare ulteriori complicazioni, che non potrebbero essere approfondite in questo lavoro.

La Logica del Primo Ordine

Si è già detto che la necessità di rappresentare la conoscenza serve per ricondurre 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 step analoghi:

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 $1^{st}OPL$.

Syntax 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*,

[1] Tipici problemi 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

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

necessaria ad esprimere relazioni *if then*. Questi elementi determinano la *Predicate Logic*, per parlare di $1^{st}OPL$ la si estende introducendo le variabili ed i *quantificatori* esistenziale ed universale.

Aspetti Generali

Le DLs costituiscono frammenti delle $1^{st}OPL$ [30]: esse consentono di esprimere i *concept*[3] sottoforma di formule logiche, usando solamente predicati unari e binari, e contenenti solo una variabile (corrispondente alle istanze del concept). Un grande vantaggio offerto dalle DLs, per le applicazioni di tipo DBMS, è costituito dalla capacità di rappresentare la semantica dei modelli di dati ad oggetti complessi (*CODMs*), recentemente proposti in ambito di basi di dati deduttive e basi di dati orientate agli oggetti. Questa capacità nasce 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 riesce nell' ambizioso fine 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 `ocdl` deriva dal precedente **ODL**, proposto in [29], che già estendeva l'espressività dei linguaggi sviluppati nell' area delle DLs. La caratteristica principale di **ODL** consiste nell'assumere una ricca struttura per il sistema dei tipi di base: oltre ai classici tipi atomici *integer*, *boolean*, *string*, *real*, e tipi *mono-valore*, viene ora considerata anche la possibilità di utilizzare dei sottoinsiemi di questi (come potrebbero essere, ad esempio, intervalli di interi). Sulla base di questi tipi di base si possono definire i *tipi valore*, attraverso gli usuali costruttori di tipo definiti nei *CODMs*, quali *tuple*, *insiemi* e *tipi classe*, che denotano insiemi di oggetti con una identità ed un valore associato. Ai tipi può poi essere assegnato un nome, mantenendo la distinzione tra nomi di *tipi valore* e nomi di *tipi classe*, che d'ora in poi denomineremo semplicemente *classi*: ciò equivale a dire che i nomi dei tipi vengono partizionati in nomi che indicano insiemi di oggetti (*tipi classe*) e nomi che rappresentano insiemi di valori (*tipi valore*).
ODL introduce inoltre la distinzione tra nomi di tipi *virtuali*, che descrivono condizioni necessarie e sufficienti per l'appartenenza di un oggetto del dominio ad un tipo (concetto che si può quindi collegare al concetto di *vista*), e nomi di tipi *primitivi*, che descrivono condizioni necessarie di appartenenza (e che quindi si ricollegano alle classi di oggetti). In [31], **ODL** è stato esteso

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

per permettere la formulazione dichiarativa di un insieme rilevante di vincoli di integrità definiti sulla base di dati. L'estensione di **ODL** con vincoli è stata denominata `ocdl`(**O**bject **C**onstraint **D**escription **L**ogics). Attraverso questo logica descrittiva, è possibile descrivere, oltre alle classi, anche le *regole di integrità*: permettono la formulazione dichiarativa di un insieme rilevante di vincoli di integrità sottoforma di regole *if-then* i cui antecedenti e conseguenti sono espressioni di tipo **ODL**. In tale modo, è possibile descrivere correlazioni tra proprietà strutturali della stessa classe, o condizioni sufficienti per il popolamento di sottoclassi di una classe data. In altre parole le rule costutuiscono un modo più generale per descrivere gli oggetti che popolano il sistema.
In [29] è stato presentato il sistema OCDL-Designer, per l'acquisizione e la validazione di schemi OODB descritti attraverso `ocdl`, che preserva la consistenza della tassonomia di concetti ed effettua inferenze tassonomiche. In particolare, il sistema prevede un algoritmo di *sussunzione* che determina tutte le relazioni di specializzazione tra tipi, e un algoritmo che rileva gli eventuali tipi *inconsistenti*, cioé tipi necessariamente vuoti.
In [31] l'ambiente teorico sviluppato in [29] è stato esteso per effettuare l'ottimizzaziome semantica delle interrogazioni, dando vita al sistema ODB-QOptimizer. Sono state inoltre sviluppate delle interfacce software per tradurre descrizioni ed interrogazioni espresse rispettivamente nei linguaggi ODL e OQL (proposti dal gruppo di standardizzazione ODMG-93 [26]) in `ocdl`.

Ottimizzazione semantica La nozione di ottimizzazione semantica di una query è stata introdotta, per le basi di dati relazionali, da King [32, 33] e da Hammer e Zdonik [34]. 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 [35].
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.

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

Riportiamo in questa sezione la sintassi del linguaggio `ocdl`, nonché una breve descrizione dei concetti di *espansione semantica* e *relazione di sussunzione*. `ocdl` deriva dalla $1^{st}OPL$, opportunamente ridotta per consentire la decidibilità di ogni suo processo di inferenza. Nel corso della descrizione, si farà inoltre riferimento ad un esempio esplicativo, di un dominio `Magazzino`, di cui riportiamo la descrizione in sintassi ODMG-93 nella tabella 3.1, e le regole di integrità su di esso definite in tabella 3.2.

L'esempio considerato è relativo alla struttura organizzativa di una società: i materiali (`Material`) sono descritti da un nome, un rischio e da un insieme di caratteristiche; gli `SMaterial` sono un sottoinsieme dei `Material`. I `DMaterial` (materiali "pericolosi") sono un sottoinsieme dei `Material`, caratterizzati da un rischio compreso tra 15 e 100. I manager hanno un nome, un salario compreso tra 40K e 100K dollari e un livello compreso tra 1 e 15.

Classi dello Schema

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

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


```
rule R1   for all X in Material (X.risk ≥ 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 3.2: Regole di integrità sul dominio `Magazzino`

I `TManager` sono manager che hanno un livello compreso tra 8 e 12. I magazzini (`Storage`) sono descritti da una categoria, sono diretti (`managed_by`) da un manager e contengono (`stock`) un insieme di articoli (`item`) che sono dei materiali, per ciascuno dei quali è indicata la quantità presente; è inoltre riportato il rischio massimo (`maxrisk`) ammissibile per i materiali conservati. Gli `SStorage` sono un sottoinsieme dei magazzini. Infine vi sono i magazzini "pericolosi" (`DStorage`) che sono un sottoinsieme degli `Storage` caratterizzati dallo stoccaggio di soli materiali "pericolosi" (`DMaterial`).

Schema e istanza del database

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

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

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

Gli identificatori di oggetti sono associati a valori tramite una *funzione totale* δ da $\mathcal{O}$ a $\mathcal{V}(\mathcal{O})$; in genere si dice che il valore $\delta(o)$ è lo *stato* dell'oggetto identificato dall'oid o.
Sia **N** l'insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che

$\mathbf{A}$, $\mathbf{B}$, e $\mathbf{N}$ siano a due a due disgiunti. $\mathbf{N}$ è partizionato in tre insiemi $\mathbf{C}$, $\mathbf{V}$ e $\mathbf{T}$, dove $\mathbf{C}$ consiste di nomi per *tipi-classe base* $(C, C' \ldots)$, $\mathbf{V}$ consiste di nomi per *tipi-classe virtuali* $(V, V' \ldots)$, e $\mathbf{T}$ consiste di nomi per *tipi-valori* $(t, t', \ldots)$. Un *path* p è una sequenza di elementi $p = e_1.e_2.\ldots.e_n$, con $e_i \in \mathbf{A} \cup \{\triangle, \forall, \exists\}$. Con ϵ si indica il path vuoto.
$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$[4] indica l'insieme di tutte le *descrizioni di tipo finite* $(S, S', \ldots)$, dette brevemente *tipi*, su di un dato $\mathbf{A}, \mathbf{B}, \mathbf{N}$, ottenuto in accordo con la seguente regola sintattica:

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

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

Dato un dato sistema di tipi $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, uno *schema* σ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ è una funzione totale da $\mathbf{N}$ a $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, che associa ai nomi di tipi la loro descrizione. Diremo che un nome di tipo N *eredita* da un altro nome di tipo N', denotato con $N \prec\!\!\!\mid_\sigma N'$, se $\sigma(N) = N' \sqcap S$. Si richiede che la relazione di ereditarietà sia priva di cicli, i.e., la chiusura transitiva di $\prec\!\!\!\mid_\sigma$, 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 σ è uno schema e $\mathbf{R}$ un insieme di regole. Ad esempio, il dominio Magazzino rappresentato in Tabella 3.1 ed esteso in 3.2 con l'aggiunta di alcune regole di integrità, è descritto in OCDL tramite lo schema con regole mostrato in Tabella 3.3.

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

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

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

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

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

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

Tabella 3.3: Schema con Regole di Integrità in linguaggio OCDL

L'interpretazione è estesa agli altri tipi come segue:

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

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 $(\triangle.\texttt{stock}.\forall.\texttt{item: SMaterial})$ è equivalente a $\triangle[\texttt{stock: }\forall\{[\texttt{item: SMaterial}]\}]$.

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 $C \in \mathbf{C}, V \in \mathbf{V}, T \in \mathbf{T}, R \in \mathbf{R}$: $\mathcal{I}[C] \subseteq \mathcal{I}[\sigma(C)]$, $\mathcal{I}[T] = \mathcal{I}[\sigma(T)]$, $\mathcal{I}[V] = \mathcal{I}[\sigma(V)]$, $\mathcal{I}[S^a] \subseteq \mathcal{I}[S^c]$.*

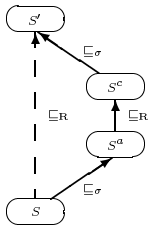
Si noti come, in una istanza legale $\mathcal{I}$, l'interpretazione di un nome di classe base (C) *è contenuta* nell'interpretazione della sua descrizione, mentre per un nome di classe virtuale (V), come per un nome di tipo-valore (T), l'interpretazione *coincide* con l'interpretazione della sua descrizione. 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 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 `ocdl` viene selezionata come istanza legale quella *più grande*, cioè viene adottata una semantica di *massimo punto fisso*. Le definizioni formali di tale semantica e le motivazioni della scelta sono discusse in [29]; in questa sede si vuole soltanto evidenziare il fatto che il modello ammette delle classi virtuali cicliche e quindi il metodo di ottimizzazione proposto è applicabile anche alle query cicliche o ricorsive [36].

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.



$S =$ `Manager` $\sqcap (\triangle.\texttt{level}: 5 \div 8)$
$S' = (\triangle.\texttt{salary}: 30 \div 70)$
$R_1 =$ `Manager` $\sqcap (\triangle.\texttt{level}: 5 \div 10) \rightarrow (\triangle.\texttt{salary}: 40 \div 60)$

$S \sqsubseteq_{\mathrm{R}} S', \quad S \not\sqsubseteq_\sigma S'$

Figura 3.1: Relazione di sussunzione dovuta alle regole

È importante notare che la relazione di sussunzione rispetto al solo schema σ, cioè considerando $\mathbf{R} = \emptyset$, denotata con $\sqsubseteq_\sigma$, è simile alle relazioni di *subtyping* o *refinement* tra tipi definite nei *CODMs* [37, 38]. Questa relazione può essere calcolata attraverso una comparazione sintattica sui tipi; per il nostro modello tale l'algoritmo è stato presentato in [29].

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 3.1, 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 $EXP(S) \sqsubseteq_\sigma S'$.*

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 [29].

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 $, : \mathbf{S} \longrightarrow \mathbf{S}$, come segue:

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

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

Esempio applicativo

Riprendiamo l'esempio descritto all'inizio di questo capitolo (in Sezione 3.1, e vediamo a quali vantaggi pratici può portare l'uso della logica `ocdl`.
L'attività di inferenza iniziale da parte del sistema, effettuata da OCDL-Designer, consiste nel determinare l'espansione semantica di ogni classe dello schema. Questo permette di rilevare relazioni *isa* non esplicitate nella definizione dell classi stesse. Ad esempio, nel calcolo dell'espansione semantica della classe `DMaterial` si rileva che tale classe è sussunta (implica) l'antecedente della regola R1: congiungendone quindi il conseguente si determina che `DMaterial` è una specializzazione di `SMaterial`. Nel caso in cui vengano applicate più regole durante l'espansione semantica di una classe, le relazioni *isa* ricavate sono meno ovvie di quella appena descritta. Ad esempio, nel calcolo dell'espansione semantica della classe `DStorage` viene applicata prima la regola R1 e successivamente la regola R2, concludendo che `DStorage` è una specializzazione di `SStorage`.

L'altra componente del sistema, ODB-QOptimizer, si occupa invece dell'ottimizzazione semantica delle interrogazioni a run-time, anch'essa basata sull'espansione semantica, in modo da specializzare le classi interessate dall'interrogazione. Prendiamo ad esempio la seguente query, espressa in *OQL*, che vuole selezionare dallo schema tutti i magazzini che contengono materiali che hanno tutti rischio maggiore o uguale a 15.

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

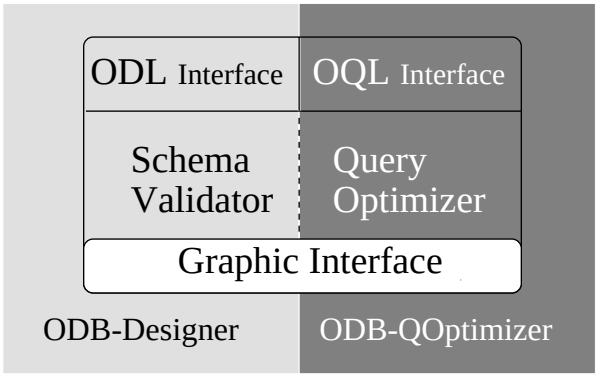



Figura 3.2: ODB-Tools

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

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

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

3.1.3 Architettura degli ODB-Tools

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

- *validazione di schemi* (ODB-Designer): l'utente può inserire la descrizione di uno schema di una base di dati, usando il linguaggio ODL, e il sistema realizza automaticamente la validazione dello schema (ve-

rificando che non esistano classi incoerenti, ovvero che non possono essere popolate da alcun oggetto) e la sua riclassificazione (determinando eventuali relazioni di specializzazione non esplicitate dallo schema stesso). Al suo interno è quindi presente un'interfaccia tra ODL_{I^3} e `ocdl`;

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

Considerazioni Un' importante osservazione riguarda la **versatilità** degli ODB-Tools: da quanto detto in questa sezione potrebbe infatti sembrare che il sistema, alla cui base sono delle complesse teorie e ricerche nel campo della logica, sia unicamente finalizzato alla validazione degli schemi ad oggetti ed all' ottimizzazione semantica delle query. In realtà, grazie proprio al completo impianto teorico alla base di questi strumenti, ODB-Tools può essere impiegato per realizzare un' infinità di altre funzioni, con l' unico limite che la conoscenza da elaborare deve essere resa esprimibile e tradotta in `ocdl`.

3.2 Fusione di gerarchie di ereditarietà

Questa tecnica è illustrata in [6] ed è stata utilizzata per tradurre la conoscenza sia estensionale che intensionale, necessaria per le fasi di integrazione e query processing, in modo da essere contemporaneamente trattabili dagli ODB-Tools in modo omogeneo, e poter conseguentemente usufruire delle tecniche di inferenza rese disponibili. Facciamo un breve esempio chiarificatore e descriviamo la metodologia ideata in [6], lasciando al Capitolo 5 la spiegazione di come viene impiegata questa tecnica per completare l' integrazione delle informazioni in MOMIS.

3.2.1 Esempio delle Biblioteche

Supponiamo di avere due sorgenti di informazioni su articoli, libri e riviste (Libreria e Progetti), e che esse non presentino conflitti intensionali relativi ai nomi ed ai tipi degli attributi e siano caratterizzate dalle gerarchie nella Figura 3.3, dove Libreria è quella di sinistra. Supponiamo che le classi delle sorgenti abbiano estensioni parzialmente sovrapposte, come raffigurato in

Figura 3.4, cioè contengano informazioni in parte sugli stessi articoli, libri, riviste, etc... .

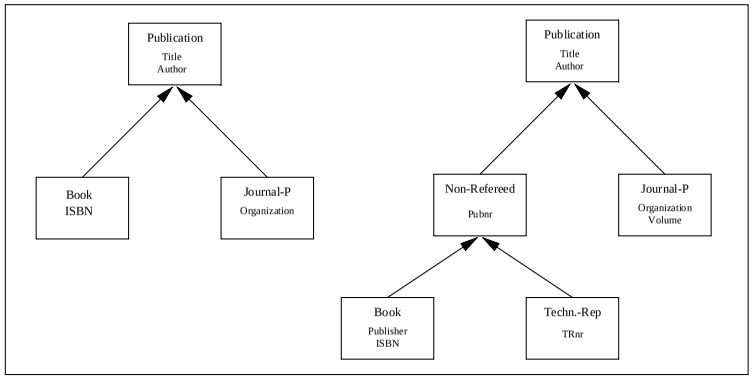


Figura 3.3: Schema delle Gerarchie Libreria e Progetti

La fusione delle due gerarchie genera la Gerarchia Integrata di Figura 3.5.

Ogni classe *fittizia* della gerarchia integrata è caratterizzata da una certa intensione ed estensione ottenute dalle informazioni sulle intensioni e sulle sovrapposizioni delle estensioni delle due gerarchie. L' estensione di ogni nuova classe è costituita dagli oggetti delle sorgenti che posseggono tutti gli attributi che caratterizzano l' intensione della stessa. Alcune di queste classi (come C1) hanno in realtà un' estensione non propria, ma determinata dall' unione insiemistica delle estensioni delle sottoclassi (perchè non esistono oggetti che abbiano solo gli attributi che caratterizzano l' intensione di quella particolare classe); altre (come C8) non hanno attributi propri ed ereditano solo quelli delle superclassi dirette, però hanno un' estensione propria caratterizzata dagli oggetti che rappresentano l' intersezione delle superclassi. Queste due tipologie di classi possono essere eliminate o mantenute dallo schema in base a considerazioni sulla comprensibilità dello schema e sulla capacità del sistema di gestire oggetti in più di una classe (gli oggetti memorizzati in C8, se questa fosse eliminata, sarebbero sia in C4 che in C6, conserviamo C8 se il sistema non gestisce il caso di avere uno stesso oggetto in più classi).

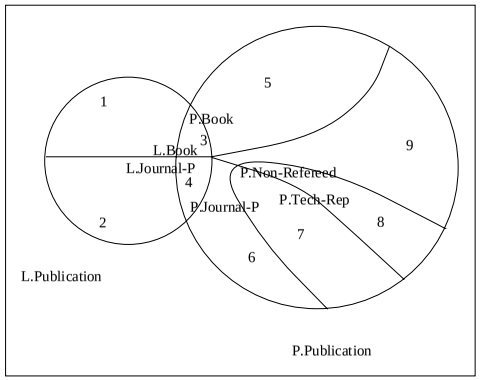


Figura 3.4: Sovrapposizione delle Estensioni delle Sorgenti Libreria e Progetti

3.2.2 Metodologia per la fusione delle gerarchie

La tecnica per ottenere questo risultato applica la teoria detta *formal context analysis* [39]. Date le gerarchie iniziali, in cui si immaginano già risolti i conflitti intensionali, i passi per ottenere uno schema integrato come quello dell' esempio precedente sono i seguenti:

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

  Le base extension costituiscono una partizione dell' insieme complessivo di tutti gli oggetti rappresentati dalle sorgenti: sono sottoinsiemi disgiunti delle estensioni delle classi e sono ottenute dall' intersezione delle stesse (vedi Figura 3.4).

- espressione delle sovrapposizioni intensionali

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

- generazione del *context*

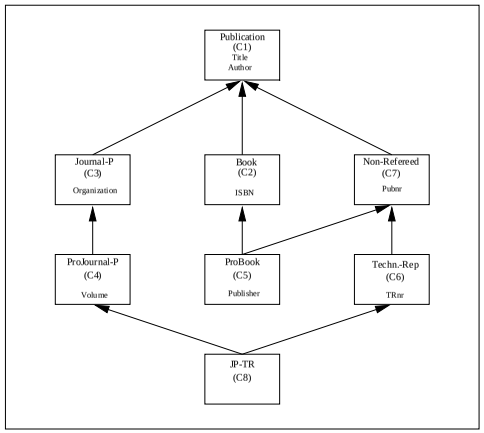


Figura 3.5: Gerarchia Integrata Libreria-Progetti

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

- creazione del lattice rappresentante la gerarchia

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

Si descrive ogni passo con maggiore dettaglio.

1. La prima informazione di cui occorre disporre è la relazione di sovrapposizione fra le estensioni delle classi dei sorgenti (Extensional overlapping, vedi Tabella 3.4), fondamentale alla creazione della gerarchia.

Definizione 4 (Base Extension) *Una base extension è un sottoinsieme dell' insieme complessivo delle estensioni ed identifica oggetti realmente esistenti in una o più sorgenti.*

La partizione dell' insieme complessivo delle estensioni, costituita da tutte le base extension, viene rappresentata da una matrice le cui righe indicano

Base Extension	1	2	3	4	5	6	7	8	9
L.Publication	x	x	x	x					
L.Book	x		x						
L.Journal-P		x		x					
P.Publication			x	x	x	x	x	x	x
P.Journal-P				x		x	x		
P.Non-Refereed			x		x		x	x	x
P.Book			x		x				
P.Tech-Rep							x	x	

Tabella 3.4: Extensional overlapping

Attribute	Title	Author	ISBN	Organ	Vol	Publsh	TRnr	Pubnr
L.Publication	x	x						
L.Book	x	x	x					
L.Journal-P	x	x		x				
P.Publication	x	x						
P.Journal-P	x	x		x	x			
P.Non-Refereed	x	x						x
P.Book	x	x	x			x		x
P.Tech-Rep	x	x					x	x

Tabella 3.5: Intensional overlapping

le classi sorgenti e le colonne le base extension. Leggendo la matrice per colonne si ha un segno in corrispondenza di ogni riga la cui classe comprenda nella propria estensione la base extension (quindi una base extension rappresenta l' insieme intersezione delle estensioni di tutte le classi che sono state contraddistinte da un segno nella casella corrispondente della matrice). Leggendo per righe si deduce invece che l' estensione di una sorgente è data dall' unione delle base extension corrispondenti alle colonne per cui esiste una casella marcata.

2. Analogamente si esprime in una matrice anche il legame fra gli attributi e le classi sorgenti (Intensional overlapping, in Tabella 3.5): per righe si hanno le classi, per colonne gli attributi: le caselle marcate indicano che un certo attributo è una proprietà della classe corrispondente.

3. I passi precedenti preparano alla individuazione degli attributi propri degli oggetti nelle base extension (Extension-Attribute Relation, Tabella 3.6). Si crea una matrice che ha per righe gli attributi e per colonne le base extension: le caselle marcate sono ricavate sulla base delle precedenti tabelle: per ogni base extension (colonna) sono marcate le caselle corrispondenti agli attributi che caratterizzano le proprietà degli oggetti che descrive. Per determinare gli attributi da marcare in corrispondenza di una certa colonna (base extension) si considera la Tabella 3.4 e si scelgono le classi corrispondenti a delle caselle marcate, cioè la cui estensione includa la base extension, poi dalla Tabella 3.5 si prende l' unione di tutti gli attributi che caratterizzano le classi scelte.

Attrib-Exten	1	2	3	4	5	6	7	8	9
Title	x	x	x	x	x	x	x	x	x
Author	x	x	x	x	x	x	x	x	x
ISBN	x		x		x				
Organization		x		x		x	x		
Volume				x		x	x		
Publisher			x		x				
TRnr							x	x	
Pubnr			x		x		x	x	x

Tabella 3.6: Extension-Attribute Relation

Un *context* è individuato da una terna (G, M, I) dove G è un insieme di oggetti, M è un insieme di attributi, I è l' insieme di coppie (oggetto g, attributo m) tali che g possiede l' attributo m. Nel nostro esempio G è l' insieme delle base extension, M degli attributi, I l' insieme delle coppie (g, m) evidenziate dalla Tabella 3.6 Extension-Attribute Relation.

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

- definizione degli insiemi *Int* ed *Ext* (in Tabella 3.7)

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

Int é l' insieme di tutti i gruppi di attributi che caratterizzano ogni base extension. Ogni elemento dell' insieme corrisponde ad una colonna della Tabella 3.6.

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

Ext é l' insieme di tutti i gruppi di base extension che sono caratterizzate da ogni attributo. Ogni riga della Tabella 3.6 è tradotta nell' insieme di base extension che posseggono quell' attributo.

- definizione degli insiemi *ConI* e *ConE* (in Tabella 3.8)

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

$$Int := \left\{ \begin{array}{l} \{Title, Author, ISBN\}, \\ \{Title, Author, Organization\}, \\ \{Title, Author, ISBN, Publisher, Pubnr\}, \\ \{Title, Author, Organization, Volume\}, \\ \{Title, Author, Organization, Volume, TRnr, Pubnr\}, \\ \{Title, Author, TRnr, Pubnr\}, \\ \{Title, Author, Pubnr\}, \end{array} \right\}$$

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

Tabella 3.7: Insiemi Int ed Ext

Ad ogni elemento di *Int* (costituito da un insieme di attributi) si associa l' insieme delle base extension che posseggono almeno tutti gli attributi dell' insieme considerato[6].

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

Ad ogni insieme di estensioni di *Ext* si associa l' insieme degli attributi che caratterizzano almeno tutte le base extension dell' insieme considerato[7].

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

$$Con := ConI \cup ConE$$

La classe *Con*, in Tabella 3.9, si ottiene dall' unione di *ConI* e *ConE*. Ognuna delle sue coppie rappresenta una nuova classe le cui estensione ed intensione sono rappresentate rispettivamente dall' insieme delle base extension e degli attributi che costituiscono la coppia in esame.

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

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

$$ConI := \left\{ \begin{array}{l} (\{Title, Author, ISBN\}, \{1,3,5\}), \\ (\{Title, Author, Organization\}, \{2,4,6,7\}), \\ (\{Title, Author, ISBN, Publisher, Pubnr\}, \{3,5\}), \\ (\{Title, Author, Organization, Volume\}, \{4,6,7\}), \\ (\{Title, Author, Organization, Volume, TRnr, Pubnr\}, \{7\}), \\ (\{Title, Author, TRnr, Pubnr\}, \{7,8\}), \\ (\{Title, Author, Pubnr\}, \{3,5,7,8,9\}) \end{array} \right\}$$

$$ConE := \left\{ \begin{array}{l} (\{Title, Author\}, \{1,2,3,4,5,6,7,8,9\}), \\ (\{Title, Author, ISBN\}, \{1,3,5\}), \\ (\{Title, Author, Organization\}, \{2,4,6,7\}), \\ (\{Title, Author, ISBN, Publisher, Pubnr\}, \{3,5\}), \\ (\{Title, Author, Organization, Volume\}, \{4,6,7\}), \\ (\{Title, Author, TRnr, Pubnr\}, \{7,8\}), \\ (\{Title, Author, Pubnr\}, \{3,5,7,8,9\}) \end{array} \right\}$$

Tabella 3.8: Insiemi ConI e ConE

$$Con := \left\{ \begin{array}{l} C1 := (\{Title, Author\}, \{1,2,3,4,5,6,7,8,9\}), \\ C2 := (\{Title, Author, ISBN\}, \{1,3,5\}), \\ C3 := (\{Title, Author, Organization\}, \{2,4,6,7\}), \\ C4 := (\{Title, Author, Organization, Volume\}, \{4,6,7\}), \\ C5 := (\{Title, Author, ISBN, Publisher, Pubnr\}, \{3,5\}), \\ C6 := (\{Title, Author, TRnr, Pubnr\}, \{7,8\}), \\ C7 := (\{Title, Author, Pubnr\}, \{3,5,7,8,9\}), \\ C8 := (\{Title, Author, Organization, Volume, TRnr, Pubnr\}, \{7\}) \end{array} \right\}$$

Tabella 3.9: Insiemi Con delle Classi Fittizie

- calcolo delle relazioni di specializzazione fra le classi

  Le relazioni di specializzazione sono memorizzate in una matrice quadrata M avente per righe e colonne le nuove classi. Un 1 indica che la classe della riga specializza quella della colonna (in Tabella 3.10).

$\prec$	C1	C2	C3	C4	C5	C6	C7	C8
C1	0	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0
C3	1	0	0	0	0	0	0	0
C4	1	0	1	0	0	0	0	0
C5	1	1	0	0	0	0	1	0
C6	1	0	0	0	0	0	1	0
C7	1	0	0	0	0	0	0	0
C8	1	0	1	1	0	1	1	0

Tabella 3.10: Matrice M di Specializzazione

- rimozione delle relazioni di specializzazione transitive

  Nella matrice M sono specificate tutte le specializzazioni comprese quelle transitive (se A specializza B e B specializza C allora ci sarà un 1 ad indicare anche che A specializza C), ma per ottenere la gerarchia a noi interessano solo quelle fra subclasse e classe specializzata (A specializza B e B specializza C). Per ottenere queste relazioni si calcola la matrice di Tabella 3.11 definita da: N=M-MxM: solo gli 1 indicano relazioni dirette di specializzazione.

$\prec$	C1	C2	C3	C4	C5	C6	C7	C8
C1	0	0	0	0	0	0	0	0
C2	**1**	0	0	0	0	0	0	0
C3	**1**	0	0	0	0	0	0	0
C4	0	0	**1**	0	0	0	0	0
C5	-1	**1**	0	0	0	0	**1**	0
C6	0	0	0	0	0	0	**1**	0
C7	**1**	0	0	0	0	0	0	0
C8	-3	0	0	**1**	0	**1**	0	0

Tabella 3.11: Matrice N di Specializzazione non Transitiva

5. Dalla matrice N si ricavano le relazioni di specializzazione fra le classi e conseguentemente la gerarchia di Figura 3.5. Se necessario, si procede all' eventuale raffinamento della gerarchia ottenuta eliminando le classi con intensione od estensione nulla (C8, C1).

3.2.3 Considerazioni

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

In generale si usa la conoscenza estensionale per creare una partizione dell' insieme complessivo delle entità[8] rappresentate nelle sorgenti. Ogni sottoinsieme (base extension) costituente la partizione contiene entità distinte non contenute dagli altri sottoinsiemi. Le proprietà delle entità in ogni partizione non sono uniche per ogni suo elemento, ma sono definite da un insieme massimo di attributi (Attribute-Extension Relation) ricavati dall' unione di tutti gli attributi delle classi la cui intersezione forma la base extension: questo consente di considerare come oggetti della estensione base sia un oggetto di una delle classi che la costituiscono, sia un oggetto virtuale ottenuto fondendo oggetti provenienti da due o più classi che partecipino alla base extension. Una volta ottenute queste informazioni, per creare le classi della gerarchia, si devono considerare gli insiemi di attributi propri di ogni gruppo di oggetti: si forma una classe per ogni insieme di attributi cui corrisponda una base extension. Nel caso pessimo si crea un numero massimo di classi pari alla somma del numero degli attributi e delle base extension. Questo caso è causato da una forte disomogeneità intensionale ed estensionale fra le sorgenti ed è abbastanza improbabile inquanto l' omogeneità intensionale è assicurata dal metodo complessivo di integrazione di MOMIS, svolto in [3, 4, 2], che verrà illustrato nel prossimo Capitolo.

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

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

deve essere confrontato contro ogni colonna per vedere se la base extension corrispondente è caratterizzata dalle proprietà in esame: questo confronto implica di esaminare gli attributi di ogni riga per cui l' insieme è computato dopo al massimo $O(n^3)$ confronti. Idem per *ConE*.

Capitolo 4

MOMIS: Integrazione Intensionale

Lo stadio di sviluppo del sistema MOMIS preesistente consentiva la risoluzione dei conflitti lessicali, ontologici ed intensionali degli schemi delle sorgenti, e la generazione della *vista* globale, completando il processo di **integrazione degli schemi**. Viceversa, con questo lavoro, si è approfondita la necessità di esprimere e manipolare la conoscenza estensionale, col proposito di progettare un Query Manager in grado di adempiere al processo di **integrazione delle informazioni**.

In questo Capitolo si illustra la soluzione proposta finalizzata all' integrazione di schemi di sorgenti autonome. Si evidenzia inoltre come l' approccio virtuale, e non materializzato, richieda di introdurre un' ulteriore fase di integrazione (che sarà approfondita nei capitoli successivi) per completare il processo di fusione e sintesi delle informazioni.

4.1 Approccio Semantico vs. Approccio Strutturale

Nel Capitolo 2 è stato descritto come siano stati già realizzati diversi sistemi diretti all'integrazione di database convenzionali, ed in tempi molto più recenti siano anche stati proposti sistemi finalizzati all'integrazione di dati semi-strutturati, caratteristici specialmente dalle pagine HTML. Questi approcci sono stati studiati e classificati in [30], distinguendoli in *semantici* e *strutturali*. Per quanto riguarda l'approccio *strutturale* (come ad esempio TSIMMIS, descritto in Sezione 2.1) si possono sottolineare i seguenti punti caratterizzanti:

- un modello comune dei dati del tipo *self-describing* è utilizzato per descrivere tutte le istanze del sistema integrato, rendendo inutili gli schemi concettuali e logici delle diverse sorgenti, ma facilitando l' integrazione anche e soprattutto di dati semi-strutturati;
- il modello comune non è standard: è debole rispetto ai modelli object-oriented standard;
- le informazioni semantiche sono inserite manualmente da un operatore attraverso l'utilizzo di linguaggi descrittivi derivati dalla logica del primo ordine (ed in particolare, in TSIMMIS, attraverso le MSL rule);
- l' assenza di uno schema globale non permette una ottimizzazione semantica delle interrogazioni, particolarmente utile nel caso di database di grandi dimensioni;
- l' intero onere dell' integrazione è a carico del progettista del mediatore, questi deve conoscere gli schemi delle sorgenti e del Mediatore stesso.

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

- essendo sviluppato specificatamente per sorgenti strutturate, dispone per ognuna di esse dei rispettivi metadati, codificati nello schema concettuale;
- insieme allo schema concettuale sono presenti le informazioni semantiche (vincoli, rule, ...), esse possono essere utilmente sfruttate sia nella fase di integrazione delle sorgenti, sia in quella di ottimizzazione delle interrogazioni;
- il modello comune dei dati è conforme allo standard object data model
- è realizzata una unificazione ragionata (parziale o totale) degli schemi (in modo semi_automatico), per arrivare alla definizione di uno schema globale.

In particolare per MOMIS si è stabilito di ricorrere, per la rappresentazione dello schema globale, all' adozione del modello ad oggetti standard ODMG93, esteso per le problematiche di integrazione: i vantaggi comportati dall' utilizzo dell' approccio semantico, sostenuto da un modello ad oggetti sono numerosi, in particolare si ricorda:

- un ausilio molto forte al progettista del Mediatore in quanto l' integrazione può essere semi_automatizzata,

- la possibilità per l' utente di formulare qualsiasi interrogazione che sia consistente con lo schema,
- la capacità di realizzare un' eventuale ottimizzazione delle interrogazioni, usufruendo delle informazioni semantiche comprese nello schema;
- la riorganizzazione delle conoscenze estensionali realizzabile attraverso l'uso delle primitive di generalizzazione e di aggregazione tipiche dei modelli ad oggetti;
- l'adozione di una semantica di mondo aperto, adottata nella logica descrittiva `ocdl`, permette inoltre di rendere l' ambiente utilizzabile anche con dati semi_strutturati: gli oggetti di una classe condividono una struttura minima comune (che è quindi la descrizione della classe stessa), ma possono avere ulteriori proprietà non esplicitamente comprese nella struttura della classe di appartenenza.
- gli ampi sforzi devoluti in ambito internazionale per lo sviluppo di standard rivolti agli oggetti: CORBA [40] per lo scambio di oggetti attraverso sistemi distribuiti; ODMG-93 [26] (e con esso i modelli ODM e ODL per la descrizione degli schemi, e OQL come linguaggio di interrogazione) per lo sviluppo di applicazioni OODBMS, etc... ;

Sono proposti in MOMIS il modello di dati comune e il linguaggio di definizione comune (ODL_{I^3}) per descrivere gli schemi delle sorgenti e del Mediatore.
Come verrà più ampiamente descritto nella Sezione 4.3, ODL_{I^3} costituisce un'estensione del corrispondente linguaggio di definizione ODL proposto dal gruppo di standardizzazione ODMG-93. Inoltre, la logica descrittiva `ocdl` (vedi Sezione 3.1.2) è utilizzata come linguaggio *kernel* del sistema, sia per automatizzare la fase di integrazione intensionale ma soprattutto **estensionale**, sia per la fase di ottimizzazione delle interrogazioni, attraverso l'ausilio degli ODB-Tools.

Il processo di integrazione si svolge in tre distinti momenti:

1. **Unificazione degli schemi:** è la fase analizzata in [3] ed in questo Capitolo. L'uso della logica descrittiva `ocdl`, insieme alle tecniche di clustering riportate in [41], permettono la realizzazione di una fase semi-automatica di integrazione degli schemi, fino a pervenire alla definizione dello *Schema Globale*, direttamente interrogabile dall'utente, che rappresenti l'unione di tutti gli schemi locali, rimuovendone incongruenze e ridondanze;

2. **Fusione delle istanze:** questa fase sarà approfondita nel Capitolo 5. L' originalità del contributo dato si manifesta nell' introduzione degli *Assiomi estensionali* e nel loro trattamento per generare gli oggetti *fusi* dalla sintesi di quelli reali provenienti dalle sorgenti. Il progettista può introdurre conoscenza sulle relazioni fra le estensioni di classi simili in sorgenti distinte e MOMIS le utilizzerà in fase di esecuzione delle interrogazioni. La buona realizzazione di questa fase si manifesta a run time perchè influisce in maniera determinante non solo sulla correttezza e sulla completezza, ma anche sulla sinteticità e sulla velocità con cui sono generate le risposte ad una query.
3. **Query Processing:** è la fase che, a partire da una interrogazione dell' utente posta sullo Schema Globale, porta alla scelta e definizione dell' insieme ottimo di interrogazioni da inviare alle sorgenti potenzialmente interessate, nonché alla presentazione di un' unica risposta. Nel Capitolo 6 si proporrà una soluzione efficace, basata sull' utilizzo di ODB-Tools e della conoscenza semantica ed estensionale, per l' otimizzazione di query in ambienti distribuiti ed autonomi, per la fase di generazione *automatica* delle interrogazioni locali e per quella di unificazione e fusione dei dati da esse estratti.

4.2 Architettura

È stata mantenuta la struttura classica di un sistema I^3. In Figura 4.1 è mostrata l'architettura del sistema MOMIS: i livelli raffigurativi sono tre:

1. **Wrapper:** si posizionano a livello delle sorgenti costituendone l'interfaccia col Mediatore vero e proprio, perciò devono essere appositamente sviluppati per il tipo di sorgente che andranno a servire. Le sorgenti sono le fonti di informazioni da integrare: possono essere dei database tradizionali (sia ad oggetti, sia basati sul modello relazionale), oppure dei semplici file system. A differenza di altri progetti analoghi, in questa prima fase del progetto, non si è ipotizzato di avere a che fare con dati di tipo multimediale o semi-strutturati. La funzione è dei Wrapper è duplice:
   - in fase di integrazione, forniscono, consultando il catalogo della sorgente, la descrizione degli schemi nel linguaggio ODL_{I^3} (descritto in Sezione 4.3);
   - in fase di query processing, devono tradurre la query ricevuta, espressa quindi nel linguaggio comune di interrogazione OQL_{I^3},

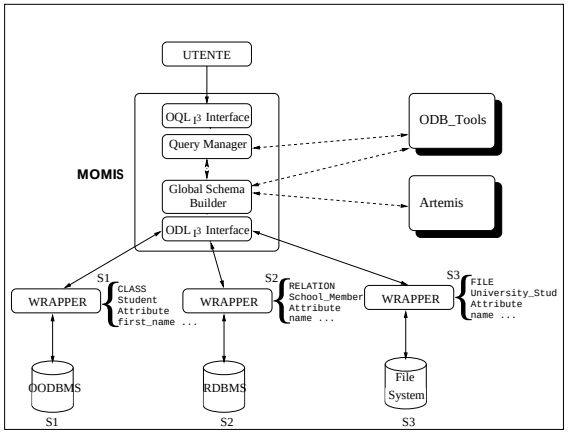


Figura 4.1: Architettura di MOMIS

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 diversi sottomoduli che si occupano dei diversi stadi del processo di integrazione descritto nel paragrafo precedente:

   - Global Schema Builder: è il modulo di integrazione degli schemi locali e di manipolazione delle conoscenze estensionali. Partendo dalle descrizioni delle sorgenti, espresse in ODL_{I^3}, genera un unico schema globale da presentare all' utente, inoltre traduce la conoscenza estensionale in una tassonomia. Questa fase di integrazione, realizzata in modo semi-automatico dall' interazione del progettista del sistema, fa uso degli ODB-Tools, delle tecniche di clustering e di quelle di fusione delle gerarchie;
   - Query Manager: è il modulo di gestione delle interrogazioni. Provvede a gestire la query dell'utente, deducendo, in base alle conoscenze intensionali ed estensionali generate dal Global Schema Builder, un insieme di sottoquery da spedire alle fonti locali, e

a *ricomporre* le informazioni da esse ricevute. Tra i suoi compiti si evidenzia l' ottimizzazione semantica delle interrogazioni, realizzata, come la fase di *Query Decomposition*, utilizzando gli ODB-Tools.

   - ODL_{I^3} interface: è l'interfaccia che serve al Mediatore per comunicare con i Wrapper in maniera uniforme, utilizzando il modello comune.
   - OQL_{I^3} interface: è l'interfaccia che serve all' utente per interrogare il Mediatore essendo consistente con ODL_{I^3}. Essendo una riduzione di OQL non genera alcuna complicazione nè per l' utente nè per il sistema.

3. **Utente:** è l'utilizzatore del sistema, colui che interagisce con il Mediatore. A lui viene presentato lo schema globale, all' interno del quale sono rappresentati tutti i concetti interrogabili: è come se si trovasse di fronte ad un unico database ad oggetti (o relazionale), da interrogare in modo tradizionale, mentre le sorgenti locali rimangono completamente trasparenti.

Lo scopo che ci si è preposti con il Progetto MOMIS è la realizzazione di un sistema di mediazione che, a differenza di molti altri progetti analizzati, contribuisca a fornire un' integrazione delle sorgenti che faciliti l' interrogazione per gli utenti, senza comprometterne la libertà od inficiare la completezza e la sinteticità della risposta.

Per realizzare l' integrazione degli schemi, son state arricchite con un componente *intelligente* le tecniche di integrazione basate sul clustering, ampliandole con fasi automatiche realizzate sfruttando gli ODB-Tools, in modo da diminuire il più possibile l' apporto manuale del progettista del sistema. Sono state inoltre rifinite le tecniche di clustering, al fine di ampliare quella che prima era una semplice analisi terminologica di nomi di classi e attributi, trasformandola in una vera e propria analisi sia dei nomi che dei domini dei tipi che devono essere integrati.

Per completare il processo di integrazione con la conoscenza estensionale si è riuscito a manipolare gli assiomi con ODB-Tools, in modo da verificarne la reciproca congruenza e dedurne le relazioni implicite. Questo consente di disporre della conoscenza necessaria a decomporre intelligentemente le query.

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

1. *Generazione del Thesaurus di relazioni terminologiche:* attraverso l' interazione col progettista, che pone relazioni terminologiche e l' utilizzo

di ODB-Tools, si cerca di derivare in maniera semi_automatica nuove relazioni di sinonimia e corrispondenza tra i termini (nomi di classi e di attributi) delle diverse sorgenti. Questo passo è realizzato dal modulo SIM_1.

2. *Analisi delle affinità **intensionali** fra le classi:* questa fase è realizzata considerando le relazioni terminologiche memorizzate nel Thesaurus ed i *coefficienti di affinità* in [41]. Questo passo, ed il successivo *3.*, è realizzato dal componente ARTEMIS, sviluppato presso l' Università di Milano.

3. *Creazione dei Cluster*, cioè di raggruppamenti di classi affini: si tratta di classi intensionalmente affini per le quali si presume esista anche una qualche sovrapposizione fra le estensioni.

4. *Generazione dello Schema Globale del Mediatore:* da ogni Cluster si definisce una Classe Globale la cui estensione è costituita dall' unione delle estensioni delle classi sorgenti che costituiscono il cluster, mentre l' intensione è ricavata dall' unione "ragionata" degli attributi delle stesse. Questa fase porta alla definizione ODL_{I^3} delle Classi Globali con le regole di mapping fra gli attributi dalla Casse Globale alle classi sorgenti

Il processo così descritto è stato presentato in [3, 4, 2]. Il contributo di questa tesi è stato diretto alle fasi di trattamento della conoscenza estensionale, che saranno più dettagliatamente descritte nel Capitolo 5, e che estendono il punto *4.* attraverso:

1. *Arricchimento del Cluster:* il progettista arricchisce la descrizione ottenuta di ogni Cluster ponendo nuovi assiomi estensionali intra ed inter classi sorgente. Questa fase porta a considerare il caso di modificarne alcuni, qualora siano riconosciuti particolari legami estensionali fra classi in Cluster diversi. Questo passo si ripete fintanto che il progettista non ha ottenuto un insieme di cluster soddisfacenti, da cui generare lo Schema Globale.

2. *Fusione della Gerarchia Virtuale:* una volta definite le Classi Globali in modo definitivo, ODB-Tools ne verifica la congruenza e porta alla deduzione degli assiomi impliciti. Si applica ad ogni Classe Globale l' algoritmo di *individuazione delle Base extension* e di *fusione delle gerarchie* [6]. Quest' ultimo costruisce una gerarchia composta da nuove classi. L' intensione di queste è un sottoinsieme delle proprietà del cluster, mentre l' estensione è costituita (virtualmente) da tutti gli

oggetti reali e/o fusi provenienti dalle sorgenti che posseggono almeno le proprietà specificate nell' intensione della classe.

Una volta ottenuto lo schema globale, in maniera semiautomatica, operazione da svolgersi solo in fase di attivazione del sistema, o di acquisizione di una nuova sorgente da integrare, il Mediatore è pronto per essere interrogato: l'utente può porre su questo schema globale ottenuto delle query, senza sapere in quale sorgente particolare andare a recuperare le informazioni desiderate. Inoltre, l' utente sarà pure esonerato dal dover conoscere i diversi linguaggi di interrogazione locali delle fonti integrate: è il compito del Query Manager ottimizzare le query ricevute e spedirle alle sorgenti in fase di Query Reformulation.

Nel seguito di questo capitolo saranno approfonditamente analizzati tutti i passi del processo che porta alla costruzione dello Schema Globale, così come presentati in [2, 3, 4].

4.2.1 Esempio: Integrazione delle sorgenti dell' Università

Questo esempio verrà utilizzato nel seguito per meglio chiarire tutti i passaggi che vengono effettuati sia nella fase di integrazione che in quella di query processing e fa riferimento alle definizioni degli schemi delle sorgenti presentati nell' Appendice C in ODL_{I^3}. Per semplicità esso è rappresentato anche in modo schematico in Figura 4.2.

L'esempio si riferisce ad una realtà universitaria: le sorgenti da integrare sono tre. La prima sorgente, `University` (S_1), è un database di tipo relazionale e contiene informazioni sui dipendenti e sugli studenti di una determinata università. È composta da sei tabelle: `University_Worker`, `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Per ogni professore (presente nella tabella `Research_Staff`), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key `dept_code`), sul suo indirizzo di posta elettronica (`email`), e sul corso da lui tenuto (`section_code`). Per il corso, viene memorizzata pure l'aula (`Room`) dove questo si svolge, mentre del dipartimento sono descritti, oltre al nome (`dept_name`) ed al codice (`dept_code`), il budget (`budget`) che ha a disposizione e l'area (`dept_area`) a cui appartiene, sia essa Scientifica, Economica,... Per gli studenti presenti nella tabella `School_Member` sono invece mantenuti il nome (nella coppia `first_name` e `last_name`), la facoltà di appartenenza (`faculty`) e l'anno di corso (`year`).
La sorgente `Computer_Science` (S_2) contiene invece informazioni sulle persone afferenti a questa facoltà, ed è un database ad oggetti. Sono presenti

Sorgente University (S_1)

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

Sorgente Computer_Science (S_2)

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

Sorgente Tax_Position (S_3)

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

Figura 4.2: Esempio di Riferimento: Sorgenti dell' Università

sei classi: `CS_Person`, `Professor`, `Student`, `Division`, `Location` e `Course`. I dati mantenuti sono comunque abbastanza simili a quelli della sorgente S_1: per quanto riguarda i professori, sono memorizzati il titolo (`title`), e la divisione di appartenenza (`belongs_to`), che a sua volta fa parte di un dipartimento (e ne può quindi essere considerata una specializzazione); per gli studenti sono memorizzati i corsi seguiti (`takes`) e l'anno di corso (`year`). Il corso ha poi un attributo complesso che lo lega al professore che ne è titolare (`taught_by`), mentre per la divisione si tiene l'indirizzo (`address`), i fondi (`fund`) e il numero di impiegati (`employee_nr`).
È presente inoltre una terza sorgente, `Tax_Position` (S_3), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (`tax_fee`). In questo caso (S_3), non si tratta di un database ma di un file system, che contiene quindi semplici tracciati record.

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

Il linguaggio ODL$_{I^3}$ è il linguaggio di definizione attraverso il quale i wrapper comunicano al mediatore (ed in particolare al Global Schema Builder) le descrizioni delle sorgenti da loro servite. Punto di partenza per la definizione

di questo linguaggio è stata la proposta di standardizzazione per i linguaggi di mediazione [42], risultato del lavoro di un workshop I^3. Partendo da questa proposta (secondo la quale i diversi sistemi di mediazione avrebbero potuto supportare sorgenti con modelli complessi, come quelli ad oggetti, e sorgenti molto più semplici, come file di strutture), si è cercato di discostarsi il meno possibile dal linguaggio ODL, a sua volta proposto dal gruppo ,di standardizzazione ODMG-93.
È stato quindi definito il linguaggio ODL$_{I^3}$ come una estensione allo standard ODL, per raggiungere i seguenti scopi:

- dichiarare relazioni terminologiche fra nomi di classi ed attributi:

  ⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** | ⟨relationship_dcl⟩**;** ⟨relationships_list⟩
  ⟨relationships_dcl⟩ ::= ⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
  ⟨relationship_type⟩ ::= **syn** | **bt** | **nt** | **rt**
  ...

- descrivere tutte le fonti di informazioni in maniera uniforme, come depositi di oggetti, e quindi utilizzare il concetto di *classe*, indipendentemente dal modello originale utilizzato[1]: integrare database relazionali, ad oggetti, e file system. Si illustra di seguito l' estensione ODL per esprimere il tipo della sorgente ed altri concetti propri del modello originario:

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

- supportare la dichiarazione di vincoli di integrità e di assiomi estensionali sottoforma di *if then rule*, siano esse definite sugli schemi locali (e

[1] 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,...)

magari da questi ricevute), siano esse riferite allo Schema Globale, e quindi inserite dal progettista. Vediamo quindi l' estensione di ODL per l' espressione *dichiarativa* di vincoli ed assiomi estensionali;

⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_pre⟩ **then** ⟨rule_post⟩
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨rule_identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_identifier⟩ ::= ⟨identifier⟩ | **(** ⟨identifier⟩ **and** ⟨identifier⟩ **)**
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** | ⟨rule_body⟩ |
⟨rule_body_list⟩ **and** ⟨rule_body⟩ |
⟨rule_body_list⟩ **and (** ⟨rule_body_list⟩ **)**
⟨rule_body⟩ ::= ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in** ⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
⟨rule_const_op⟩ ::= $= \mid \geq \mid \leq \mid > \mid <$
⟨rule_cast⟩ ::= **(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**⟨dotted_name⟩
⟨forall⟩ ::= **for all** | **forall**

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

⟨attr_dcl⟩ ::= **[readonly] attribute**
⟨domain_type⟩ ⟨attribute_name⟩
[⟨fixed_array_size⟩] [⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ ::= **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ ::= ⟨rule⟩ | ⟨rule⟩**,**⟨rule_list⟩
⟨rule⟩ ::= ⟨local_attr_name⟩ | **'**⟨identifier⟩**'**
⟨and_expression⟩ | ⟨or_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and** ⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **and** ⟨and_list⟩
⟨or_expression⟩ ::= **(** ⟨local_attr_name⟩ **or** ⟨or_list⟩ **)**
⟨or_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **or** ⟨or_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩**.**⟨class_name⟩**.**⟨attribute_name⟩
...

- essendo tradotto automaticamente nella logica descrittiva `ocdl`, consentire l' utilizzo di una *semantica di mondo aperto*, che permette di effettuare i controlli di consistenza e l' ottimizzazione semantica delle interrogazioni, forniti da ODB-Tools.

Le descrizioni ODL$_{I^3}$ di tutte le classi da integrare sono fornite al Mediatore dai Wrapper: da sottolineare che le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema. Non è quindi detto che lo schema locale ricevuto dal mediatore rappresenti l' intera sorgente, bensì ne descrive il sottoinsieme di informazioni che possono essere rese accessibili ad un utente esterno.
La sintassi del linguaggio ODL$_{I^3}$ è riportata, in BNF, nell' Appendice B. Mostriamo qui brevemente un tipo di descrizione di classi ODL$_{I^3}$, rimandando invece all'Appendice C per l'intera descrizione dell'esempio di riferimento di Sezione 4.2.1.

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

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

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

4.4 Processo di Integrazione Intensionale

In Figura 4.3 sono mostrati i passi, descritti in [3, 4, 2] ed arricchiti dall' introduzione delle rule estensionali, che portano alla definizione dello Schema Globale. In Figura 4.3 sono mostrati i risultati dei primi due passi, in riferimento all' esempio in Appendice C.

4.4.1 Generazione dello Schema Globale

In questa sezione viene presentato l' ultimo passo della fase di integrazione intensionale: a partire dai cluster determinati dallo step di Analisi di Affinità si definisce lo *Schema Globale* del Mediatore. Questo processo richiede l'

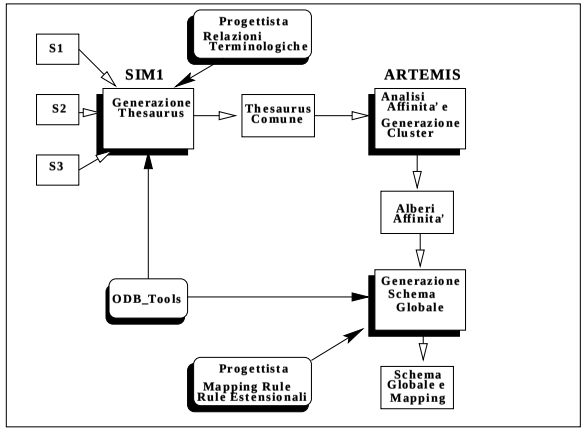


Figura 4.3: Fasi dell' Integrazione Intensionale

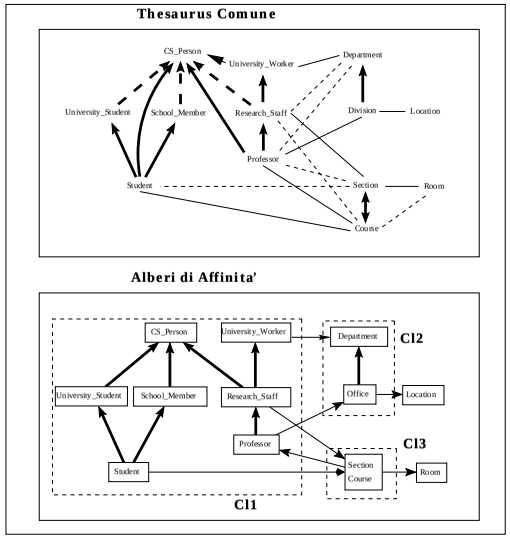


Figura 4.4: Prodotti Parziali dell' Integrazione Intensionale

iterazione col progettista: egli pone gli assiomi estensionali[2], per completare la definizione di ogni cluster, ed, infine, le Mapping rule [3, 4, 2].

La prima fase di questo processo prevede l' inserimento degli assiomi estensionali: questa fase può modificare la composizione dei cluster, per assicurare che classi con estensioni parzialmente sovrapposte non siano separate e messe in cluster differenti.

In un secondo momento, quando si è definitivamente stabilito quali classi sorgenti partecipino ad ogni cluster, viene realizzata, automaticamente per ogni cluster, una $classe\_globale_i$ rappresentativa di tutte le classi che fanno parte del cluster (ovvero una classe che costituisca una visione unificata di queste classi). Sia Cl_i un cluster determinato nella fase precedente: ad esso viene associata la $classe\_globale_i$, alle quale corrisponderà quindi un insieme di attributi globali. La fase di determinazione degli attributi è realizzabile in modo automatico, basandosi sulle relazioni tra attributi memorizzate nel Thesaurus e seguendo i seguenti criteri:

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

Esempio 1 Riferendoci al cluster Cl_1 di Figura 4.4, viene definita automaticamente la seguente classe globale:

$$\texttt{GC}_1 \;=\; (\texttt{name, rank, title, year, takes, email, pay, relation, student\_code, dept\_code, tax\_fee, section\_code, faculty})$$

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

[2]Nell' esempio in esame si supporrà che gli Assiomi estensionali definiti non richiedano di reiterare il processo di definizione dei Cluster

Oltre a questa semplice unione ragionata degli attributi, è necessaria una fase di raffinamento delle informazioni presenti nello schema globale già determinato, raffinamento che necessariamente richiederà l'intervento del progettista del sistema. In particolare, MOMIS permette agevolmente di affrontare una vasta casistica di problematiche legate all'interrogazione di un mediatore rifinendo lo schema globale con le seguenti informazioni:

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

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

  - corrispondenza in *and*: l'attributo globale corrisponde all'unione, in uno specificato ordine, degli attributi locali ad esso corrispondenti.
  - corrispondenza in *or*: l'attributo globale è equivalente ad ogni singolo attributo locale con cui è messo in corrispondenza, e deve quindi originare più interrogazioni contemporaneamente per quella classe in cui vi è questo tipo di corrispondenza;

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

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


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

Figura 4.5: Esempio di classe globale in ODL$_{I^3}$

Al fine di poter specificare in modo dichiarativo questo insieme di informazioni aggiuntive, è stata proposta in ODL$_{I^3}$ una estensione alla classica definizione di classe attraverso il linguaggio ODL. Un esempio di descrizione di classe globale in ODL$_{I^3}$ è dato nella firgura 4.5, in riferimento alla GC$_1$.

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

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

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

Figura 4.6: Mapping Table di University_Person e Workplace

In MOMIS, questa definizione di classe, e le informazioni in essa contenute, dà origine ad una struttura dati tabellare, definita *mapping table*, che sarà necessaria a completare il processo di Query Decomposition. Come esempio, sono riportate in Figura 4.6 le mapping table dei cluster Cl_1 e Cl_2 di Figura 4.4, rappresentanti rispettivamente delle classi globali `University_Person` e `Workplace`.

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

4.5 Un Esempio di Decomposizione di Query

Il processo che è stato illustrato, comprensivo della possibilità di definire gli assiomi estensionali e di iterare la fase di generazione dello Schema Globale, conclude il *momento* dedicato all' *integrazione intensionale*. Lo schema ge-

nerato è di grande aiuto nel processo di Query Decomposition solo in senso lessicale: la definizione delle classi che partecipano ad una stessa classe globale (nella dichiarazione `extent`) e la Mapping table servono semplicemente a tradurre una query globale in una specifica per ogni sorgente:

Esempio 2 Riprendendo l'esempio dell'attributo globale `name`, è evidente come, in fase di interrogazione delle classi presenti nella sorgente S_1, l'attributo debba essere contemporaneamente tradotto dalla coppia `first_name` e `last_name`, e non in uno solo tra entrambi, in quanto è *equivalente* all'unione dei due. In questo modo, la query Q1 che richiede i nomi di tutte le persone presenti nella classe globale `University_Person`,

```
Q1  select  name
    from    University_Person
```

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

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

Volendo però **ottimizzare** il processo di formulazione delle subquery e di unione delle risposte è indispensabile poter trattare anche la conoscenza estensionale. Ad esempio può essere utile evitare di inviare subquery *ridondanti*, cioè che recuperano informazioni inutili, in quanto già recuperate da altre sorgenti.

Esempio 3 Si supponga di voler conoscere la nome, facoltà di appartenenza e rata delle tasse di uno studente, a partire dal suo numero di matricola:

```
Q3  select  name, faculty, tax_fee
    from    University_Person
    where   student_code=952
```

Le subquery che verrebbero formulate semplicemente dall' analisi della Mapping table sono:

```
Q3SM    select  first_name, last_name, faculty
        from    School_Member

Q3US    select  name, faculty_name, tax_fee
        from    University_Student
        where   student_code=952

Q3CSPS  select  first_name, last_name, faculty="CS"
        from    Student

        ...
```

In questo caso, come in molti altri, non ha senso interrogare tutte le sorgenti. In particolare, sapendo che l' estensione di `Student` è un sottoinsieme di quella di University_Student, e che le estensioni di `School_Member` ed University_Student sono equivalenti, si può evitare di inviare le subquery alle classi `School_Member` e `Student` semplicemente osservando che la subquery a University_Student recupera un sovrainsieme, rispetto alle altre sorgenti, delle proprietà richieste dalla `select`, e che inoltre, le altre sorgenti recuperano tutti i propri oggetti (non potendo verificare la clausola where) e quindi alla fine si richiede anche l' esecuzione dei join.

Nei seguenti capitoli si illustra come utilizzare la conoscenza estensionale per risolvere il problema della decomposizione delle query, rendendo il sistema *intelligente*, cioè' in grado di *scegliere* le sorgenti utili a produrre risultati significativi ed originali.

Capitolo 5

MOMIS: Integrazione Estensionale

L'utilizzo delle tecniche di clustering realizza l' **integrazione degli schemi** risolvendo le ambiguità intensionali e terminologiche fra le sorgenti. Per ottenere l' **integrazione delle informazioni** si deve introdurre la possibilità di definire e trattare anche la conoscenza estensionale, rappresentata dalle relazioni insiemistiche fra le estensioni delle classi, e le proprietà (chiavi) caratteristiche delle singole istanze.

Nella Sezione 5.1 si illustra come la sintassi delle rule nel linguaggio ODL_{I^3} consenta la proposizione di assiomi estensionali e si descrive il processo di individuazione delle *base extension* (vedi Sezione 3.2) a partire dagli assiomi, attraverso l' applicazione di ODB-Tools; inoltre si andrà ad illustrare l' architettura del modulo dedicato alla fusione delle gerarchie. Nella Sezione 5.2 si introduce il processo per eseguire la fusione delle istanze, ovvero dell' integrazione, unione e fusione dei dati, generate dall' interrogazione delle sorgenti, descrivendo anche le scelte di TSIMMIS. I progetti descritti nel Capitolo 2 non prevedono infatti di dover affrontare questo tipo di evenienza.

5.1 Fusione delle Gerarchie

Nel Capitolo 3 è stato descritto l' algoritmo per la fusione di gerarchie e classi intensionalmente omogenee (ma non identiche) e aventi estensioni sovrapposte. Riferendoci ad una generica classe globale dello schema integrato, per la quale siano state esplicitate le relazioni estensionali fra le classi sorgenti e le mapping rule, si descrive come integrare la tecnica

descritta precedentemente per generare la gerarchia *estensionale*[1] dalle classi che cooperano alla definizione del cluster. In particolare si evidenzia come ODB-Tools possa essere impiegato per generare la tabella delle *base extension* (in Sezione 3.2) a partire dagli assiomi estensionali dati dal progettista.

I passi, che saranno descritti nel seguito, possono essere sintetizzati come segue:

- Traduzione degli assiomi estensionali in proprietà intensionali
- Verifica di congruenza degli assiomi
- Individuazione dell' insieme delle *base extension*
- Creazione della Gerarchia Estensionale.

5.1.1 Gli assiomi estensionali

Gli assiomi sono introdotti per esprimere le relazioni insiemistiche tra le estensioni di classi definite in sorgenti autonome (indipendentemente dalla loro intensione). Negli schemi ad oggetti tradizionali le relazioni di inclusione ed intersezione fra le estensioni, all' interno di uno stesso schema, derivano direttamente da come sono definite le gerarchie d' ereditarietà (isa relationship), perciò non è necessario specificarle ulteriormente. Avendo scelto come linguaggio di definizione dello schema ODL_{I^3}, che è ad oggetti, ci si ispira a questa proprietà per manipolare gli assiomi estensionali ed estrapolarne le informazioni necessarie ad individuare le *base extension*.

Ogni Classe Globale dello schema integrato ha associato gli assiomi estensionali che predicano le relazioni di *inclusione*, *equivalenza* e *disgiunzione*. Per tutte le coppie di classi originarie per cui non è predicata nessuna particolare relazione estensionale si suppone esistere una *intersezione non nulla* delle estensioni. A livello implementativo si memorizzano le relazioni fra le coppie di classi in una matrice, il cui utilizzo sarà spiegato nel seguito.

Con riferimento all' esempio della classe globale `University_Person`, precedentemente introdotto ammettiamo la definizione di alcuni assiomi e illustriamo la matrice risultante in Tabella 5.1:

[1] Col termine *gerarchia estensionale* si indicherà nel seguito la gerarchia ottenuta col metodo illustrato nella Sezione 3.2.

```
rule RE1a forall x in School_Member then x in University_Student;

rule RE1b forall x in University_Student then x in School_Member;

rule RE2 forall x in Research_Staff then x in University_Worker;

rule RE3 forall x in Student then x in School_Member;

rule RE4 forall x in Professor then x in Research_Staff;

rule RE5 forall x in (Professor and School_Member)
                                  then x in bottom;

rule RE6 forall x in (Research_Staff and University_Student)
                                  then x in bottom;

rule RE7 forall x in (Research_Staff and Student)
                                  then x in bottom;
```

	US	**SM**	**CSP**	**P**	**S**	**UW**	**RS**
University_Student(**US**)	x	R1b					R6
School_Member(**SM**)	R1a	x		R5	R3		
CS_Person(**CSP**)			x				
Professor(**P**)		R5		x			R4
Student(**S**)		R3			x		R7
University_Worker(**UW**)							R2
Research_Staff(**RS**)	R6			R4	R7	R2	x

Tabella 5.1: Matrice delle Relazioni Estensionali

Si osservi che la matrice deve risultare simmetrica: le rule esprimono relazioni fra le estensioni di coppie di classi, la matrice esprime questo fatto indipendentemente dalla specifica relazione e dall' ordine in cui considerarle; in questo caso si evidenzia un' asimmetria nelle caselle (University_Student, School_Member) e (School_Member, University_Student) che valgono rispettivamente R1b e R1a ad indicare una relazione d' equivalenza. Per questa ragione la matrice viene modificata onde estendere le rule su School_Member ad University_Student, e viceversa. La modifica appare nella Tabella 5.2. Si osservi inoltre che nella matrice sono state introdotte ($\sim$) anche le relazioni estensionali (inclusioni) derivabili dalle **isa** degli schemi ad oggetti.

	US	**SM**	**CSP**	**P**	**S**	**UW**	**RS**
University_Student(**US**)	x	R1b		**R5**	**R3**		R6
School_Member(**SM**)	R1a	x		R5	R3		**R6**
CS_Person(**CSP**)			x	$\sim$	$\sim$		
Professor(**P**)	**R5**	R5	$\sim$	x			R4
Student(**S**)	**R3**	R3	$\sim$		x		R7
University_Worker(**UW**)							R2
Research_Staff(**RS**)	R6	**R6**		R4	R7	R2	x

Tabella 5.2: Matrice delle Relazioni Estensionali Modificata

Nell' esempio in esame si devono cioè specificare relazioni fra CS_Person e Professor e Student.

Sintassi e Semantica degli Assiomi

Gli assiomi possono essere espressi come rule nel linguaggio ODL$_{I^3}$, senza bisogno di estendere ulteriormente la sintassi e senza creare complicazioni all' utilizzatore. La ragione che consente di ricorrere alla sintassi delle rule risiede nel fatto che assiomi estensionali e rule sono semanticamente equivalenti. Le rule infatti costituiscono un modo per dire che un insieme di istanze di un certo concetto C1, che godono di certe proprietà, appartengono ad un altro concetto C2, vincolando così delle relazioni estensionali a proprietà intensionali. Gli assiomi, anche se indipendenti dalle caratteristiche intensionali, aspirano a *ridefinire* i tipi delle istanze esistenti: dire che A e B hanno estensioni disgiunte significa dire che non esistono tipi di istanze con le proprietà e di A e di B, mentre affermare che A e B hanno estensioni equivalenti significa che tutte le istanze di A sono anche istanze di B e tali istanze hanno le proprietà sia di A sia di B.

È stato pensato di esprimere le relazioni di disgiunzione, inclusione ed equivalenza fra le estensioni, sottintendendo l'intersezione in quanto si presume che classi che costituiscono uno stesso cluster abbiano almeno estensioni la cui intersezione non è nulla. In particolare avremo:

1. A e B sono estensioni disgiunte: $A \cap B = \emptyset$

   ```
   rule RE1 forall x in (A and B) then x in bottom
   ```

2. B è inclusa in A: $B \subseteq A$

   ```
   rule RE2 forall x in B then x in A
   ```

3. A e B sono equivalenti: $A = B$

   `rule RE3 forall x in A then x in B`

   `rule RE4 forall x in B then x in A`

Controllo di Correttezza e Generazione delle *Base Extension*

È necessario evidenziare che l' affermazione di disgiunzione, unita a quelle di inclusione ed equivalenza, può generare errori impliciti, che devono essere rilevati prima di attivare il processo di identificazione delle *base extension*. Ad esempio le seguenti tre affermazioni portano ad affermare che C è un concetto incoerente ($C \simeq \perp$):

1. A e B hanno estensioni disgiunte
2. C è inclusa in A
3. C è incluso in B

Per rilevare queste possibili incongruenze si ricorre ad ODB-Tools ed alla subsumption. Si traducono le rule estensionali in proprietà intensionali: in particolare l' inclusione si traduce in una relazione isa, l' equivalenza genera una classe *equivalente*, una rule di disgiunzione origina una classe intersezione di tipo *bottom*. Per tutte le coppie di classi originarie per cui non sia stata predicata nessuna relazione estensionale si suppone che esista una relazione di intersezione non nulla. Queste relazioni implicite vengono tradotte attraverso la definizione di classi, con la semantica di classi virtuali rappresentanti l' intersezione delle classi. In Figura 5.1 si dà un esempio delle *trasformazioni intensionali* da eseguire, facendo riferimento alle seguenti relazioni:

1. `School_Member` ed `University_Student` hanno estensioni equivalenti
2. l' estensione di `Professor` è un sottoinsieme di quella di `Research_Staff`
3. le estensioni delle classi Student e Professor sono disgiunte
4. le estensioni delle classi `CS_Person` e `University_Worker` sono parzialmente sovrapposte

Le frecce tratteggiate rappresentano le nuove relazioni isa. Le classi col bordo più scuro le classi *ridefinite* (ad es. `Professor`), o di nuova introduzione, quelle tratteggiate rappresentano le *classi intersezione*.

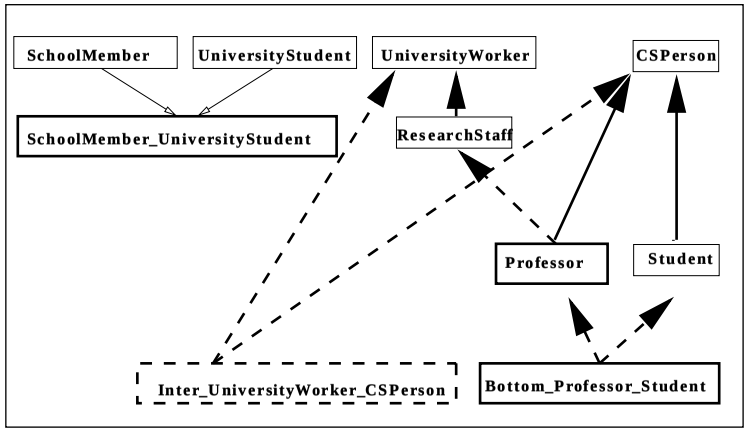


Figura 5.1: Esempio di Trasformazione Intensionale

Traducendo così tutti gli assiomi estensionali e calcolando la subsumption sull' insieme di classi ottenute, si può effettuare un controllo di correttezza degli assiomi. Se infatti è riscontrata una relazione di sussunzione fra una classe di tipo *bottom* ed una *classe intersezione*, significa che è stata fatta una affermazione incongruente e si è anche in grado di evidenziare la relazione di disgiunzione che ha generato l' incoerenza.

Traduzione degli assiomi estensionali in proprietà intensionali È stato appena affermato che per utilizzare ODB-Tools si devono trasformare gli assiomi estensionali in proprietà intensionali. ODB-Tools è uno strumento che, basandosi sulle intensioni, consente di organizzare classi e tipi virtuali in tassonomie: una tassonomia evidenzia relazioni di specializzazione fra le intensioni di classi attraverso relazioni isa.

Spieghiamo con maggiore precisione in che cosa consiste la trasformazione degli assiomi estensionali in proprietà intensionali. Siano stati definiti, relativamente ad ogni cluster: gli assiomi estensionali, le descrizioni delle classi delle sorgenti e la *mapping table*: ODB-Tools non riesce a generare automaticamente un' unica tassonomia perché:

- le classi sorgenti non sono legate da un diretto legame intensionale

- non riesce ad interpretare gli assiomi estensionali.

Per poter operare è necessario eseguire preventivamente le seguenti operazioni:

- ridefinizione delle classi utilizzando la mapping table in modo da risolvere le maggiori incongruenze intensionali, rendendo i nomi degli attributi omogenei;
- traduzione degli assiomi estensionali in relazioni d' ereditarietà con il seguente approccio[2]:

  1. ogni asserzione d' equivalenza fra due classi porta alla generazione di una classe con intensione corrispondente all' unione delle due intensioni. Ad esempio date le seguenti definizioni:

```
interface School_Member
{ attribute string  name;
  attribute string  faculty;
  attribute integer year; } ;
```

```
interface University_Student
{ attribute string name;
  attribute integer studcode;
  attribute string faculty;
  attribute integer tax; } ;
```

```
rule RE1a forall x in School_Member then x in University_Student;

rule RE1b forall x in University_Student then x in School_Member;
```

     si introduce la *classe equivalente* che sostituisce University_Student e School_Member, e la cui intensione è pari all' unione delle rispettive intensioni:

```
interface School_Member_University_Student
{ attribute string  name;
  attribute string  faculty;
  attribute integer studcode;
  attribute integer year;
  attribute integer tax; } ;
```

  2. ogni asserzione d' inclusione *ridefinisce* la classe inclusa introducendo l' ereditarietà dalla superclasse. Ad esempio:

```
interface University_Worker
{ attribute string name
  attribute integer deptcode;
  attribute integer pay; } ;
```

```
interface Research_Staff
{ attribute string name
  attribute string relation;
  attribute string email;
  attribute integer deptcode;
  attribute integer sectioncode;
  attribute integer pay; } ;
```

[2]Si faccia riferimento all' esempio di riferimento in Appendice C


```
interface CS_Person
{ attribute string name; } ;
```

```
interface Student : CS_Person
{ attribute integer year;
  attribute set<Course> takes;
  attribute string rank; } ;
```

```
interface Professor : CS_Person
{ attribute string title;
  attribute Division belongsto;
  attribute string relation; } ;

rule RE2 forall x in Research_Staff then x in University_Worker;

rule RE3 forall x in Student then x in School_Member_University_Student;

rule RE4 forall x in Professor then x in Research_Staff;
```

Research_Staff, Professor e Student sono così ridefinite[3]:

```
interface Research_Staff : University_Worker
{ attribute string email; } ;
  attribute string relation;
  attribute integer sectioncode; } ;

interface Professor : CS_Person, Research_Staff
{ attribute string title;
  attribute Division belongsto;
  attribute string relation; } ;

interface Student : CS_Person , School_Member_University_Student
{ attribute integer year;
  attribute set<Course> takes;
  attribute string rank; } ;
```

  3. ogni asserzione di disgiunzione introduce un tipo *bottom* che eredita dalle classi disgiunte. Ad esempio:

```
rule RE5 forall x in (Professor and
   School_Member_University_Student) then x in bottom;
```

[3]Si osservi che la *classe equivalente* deve sostituire sempre le classi cui si riferisce, come ad esempio nelle rule RE3, RE5, RE6 la classe School_Member_University_Student sostituisce School_Member e University_Student.

```
rule RE6 forall x in (Research_Staff and
   School_Member_University_Student) then x in bottom;
```

generano le nuove classi *bottom*:

```
view Bottom_P_SS : Professor , School_Member_University_Student
{} ;
view Bottom_RS_SS: Research_Staff ,
                   School_Member_University_Student
{} ;
```

4. ogni *classe intersezione*[4] porta alla definizione di una nuova classe **virtuale** che specializza entrambe le classi. Ad esempio:

```
view Inter_CSP_UW : CS_Person , University_Worker
{} ;
```

Questa classe rappresenta le estensioni comuni a `CS_Person` ed `University_Worker`.

Verifica di congruenza Le classi originarie *ridefinite* in base agli assiomi di inclusione e le classi create dagli assiomi di disgiunzione ed equivalenza sono introdotte nello schema. Inoltre anche le *classi intersezione*, generate dalle relazioni implicite d' intersezione, sono introdotte nello schema. Viene attivato ODB-Tools sullo schema così definito, questi calcola la subsumption e scopre le classi incoerenti, cioè sussunte dalle classi *bottom*. Se non ne sono riscontrate significa che gli assiomi dati sono fra loro congruenti. Se invece una view è trovata incoerente, analizzando la sua gerarchia d' ereditarietà si riesce a recuperare l' assioma di disgiunzione che ha creato l' incoerenza. Spetta al progettista decidere se l' assioma deve essere eliminato o se specificare un assioma di disgiunzione fra le classi che definiscono la vista. Ad esempio, ritornando al caso delle rule precedentemente dichiarate per la classe globale `University_Person`, si evidenzia che non è stata predicata nessun tipo di relazione fra le classi `Student` e `Research_Staff`, che perciò danno origine alla vista `Inter_RS_S` :

```
view Inter_RS_S : Research_Staff , Student
{} ;
```

[4] Queste sono implicite per ogni coppia di classi per cui non sia stata predicata esplicitamente nessuna relazione estensionale e sono ricavate dall' analisi della matrice introdotta all' inizio di questa Sezione.

In Figura 5.2 si mostra l' output di ODB-Tools, dove si mette in evidenza che il tipo `Inter_RS_S` è sussunto dal tipo *bottom* `Bottom_RS_SS`, precedntemente introdotto, costituendone una specializzazione: in questo caso si lascia al progettista la scelta se specificare la rule che esprime la disgiunzione delle estensioni di `Research_Staff` e `Student`:

```
rule RE7 forall x in (Research_Staff and Student)
then x in bottom;
```

o se eliminare la `rule RE6` che ha generato la definizione `Bottom_RS_SS`. Analogamente si verifica per il tipo `Inter_P_S` .

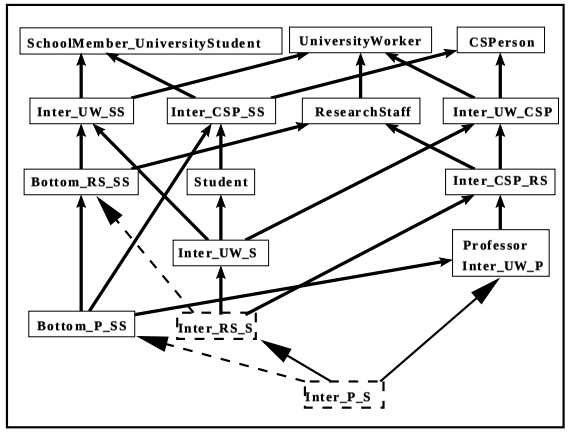


Figura 5.2: Esempio di Verifica di Congruenza

Individuazione delle Base Extension Una volta verificata la correttezza si procede all' identificazione delle *base extension*. L'idea avuta per generarle comporta queste fasi:

1. costruzione, con ODB-Tools, della tassonomia utilizzata per la *verifica di congruenza* a meno dei tipi *bottom*[5]

[5] Si osservi che le viste devono essere ridefinite come concetti primitivi e non virtuali per evitare che siano stabilite relazioni di specializzazione indesiderate.

2. assegnazione ad ognuna delle classi delle *base extension*: tale assegnamento preserva le relazioni d' ereditarietà evidenziate nella tassonomia.

Illustriamo i passi dell' algoritmo:

- Ad ogni classe della tassonomia si assegna una propria *base extension* che identifica le istanze proprie solo di quella classe (questa è la ragione per cui risulta inutile considerare le classi legate da assiomi di equivalenza e si considera solo la rispettiva classe equivalente).
- Ad ogni classe sono aggiunte le *base extension* delle sottoclassi: si parte dai nodi foglia della tassonomia e si riassegnano le estensioni del livello superiore. Questo processo viene reiterato fino al completamento della gerarchia.

Il procedimento ideato assicura di popolare le classi originarie esattamente con le proprie estensioni, partizionate rispetto agli assiomi espressi.

Esempio: base extension della classe globale `University_Person` Si consideri la classe globale `University_Person` arricchita dagli assiomi estensionali ottenuti alla fine del processo di verifica della coerenza. In Figura 5.3 si illustra il processo di generazione delle *base extension*, in particolare si evidenzia come la *base extension 1, 2*, caratteristica di `Professor`, venga inserita fra le estensioni delle classi `CS_Person, University_Worker, Research_Staff`, ma non in quelle di `School_Member_University_Student` e `Student`. In Tabella 5.3 sono raccolte le relazioni estensionali trovate dall' algoritmo per le classi originarie[6].

5.1.2 L'architettura del modulo IHB

Il modulo IHB in Figura 5.4 riceve in input le descrizioni ODL_{I^3} delle classi globali e sorgenti, la mapping table e gli assiomi estensionali introdotti dal progettista e genera, in maniera semi automatica, la *Gerarchia Estensionale*. Il processo non è completamente automatizzato per lasciare al progettista la possibilità di interagire qualora si evidenzino situazioni di incongruenza fra gli assiomi.

In particolare il modulo IHB si scompone in due sottomoduli che eseguono le seguenti operazioni:

[6]Si osservi che le estensioni *1, 2*, che caratterizzano `Professor`, sono state collassate in *1*. Si è potuto collassarle inquanto *1* e *2* non comparivano mai separate, in nessuna altra classe originaria.

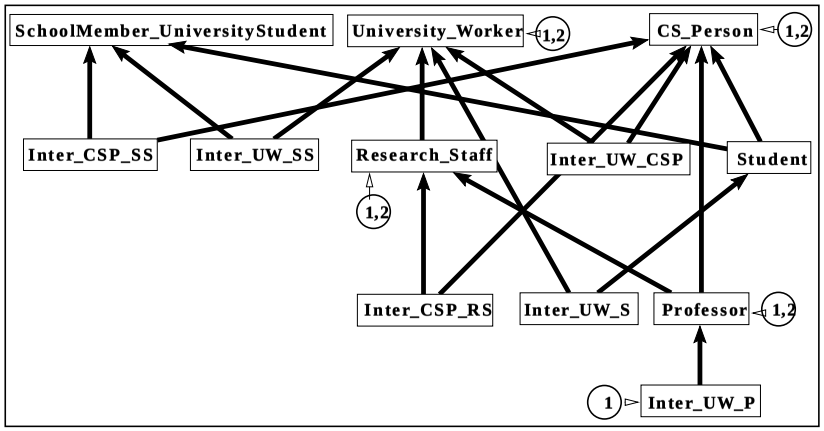


Figura 5.3: Esempio di Assegnamento delle Base Extension

1. traduzione degli assiomi estensionali, verifica di congruenza e generazione della tabella delle *base extension*, (IHBA)
2. generazione della *Gerarchia Estensionale*, come descritto nella Sezione 3.2, (IHBB).

Gerarchia Estensionale di `University_Person`
Consideriamo l' esempio di riferimento, in Appendice C.

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

Tabella 5.3: Base Extension delle Classi del Cluster `University_Person`

Figura 5.4: Architettura del modulo IHB

La Tabella 5.4 rappresenta le definizioni delle classi in forma tabellare, (per ragioni di spazio non sono mostrati tutti gli attributi). La Tabella 5.5 si ottiene sovrapponendo alla precedente tabella quella delle *base extension* 5.3 ed indica da quale intensione minima sono caratterizzate le istanze appartenenti ad ogni base extension.

Attribute	name	dept	pay	fac	year	tax	rank	rel
University_Student	x			x		x		
School_Member	x			x	x			
CS_Person	x							
Professor	x	x					x	
Student	x				x		x	
University_Worker	x	x	x					
Research_Staff	x	x	x					x

Tabella 5.4: Proprietà Intensionali delle Sorgenti di University_Person

Mostriamo come procede l' algoritmo elaborato in [6]. Sia data in input la Tabella 5.5: siano M={name,dept,pay,faculty,...} e G={1,2,3,4,5,6,7,...} l' insieme degli attributi e quello delle *base extension*, rispettivamente. Illustriamo i risultati dei passi fondamentali:

Base Extension	1	2	3	4	5	6	7	8	9	10	11
name	x	x	x	x	x	x	x	x	x	x	x
dept	x	x	x		x	x	x				
pay	x	x	x		x	x	x				
fac		x					x	x	x	x	
year		x					x	x	x	x	
tax		x					x	x	x	x	
rank	x	x						x			
email	x				x						
relation	x				x						
takes		x						x			
title	x										
section	x				x						
studcode		x					x	x	x	x	

Tabella 5.5: Relazioni Estensioni-Attributi

1. Definizione degli insiemi *Int* ed *Ext* (in Tabella 5.6): *Int* è l' insieme delle intensioni (insiemi di attributi) di ogni *base extension* e si ricava esaminando la Tabella 5.5 per colonne[7]; viceversa *Ext* è l' insieme delle estensioni corrispondenti ad ogni attributo m di M e si ricava esaminando per righe la medesima tabella[8].

2. Costruzione degli insiemi *ConI* e *ConE* (in Tabella 5.7). *ConI* e *ConE* sono insiemi di *concept*[9] cioè di coppie: ogni coppia è costituita da un insieme di attributi e di *base extension*, in particolare *ConI* è costruito a partire dagli insiemi di attributi di *Int*, mentre *ConE* a partire dagli insiemi di estensioni di *Ext*. Dato un insieme di attributi, l' insieme di *base extension* corrispondenti nella coppia è costituito da tutte quelle che hanno almeno quegli attributi. Analogamente, ad un insieme di *base extension* corrisponde l' insieme di attributi comuni a tutte.

3. Costruzione delle classi che compongono la Gerarchia Estensionale (in

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

[8]Non vengono replicati gli insiemi base extension nel caso presentino i medesimi attributi.

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

$$Int := \left\{ \begin{array}{l} \{name, dept, pay, rank, email, relation, title, section\}, \\ \{name, dept, pay, faculty, year, tax, rank, takes, studcode\}, \\ \{name, dept, pay\}, \\ \{name\}, \\ \{name, dept, pay, email, relation, section\}, \\ \{name, dept, pay, faculty, year, tax, studcode\}, \\ \{name, faculty, year, tax, rank, takes, studcode\}, \\ \{name, faculty, year, tax, studcode\}, \end{array} \right\}$$

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

Tabella 5.6: Insiemi Int ed Ext

Tabella 5.8). Dall' unione insiemistica di *ConI* e *ConE* si ottengono le definizioni in termini di intensioni e di estensioni delle nuove classi. In particolare l' intensione di una classe è costituita dagli attributi che costituiscono il primo elemento di una coppia, l' estensione dagli oggetti delle *base extension* corrispondenti al secondo insieme della coppia.

4. Generazione della Gerarchia Estensionale (in Figura 5.5). Per rilevare le relazioni di specializzazione fra le classi dell' insieme *Con*, le si dà in pasto ad ODB-Tools, che ne ricava la tassonomia, senza bisogno di dover ricorrere alla costruzione delle matrici M ed N, come invece è stato descritto in [6] e nella Sezione 3.2.

Una volta ottenuta la Gerarchia Estensionale sarà rispetto a questa che verranno espanse ed ottimizzate le query. Individuata la classe dell' insieme *Con* che gneralizza una certa query, si devono recuperare le classi sorgenti ad essa collegate. Per ritrovare questo legame si risale alle *base extension* rappresentate, poi, dalla Tabella delle *base extension*, si risale alle classi coinvolte. Si veda un esempio dei legami tra la classe **C6** della Gerarchia e le sorgenti in Figura 5.6.

Si deve osservare che non si ricercano le sorgenti che presentano **tutte** le

$$ConI := \left\{ \begin{array}{l} (\{name, dept, pay, rank, email, relation, title, section\}, \{1\}), \\ (\{name, dept, pay, faculty, year, tax, rank, takes, studcode\}, \{2\}), \\ (\{name, dept, pay\}, \{1,2,3,5,6,7\}), \\ (\{name\}, \{1,2,3,4,5,6,7,8,9,10,11\}), \\ (\{name, dept, pay, email, relation, section\}, \{1,5\}), \\ (\{name, dept, pay, faculty, year, tax, studcode\}, \{2,7\}), \\ (\{name, faculty, year, tax, rank, takes, studcode\}, \{2,8\}), \\ (\{name, faculty, year, tax, studcode\}, \{2,7,8,9,10\}) \end{array} \right\}$$

$$ConE := \left\{ \begin{array}{l} (\{name\}, \{1,2,3,4,5,6,7,8,9,10,11\}), \\ (\{name, dept, pay\}, \{1,2,3,5,6,7\}), \\ (\{name, faculty, year, tax, studcode\}, \{2,7,8,9,10\}), \\ (\{name, rank\}, \{1,2,8\}), \\ (\{name, dept, pay, email, relation, section\}, \{1,5\}), \\ (\{name, faculty, year, tax, rank, takes, studcode\}, \{2,8\}), \\ (\{name, dept, pay, rank, email, relation, title, section\}, \{1\}) \end{array} \right\}$$

Tabella 5.7: Insiemi ConI e ConE

base extension che caratterizzano la classe prescelta della gerarchia, quanto **almeno una** di esse.

Considerazioni Il metodo illustrato fonde delle classi aventi intensioni ed estensioni parzialmente sovrapposte. È stata usata la conoscenza estensionale, derivata dagli assiomi e dalle relazioni di ereditarietà, per creare una partizione dell' insieme totale delle entità[10] contenute nelle sorgenti, intendendo per entità non tanto gli oggetti di ogni sorgente locale caratterizzati dalle relative proprietà, quanto l' insieme delle informazioni, su uno stesso oggetto, che si ottengono complessivamente dalle diverse sorgenti. Ogni sottoinsieme (*base extension*) della partizione contiene entità distinte non contenute dagli altri sottoinsiemi. Le proprietà delle entità in ogni partizione sono definite da un insieme di attributi (questo è il significato della Tabella 5.5 delle relazioni fra estensioni ed attributi) ricavati dall' unione di tutti gli attributi delle classi la cui intersezione forma la *base extension*. Una volta ottenute queste informazioni, per creare le classi della gerarchia, si devono riunire le istanze delle *base extension* sulla base delle similarità intensionali. Per fare

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

$$Con: = \left\{ \begin{array}{l} C1: (\{name\}, \{1,2,3,4,5,6,7,8,9,10,11\}), \\ C2: (\{name, dept, pay\}, \{1,2,3,5,6,7\}), \\ C3: (\{name, faculty, year, tax, studcode\}, \{2,7,8,9,10\}), \\ C4: (\{name, dept, pay, faculty, year, tax, studcode\}, \{2,7\}), \\ C5: (\{name, faculty, year, tax, rank, takes, studcode\}, \{2,8\}), \\ C6: (\{name, dept, pay, email, relation, section\}, \{1,5\}), \\ C7: (\{name, dept, pay, faculty, year, tax, rank, takes, studcode\}, \{2\}), \\ C8: (\{name, rank\}, \{1,2,8\}), \\ C9: (\{name, dept, pay, rank, email, relation, title, section\}, \{1\}) \end{array} \right\}$$

Tabella 5.8: Classi della Gerarchia Estensionale di `University_Person`

ciò si devono considerare gli insiemi di attributi propri di più *base extension* e formare una classe per ogni insieme di attributi cui corrisponda una, o più, *base extension*. Ognuna di queste classi contiene istanze aventi un insieme minimo di proprietà in comune (gli attributi della classe), ed inoltre sono ammesse anche le istanze con altri attributi (secondo una semantica di mondo aperto): a seconda di queste proprietà le istanze saranno riclassificate nelle sottoclassi della classe in considerazione.

5.2 Fusione delle Istanze

La gerarchia creata rappresenta i diversi *tipi* ottenibili *fondendo gli oggetti* provenienti dalle sorgenti: le sue classi sono contenitori vuoti che vengono eventualmente riempiti solo in fase di raccolta delle risposte alle subquery inviate.
Un problema fino ad ora trascurato riguarda la realizzazione della fusione delle istanze, in particolare:

- come identificare istanze *equivalenti*, cioè relative ad una stessa entità del mondo reale, provenienti da sorgenti **autonome**
- come generare un' unica istanza a partire da due o più *equivalenti*

Prima di proporre una soluzione per MOMIS, si cerca di dare una visione generale su come sono identificati gli oggetti nei sistemi DB, KB e su come sono riconosciuti *equivalenti* e quindi fusi in TSIMMIS.

Figura 5.5: Gerarchia Estensionale di `University_Person`

5.2.1 Identificazione: OID vs. UID

Un' adeguata identificazione degli oggetti persistenti, nelle basi di dati, consente il riconoscimento delle entità uguali nei valori ma distinte come istanze del mondo reale, e di reperirle in maniera automatica.

I **sistemi relazionali** considerano le tuple come set di valori e per distinguere tuple uguali nei valori ma distinte come istanze si ricorre all' uso di chiavi: gli UID (User-defined IDentifier). Gli uid devono essere definiti esplicitamente dal progettista fra i campi delle relazioni. La scelta dei campi che li compongono richiede un' accurata valutazione in fase di progettazione: un identificatore deve essere unico per ogni possibile tupla legale del database, non deve essere mai nullo (o parzialmente nullo nel caso di chiave composta) per evitare il rischio di perdere il riferimento all'oggetto che identifica, deve consentire riferimenti e collegamenti fra le tuple, deve rimanere immutato durante l' intera esistenza dell' oggetto per non dover cambiare tutti i riferimenti incrociati, qualora si scegliesse un identificatore numerico la sua gestione dovrebbe essere trasparente all' utente, anche se l'uid non è trasparente all' utente e quindi è consentito recuperare l' identificatore a partire dall' oggetto. Viceversa i **sistemi ad oggetti** fanno riferimento agli OID (Object-IDentifier). Essi sono assegnati dal sistema in modo sicuramente univoco al momento della creazione di ogni oggetto, restano immutati

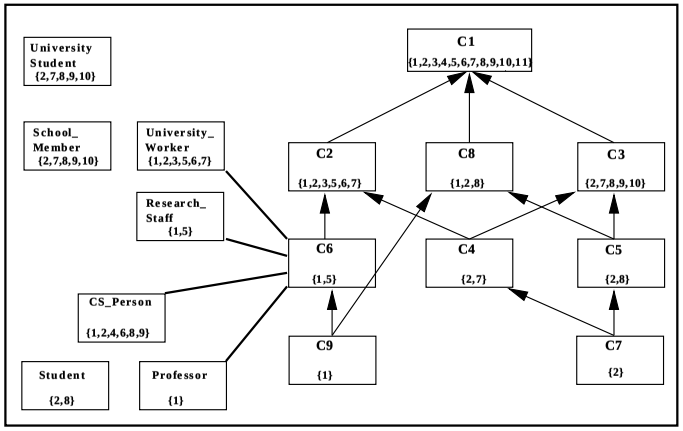


Figura 5.6: Legami fra la Gerarchia e le Sorgenti

durante tutta la sua esistenza e sono invisibili all' utente, inoltre la gestione degli oggetti complessi consente automaticamente di mantenere i collegamenti fra gli stessi, consentendo un meccanismo di recupero delle informazioni navigazionale e non tramite join.

Esistono anche approcci che tradizionalmente non rappresentano l' identità: in particolare quelli a base di conoscenza realizzati attraverso modelli che fanno uso di linguaggi dichiarativi basati sulla logica e le regole non trattano l' identità delle istanze. Attualmente si sta cercando di estendere anche a questi modelli con la possibilità di definire e manipolare l' identità degli oggetti [43].

5.2.2 Fusione nei Sistemi Mediatore

I pricipi generali per cui gestire l' identità restano validi nei sistemi di mediazione, ma occorre svolgere anche altre considerazioni fondamentali legate al fatto che i Mediatori *non sono sistemi proprietari* degli oggetti che mostrano agli utenti, ma costituiscono una vista di informazioni mantenute a livello di sorgenti distinte ed indipendenti.

La scelta di come identificare gli oggetti del Mediatore deve avere in considerazione il fatto che le sorgenti che esso integra possano anche non ricorrere agli oid (in questi casi il Wrapper, esportando gli elementi dalle

sorgenti con un approccio Object-Oriented [8, 16], ne genererà uno casuale a run time) e che comunque gli oid di una sorgente sono completamente indipendenti da quelli di un'altra. Gli oid con cui sono esportati gli oggetti di una sorgente sono significativi solo nell' ambito della singola query, infatti lo stesso oggetto, reperito da un' interrogazione fatta in un momenti diversi, potrà avere tutt' altro oid.

Il Mediatore deve trattare gli oid degli oggetti che gli provengono dai Wrapper, cioè

- deve essere realizzata la fusione degli oggetti rappresentanti la stessa informazione
- devono poter essere creati degli oggetti complessi.

Il primo punto comporta il riconoscimento, sulla base di informazioni mantenute in un qualche attributo semanticamente significativo, dell' identità di due elementi. Ad esempio due oggetti provenienti da due Sorgenti/Wrapper diverse potrebbero avere un medesimo oid ma essere due informazioni non collegate, mentre due entità con oid diversi potrebbero essere benissimo riconosciute come parte di una medesima informazione sulla base di un attributo di join. Il secondo punto esprime la necessità di mantenere e/o creare oggetti complessi che soddisfino lo schema del Mediatore. Ad esempio siano date due sorgenti A e B, A ad oggetti e B relazionale, Consideriamo un oggetto, esportato dal Wrapper A con identificatore O1, che abbia un campo il cui valore è un oggetto esportato dal medesimo Wrapper con identificatore O2, inoltre il Wrapper B assegni ad una certa tupla l'oid O2 (che però non ha nulla a che vedere con O2 della sorgente A) ed in base ad una FK si sappia che esiste un collegamento con una tupla O3 della sorgente B che deve essere esportata. Si riconosca che O3 di B ed O2 di A debbano essere fuse. In conclusione O1 del Wrapper A ed O2 del Wrapper B sono due oggetti distinti ma entrambi mantengono un riferimento ad un medesimo oggetto (corrispondente ad un altro cluster del Mediatore) che è stato esportato dal Wrapper A con identificatore O2 dal Wrapper B con O3 e che dovrà essere fuso nel cluster in un oggetto privato del Mediatore ed identificato in maniera univoca da O1 di A ed O2 di B. Si veda, a titolo chiarificatore, l' esempio di Figura 5.7.

TSIMMIS

Tsimmis è un' architettura ideata per favorire l' integrazione di sorgenti semistrutturate, ove le informazioni recate dalle diverse sorgenti sono unite con lo scopo di creare un' informazione maggiormente completa ed aggiornata. Un importante aspetto del Mediatore consiste perciò nella capacità

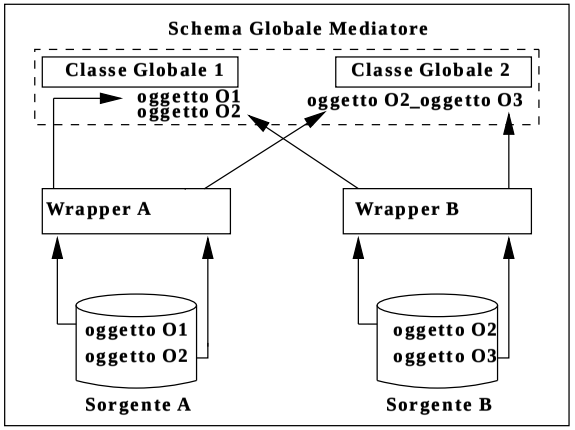


Figura 5.7: Identificazione e Fusione degli Oggetti

di fondere (raggruppare) in un unico oggetto le informazioni provenienti da oggetti siti in sorgenti differenti, ma relative ad una stess istanza della realtà. Per ottenere questo risultato in TSIMMIS si hanno due approcci: uno basato sul concetto di *unifier*, l' altro sull' assegnazione di *oid specifici* o *inventati* applicando una funzione di skolemizzazione ad una chiave semantica comune.

Nell' approccio tramite *unifier*, per ottenere le subquery per ogni sorgente si creano dei *binding* (uguaglianze) fra i valori associati alle variabili contenute nel body della query e quelle nella rule: ogni predicato del body della regola esprime una possibile subquery che viene inviata alla sorgente. Ad esempio la query `Q1` sulla descrizione di `cs_person` data in `MSL1` si esprime così:

```
(MSL1) Rules:
<cs_person {<name N> <rel R> Rest1 Rest2}>
       :- <person {<name N> <dept 'CS'> <relation R>|Rest1}>@whois
and decomp(N, LN, FN)
and <R{<firstname FN> <lastname LN>|Rest2}>@cs
(Q1) Query:
JC :- JC:<cs_person {<name 'Joe Chung'>}>@med
```

viene espansa nel *Logical Datamerge Program* specificato dalla `rule MSL2`:

```
(MSL2) Rules:
```



```
<cs_person {<name 'Joe Chung'> <rel R> Rest1 Rest2}>
       :- <person {<name 'Joe Chung'> <dept 'CS'> <relation R>|Rest1}>@whois
and decomp('Joe Chung', LN, FN)
and <R{<firstname FN> <lastname LN>|Rest2}>@cs
```

A run time, in risposta alla query si creano delle *binding list*, cioè delle associazioni fra le variabili utilizzate nella query ed i valori degli oggetti soddisfacenti l' interrogazione specificata. Ad esempio i *binding* per `whois` sono sulle variabili `R` e `Rest1`, ma sono le variabili N ed R quelle su cui uguagliare gli oggetti. Il Wrapper associa ad ogni variabile una *binding list* coi valori relativi ad ogni oggetto che riceve in risposta all' interrogazione fatta e la spedisce al Mediatore. A questo livello sono generati gli oggetti virtuali: si considerano solo quelle *binding list* che, relative a sorgenti diverse, associano a variabili con lo stesso nome gli stessi valori, dall' unione di queste liste si ottengono i vari oggetti. Tecnicamente si dice *unifier* quella *binding list* che, date due espressioni logiche p e q contenenti variabili e costanti, le uguagli [28]. Questa soluzione è detta a *unifier* perché dscerive il *matching* fra la rule e la query. Questo approccio è intuitivo e non comporta complicazioni in fase di fusione degli oggetti. Un limite consiste però nel fatto che solo gli oggetti comuni alle sorgenti sono considerati, mentre quelli non "duplicati" sono ignorati; inoltre non rende possibile recuperare i dati di un oggetto a partire dal proprio identificatore perché l' oid viene generato dal componente run time del Mediatore in modo casuale ad ogni risposta ad un' interrogazione e quindi l' utente può solo inferire relazioni di composizione fra gli oggetti all' interno di una certa risposta, mentre per recuperare un oggetto il Mediatore dovrebbe mantenere una tabella permanenete con le associazioni oid-oggetto.

La seconda specificazione del mediatore rivede l' approccio relazionale degli identificatori semantici cercando di legarli agli oid in accordo con il modello OEM. Fare riferimento al concetto di uid significa riconoscere nella struttura degli oggetti esportati un campo che semanticamente li identifichi univocamente e sia significativo per l' utente (una chiave). Si scrive una rule per ogni sorgente da cui esportare (e non una singola rule nel cui body siano specificate le descrizioni di tutte le sorgenti) e si impone nel campo oid della rule hea il nome della variabile associata al campo semanticamente significativo.

```
<N cs_person {<name N>, Restwhois}>
         :-  <person{<name N>, <dp:'CS'> | Restwhois}>@whois
<N cs_person {<name N>, <rel R>, Restcs}>
         :-  <R{<name N> | Restcs}>@cs
```

La variabile N serve a creare l' *oid specifico* dal campo name.
Questo identificatore non è "permanente": le sorgenti da cui l' oggetto proviene possono modificarlo e/o cancellarlo essendo questo di loro proprietà, inoltre localmente non sarà identificato dall' oid specifico generato dal Mediatore, quindi per reperire l' oggetto a partire dal suo oid specifico occorrerà un passaggio che dalla specificazione dell' oid porti alla specificazione del valore del campo prescelto perché semanticamente valido. Vediamo un esempio di oggetto esportato con oid specifico 'Joe Chung' [7]:

```
<'Joe Chung', cs_person, set, {&mn1, &mrel1, ...}
        <&mn1, nome, string, 'Joe Chung'>
        <&mrel1, relation, string, 'professor'>
        ...
```

Vediamo una query orientata all' oid per recuperare l' oggetto a partire dall' identificatore che appunto è 'Joe Chung'.

```
person :- person:<'Joe Chung' cs_person V>@med-oid
```

Per ottenere le subquery essa è tradotta in query orientate al valore ottenute col meccanismo di *matching*:

```
<'Joe Chung' cs_person {<name 'Joe Chung'>, Restwhois}>
        :- <person{<name 'Joe Chung'>, <dp:'CS'> | Restwhois}>@whois
<'Joe Chung' cs_person {<name 'Joe Chung'>, <rel R>, Restcs}>
        :-  <R{<name 'Joe Chung'> | Restcs}>@cs
```

Un vantaggio offerto da questa soluzione deriva dalla capacità di raggruppare (fondere) gli oggetti, a prescindere dalla sorgente in cui si trovano, sulla base dell' identificatore prescelto per la definizione dell' oid specifico: questo implicitamente significa che non è strettamente necessario che la chiave sia intesa come identificatore unico di un oggetto all' interno di una relazione. [7]
Questo approccio consente di trattare simultaneamente diversi problemi legati all' integrazione, quali ad esempio il caso che una stessa classe contenga più oggetti che facciano riferimento ad un' unica entità, o che non ci sia il corrispettivo di un certo oggetto in altre classi, senza la necessità che effettivamente sia definita per ogni classe una chiave. Rispetto all' approccio tramite *unifier* realizza un modello più vicino ad un approccio agli oggetti perché sfrutta in maniera completa la possibilità di avere oid che non siano valori assegnati casualmente (l' approccio tramite *unifier* in questo senso assegnava un significato fittizio all' oid), e ottiene una maggior modularità della soluzione inquanto l' introduzione di una nuova sorgente determina esclusivamente la scrittura di una nuova rule, col vantaggio di non avere eccessivi

legami fra i nomi delle variabili utilizzate nelle rule. Una condizione che può essere vincolante risiede nella necessità che esista una chiave semantica comune a tutte le sorgenti: la sua presenza è indispensabile per poter fondere gli oggetti. Nell' approccio tramite *unifier* la fusione era realizzata mediante un' uguaglianza dei valori assegnati alle variabili omonime, quindi in modo apparentemente simile ad un equijoin, anche se questa soluzione era svantaggiosa dal punto di vista della modularità perché richiedeva di riconoscere i campi del join nelle sorgenti e di assegnare loro nella rule variabili omonime. La soluzione attraverso invenzione dell' oid invece rende irrealizzabile la fusione qualora in una sorgente non sia presente l' attributo che per le altre sorgenti è stato identificato come significativo: si è costretti ad esportare oggetti senza poterli fondere perché gli oid generati per oggetti provenienti da sorgenti diverse non sono compatibili.

MOMIS

La soluzione proposta per MOMIS vuole mantenere sia la capacità di fondere gli oggetti che di creare oggetti complessi.

Per assicurare la fusione delle istanze si definiscono su ogni Classe Globale del Mediatore una o più chiavi. Le chiavi definite vengono poi propagate alle classi della *Gerarchia Estensionale*. La fusione è realizzata facendo degli outerjoin sui valori delle chiavi stesse: questa operazione è eseguita al termine della fase di query processing, quando sono rese disponibili dalle sorgenti le risposte alle subquery ricevute. Infine ogni oggetto finale viene dotato di un proprio oid: questa soluzione consente di conservare la capacità di descrivere anche oggetti complessi.

Il vantaggio di questa soluzione consiste nel superare la forte restrizione che in TSIMMIS richiede che tutte le classi, aventi estensione parzialmente sovrapposta, presentino la **medesima chiave semantica**. Il nostro sistema è in grado di realizzare la fusione utilizzando più chiavi, purchè ogni classe della *Gerarchia Estensionale* sia collegata a classi sorgenti che presentino chiavi omogenee. Ad esempio si consideri un' ipotetica classe così definita:
$C1$: $(\{name, codfisc, rank\}, \{1, 2, 5\})$
Siano name e codfisc due chiavi alternate per la Classe Globale e quindi anche per C1. Supponiamo la relazione fra classi sorgenti e *base extension* evidenziata nella Tabella 5.9.

Le classi da interrogare risultano: `CS_Person`, `Professor`, `University_Worker` ed `Research_Staff`. Se `CS_Person` e `Professor` presentano l' attributo `name` ma non `codfisc`, e viceversa accade per `University_Worker` ed`Research_Staff`, non si possono fondere le istanze delle prime due classi con quelle delle altre due, anche se risulta evidente

Base Extension	1	2	3	4	5
University_Student			x	x	
School_Member			x	x	
CS_Person	x	x		x	
Professor	x	x			
Student				x	
University_Worker	x	x	x		x
Research_Staff	x	x	x		

Tabella 5.9: Esempio: Relazioni fra le Estensioni

dalla tabella delle estensioni che questi due gruppi hanno in comune le *base extension 1, 2*.

Capitolo 6

Progetto del Query Manager

La scelta delle sorgenti da interrogare e la determinazione delle istanze da recuperare sono problemi specifici di ogni Query Manager sviluppato per l' interrogazione di sorgenti autonome ed eterogenee. Queste due funzionalità, se realizzate nel modo opportuno, consentono di completare e risolvere il processo di sintesi delle informazioni: l' esecuzione delle subquery costituisce infatti l' ultima fase dell' intero processo di integrazione e contribuisce direttamente alla completezza e alla correttezza della risposta generata per una interrogazione.

È stato già descritto nel Capitolo 1 come a tutti i livelli dell' architettura I^3 siano necessari servizi che supportino queste funzionalità e come i Servizi di Trasformazione ed Integrazione Semantica, in particolare, debbano comprendere attività di scomposizione delle query, di aggregazione e sintesi dei risultati, e di risoluzione delle incongruenze riscontrate fra i dati raccolti.

In questo Capitolo si illustra come l' architettura di MOMIS, interagendo con ODB-Tools, strumento che fa uso di tecniche di inferenza proprie delle Logiche Descrittive (DLs), realizzi perfettamente le funzionalità necessarie ed inoltre consenta servizi aggiuntivi di espansione ed ottimizzazione semantica delle interrogazioni. Nella Sezione 6.1 si illustra il funzionamento di un generico Query Manager per un' architettura dedicata all' integrazione; nella Sezione 6.2 si mostra dettagliatamente come, potendo disporre della *Gerarchia Estensionale* e di ODB-Tools, sia facilitato il meccanismo di ottimizzazione delle query in ambiente distribuito ed autonomo e di unione e fusione delle istanze in fase di Query Processing. Nella Sezione 6.3 si descrive l' architettura del Query Manager del sistema MOMIS, illustrando come questi utilizza la conoscenza semantica, generata in precedenza, ed ODB-Tools per completare il processo di integrazione.

6.1 Generazione ed Ottimizzazione delle Sub_Query

Il Query Manager sviluppato per un' architettura mirante ad integrare sorgenti autonome ed eterogenee deve risolvere ulteriori problemi oltre a quelli di un DB tradizionale. I comuni servizi DBMS che devono essere offerti riguardano: l' analisi sintattica e semantica delle richieste, i controlli per le eventuali autorizzazioni, la gestione dei cataloghi, delle tabelle di sistema, dei links e della memoria di lavoro e fisica. Viceversa i servizi richiesti per il trattamento di una richiesta di integrazione comportano le seguenti operazioni supplementari:

1. ottimizzazione semantica della query globale
2. scelta delle sorgenti da interrogare
3. generazione delle subquery specifiche per ogni sorgente
4. ottimizzazione semantica locale di ogni subquery
5. unificazione degli oggetti componenti la risposta

Sviluppando l' architettura del Mediatore su quella di un DBMS si possono garantire efficacemente ed efficientemente le operazioni tradizionali ed integrare quelle specifiche attraverso opportuni moduli dedicati. Ad esempio, le fasi di precompilazione di un canned_program, di analisi sintattica e semantica della richiesta, di controllo delle autorizzazioni (qualora si prevedano permessi distinti sulle classi globali per categorie di utenti) possono essere comunemente svolte dai moduli preposti del DBMS; come del resto possono essere supportate sofisticate operazioni di formulazione e valutazione dei piani di accesso alle sorgenti, qualora il Mediatore disponga delle informazioni adeguate sugli indici e sulla loro selettività, sull' ordinamento dei file dati, etc... .

Viceversa le operazioni di ottimizzazione semantica globale, di selezione delle sorgenti, di formulazione delle subquery, di ottimizzazione locale e di fusione dei risultati richiedono la manipolazione della conoscenza intensionale ed estensionale delle sorgenti, prodotta durante la prima fase del processo di integrazione, e perciò devono essere effettuate dai moduli specifici del Mediatore.

In questo lavoro si è volutamente trascurato di descrivere ulteriormente la possibile cooperazione fra DBMS e Mediatore per poter meglio evidenziare le caratteristiche peculiari delle architetture dedicate all' integrazione. Illustriamo brevemente le principali funzionalità di un Query Manager

per applicazioni intersorgenti rispetto alle caratteristiche dell' architettura I^3 e mostriamo come, utilizzando ODB-Tools, siano facilmente supportate le richieste funzioni di ottimizzazione.

6.1.1 Ottimizzazione Semantica Globale

Qualora esistano e siano noti particolari vincoli di integrità intersorgenti, il progettista può cercare di aggiungerli per migliorare il processo di generazione delle risposte.

Come si è descritto nel Capitolo 3, ODB-Tools consente la definizione e la manipolazione dei vincoli sotto forma di rule ODL_{I^3}, tradotte con due descrizioni di tipo in `ocdl`. Il processo di ottimizzazione semantica di una query realizzato da ODB-Tools sfrutta la subsumption: traduce la query in una vista e la posiziona nella tassonomia. Una query la cui descrizione sia sussunta dall' antecedente può essere riformulata aggiungendo o specificando i predicati del conseguente, ottenendo così una formulazione semanticamente equivalente che origina la medesima risposta [44]. L' ottimizzazione semantica favorisce l' esecuzione della query in un tradizionale DB perchè:

- aumenta la possibilità di utilizzare degli indici:

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

  (Q_2) $Storage \sqcap (qty > 150)$
  (R_3) $Storage \sqcap (container\text{: ``A2''}) \leftarrow Storage \sqcap (qty > 100)$
  $(Qopt_2)$ $Storage \sqcap (qty > 150) \sqcap (container\text{: ``A2''})$

- consente di eliminare o modificare dei join impliciti:

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

  (Q_4) $Storage \sqcap (stock.risk > 15) \sqcap (mangedby.level > 5)$
  (R_1) $Storage \sqcap (mangedby\text{:}\, TManager) \leftarrow Storage \sqcap (stock.risk > 10)$
  (S_3) $TManager = Manager \sqcap (level > 10)$
  $(Qopt_4)$ $Storage \sqcap (stock.risk > 15)$

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

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

  (Q_1) $Storage \sqcap (stock.risk > 30)$
  (R_2) $SStorage \leftarrow Storage \sqcap (stock.risk > 20)$
  (S_7) $SStorage = Storage \sqcap [stock\text{:}\, SMaterial]$
  $(Qopt_1)$ $SStorage \sqcap (stock.risk > 30)$

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

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

  (Q_5) $Storage \sqcap (mangedby.level > 5)$
  (R_1) $Storage \sqcap [mangedby\text{:}\, TManager] \leftarrow Storage \sqcap (stock.risk > 10)$
  (S_8) $PStorage = Storage \sqcap (stock.risk > 30)$
  (Q') $Storage \sqcap (mangedby.level > 5)$
  $(Qopt_5)$ $Q' \cup PStorage$

e tutto questo senza creare un rilevante overhead.

Le informazioni necessarie in questa fase sono vincoli di integrità intersorgenti che devono essere resi noti al progettista dai disegnatori o dai gestori delle varie sorgenti o dalle condizioni al contorno del progetto.

La codifica delle informazioni avviene attraverso comuni rule trattabili da ODB-Tools [14, 15].

6.1.2 Scelta delle Sorgenti e Fusione delle Istanze

Questo processo è fondamentale sia in relazione alla correttezza che alla completezza della risposta generata: si vuole determinare il migliore sottoinsieme di subquery che generi una risposta corretta e che sia più completa possibile. La correttezza della risposta è legata alla capacità di reperire le istanze che rispondono alle condizioni espresse nelle query e di fonderle correttamente; la completezza si riferisce alla capacità di interrogare tutte le sorgenti che potenzialmente possono contribuire alla risposta

Data una query globale ottimizzata si deve poter determinare:

1. le sorgenti da interrogare
2. gli oggetti da recuperare
3. le subquery da formulare

Inoltre, se si trattano sorgenti semistrutturate o la cui struttura è considerata soggetta a variazioni e modifiche, come in TSIMMIS [7], questa fase deve determinare la struttura stessa degli oggetti recuperati.

Le informazioni di cui deve disporre un' architettura finalizzata all' integrazione delle informazioni site in sorgenti autonome ed eterogenee in fase di *Query Decomposition* riguardano:

1. la descrizione degli schemi delle sorgenti
2. le relazioni intensionali fra gli schemi
3. le relazioni estensionali fra le classi

La conoscenza intensionale è indispensabile per valutare la correttezza delle risposte prodotte, quella estensionale per assicurare la completezza e la minimalità dell' insieme di sorgenti da interrogare. Si supponga di voler recuperare uno specifico insieme di proprietà di oggetti che soddisfano le condizioni richieste; la correttezza delle istanze trovate è funzione della capacità di verificare i valori degli attributi specificati nelle clausole `where`: se una classe sorgente non è in grado di valutare un predicato, perché tale attributo non è specificato nella sorgente, la correttezza della risposta generata può essere compromessa. Le informazioni estensionali sono preziose non solo per fondere opportunamente le istanze evitando di calcolare join fra elementi appartenenti a sorgenti disgiunte, ma anche per

- minimizzare il numero delle sorgenti da interrogare, assicurando comunque la completezza della risposta. Ad esempio, facendo riferimento alle classi dell' Appendice C, si supponga di voler conoscere i nomi di tutti gli studenti delle facoltà di *"Phisycs"*. Le sorgenti in grado di dare una risposta sono `School_Member`, University_Student e `Student`: in realtà `Student` non possiede l' attributo `faculty`, anzi il processo di arricchimento semantico può aver portato a concludere che tutte le istanze di `Student` hanno `faculty` = *"CS"*: in questo caso la sorgente deve essere scartata. Viceversa `School_Member` ed `University_Student` posseggono l' attributo `faculty` e sono equivalenti dal punto di vista estensionale: per evitare di mandare due subquery equivalenti dal punto di vista della risposta generata si può stabilire di interrogare un' unica sorgente minimizzando tempi e costi.

- fondere istanze per le quali non sia completamente verificata la correttezza, generando oggetti che invece siano corretti. Ad esempio, facendo riferimento alle classi dell' Appendice C, si supponga ora di voler conoscere i corsi (`takes`) seguiti dagli studenti con `faculty` = *"CS"* e che non hanno ancora pagato le tasse `tax` = *"notpayed"*; interrogando `Student` non si riesce a verificare l' attributo `tax`, da `School_Member` non si ricavano informazioni su `takes` e `tax`, da `University_Student` non si hanno informazioni su `takes`. Sapendo però che `School_Member` ed `University_Student`sono equivalenti e che `Student` è incluso in `School_Member`, si deduce che interrogando `Student` ed `University_Student` e fondendo le istanze si riesce a generare la risposta corretta e completa cercata.

Queste informazioni devono essere manipolate in modo da assicurare un veloce processamento delle query; nel seguito si metteranno in evidenza i vantaggi e le funzionalità, derivate dall' utilizzo di ODB-Tools e della *Gerarchia Estensionale*, per affrontare i problemi di scelta delle sorgenti e formulazione delle subquery.

6.1.3 Ottimizzazione Locale

Una volta generate le subquery per ogni sorgente, si può pensare di sfruttare le capacità di ODB-Tools per ottimizzarle ulteriormente. Questa opportunità facilita l' esecuzione della query nelle sorgenti co modelli dei dati meno ricchi purchè siano portati a livello del Mediatore i vincoli sulle classi. È importante osservare che i benefici di un' ottimizzazione perdono di rilevo se la rete non è dedicata e se i dati sono recuperati da sorgenti non eccessivamente popolose. In un contesto di sorgenti di medie e piccole dimensioni collegate in Internet il vantaggio di una buona ottimizzazione può essere significativamente ridotto.

6.2 Query Processing

Per comprendere l' utilità della gerarchia ottenuta col processo descritto nel Capitolo 5, sia in fase di formulazione delle subquery, sia in fase di fusione degli oggetti, faremo prima qualche esempio riferito a query globali formulate sulla classe `University_Person` del Mediatore. In seguito descriveremo l' architettura generale del modulo del Query Manager di MOMIS svolgendo le necessarie valutazioni sui pro ed i contro di questo approccio.

Esempio Consideriamo le 8 seguenti query:

```
Q1   select name, tax, email
     from University_Person
     where faculty="Economics"

Q2   select name, pay
     from University_Person
     where faculty="Economics"
```

```
Q1'  select name, tax
     from University_Person
     where faculty="Economics"

Q2'  select name, pay, rank
     from University_Person
     where faculty="CS"
```

```
     and year="1973"                    and year="1973"

Q3   select name, year          Q3'   select name, year
     from University_Person           from University_Person
     where dept="Maths"               where dept="CS"
     and rank="Professor"             and rank="Professor"

Q4   select name, title         Q4'   select name, title
     from University_Person           from University_Person
     where faculty="Maths"            where dept="CS"
```

a. Il primo passo da compiere consiste nell' eseguire, quando possibile, l' ottimizzazione globale. Supponiamo che sia noto il seguente vincolo:

```
rule RUP forall x in University_Person : x.rank="Professor"
        then x.pay>60.000
```

Le query `Q3` e `Q3'` vengono modificate come segue:

```
Q3   select name, year          Q3'   select name, year
     from University_Person           from University_Person
     where dept="Maths"               where dept="CS"
     and rank="Professor"             and rank="Professor"
     and pay>60.000                   and pay>60.000
```

b. Disponendo della gerarchia integrata mostrata in Figura 5.5, la fase di scelta delle sorgenti e formulazione delle subquery consiste nel riformulare la query, attraverso un' opportuna espansione semantica, sfruttando le definizioni delle classi della *Gerarchia Estensionale*. Le query, scritte come nell' esempio, devono essere riformulate sulla top class (C1) della *Gerarchia Estensionale*; questa contiene, idealmente, tutte le entità delle sorgenti, opportunamente fuse. Per scoprire la classe della gerarchia cui sia più opportuno rivolgere l' interrogazione si ricorre ad ODB-Tools: la query viene tradotta ed inviata all' ottimizzatore di ODB-Tools che la posiziona nella gerarchia (si considerano gli attributi presenti sia nella clausola `select` sia in quella `where`), come è mostrato in Figura 6.1 per la query `Q2`. Se la query è rilevata incoerente, come ad esempio per `Q1`, `Q4` e `Q4'`, si può affermare, senza spedire alcuna subquery, che la risposta è vuota[1]. Una tale risposta significa o che nessuna istanza può soddisfare le condizioni della clausola `where`, o che non è possibile trovare oggetti con tutte le proprietà (attributi) specificate.

[1]In realtà si mostrerà in seguito che `Q4'` non ha risposta vuota, e come evitare di commettere questo errore

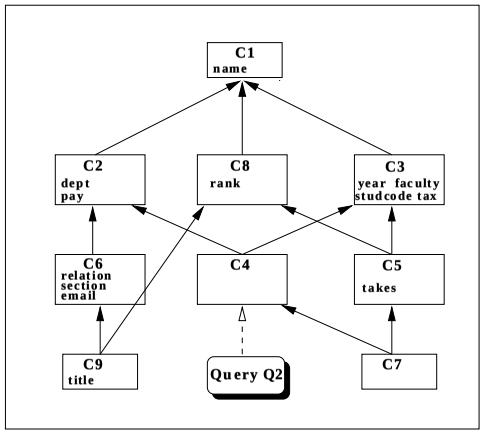


Figura 6.1: Espansione delle Query Q2

c. Nel caso sia stata individuata la classe adeguata nella gerarchia, si deve procedere all' individuazione delle classi delle sorgenti che le sono collegate (si veda un esempio in Figura 5.6). Questa informazione si ricava esaminando le base extension che partecipano alla definizione della stessa nella definizione di *Con* in Tabella 5.8. Scoperte le base extension coinvolte si devono trovare le sorgenti che ne contengono almeno una esaminando la Tabella 5.3 dell' Extensional overlapping. Ad esempio per `Q2` si trova che la classe è C4, le cui base extension sono: $\{2, 7\}$. Analizzando la Tabella 5.3 si trova che le classi cui spedire le subquery sono: `School_Member`, `University_Student`, `CS_Person`, `Student`, `University_Worker`.

d. Si devono poi tradurre le subquery per le sorgenti scelte tenendo conto del fatto che alcuni attributi non sono presenti nella classe per cui devono essere tolti dalle clausole `select` o tolte dalle condizioni `where`. Ad esempio nel caso di `Q2` si avrà:

```
Q2   select name, pay           QUW   select name, pay
     from University_Person           from University_Worker
     where faculty="Economics"
     and year="1973"
```

```
QSM   select name                QCSP   select name
      from School_Member                from CS_Person
      where faculty="Economics"
      and year="1973"

QCSPS select name                QUS    select name
      from Student                      from University_Student
      where year="1973"                 where faculty="Economics"
                                        and year="1973"
```

d. Infine, avendo a disposizione le rule sulle singole classi sorgente, si procede ad eseguire l' ottimizzazione semantica locale. Ad esempio la rule:

```
rule RUP forall x in University_Student : x.faculty="Economics"
         then x.tax>12.000
```

modifica la query per `University_Student`:

```
QUS    select name
       from University_Student
       where faculty="Economics"
       and year="1973"
       and tax>12.000
```

6.2.1 Osservazioni

Una cosa che non è stata approfondita riguarda la specificazione degli attributi di default [3] nella fusione della gerarchia. Per *attributi di default* si intendono quegli attributi che, pur non essendo propri di alcune classi sorgente, ammettono per esse un valore di default che è implicito nella loro semantica. Questo valore viene assegnato dal progettista durante il processo di integrazione semantica nella definizione della Mapping table delle classi globali sulla base di specifiche conoscenze inter ed intra sorgenti. Un esempio si ha per l' attributo *faculty* della classe globale `University_Person`: la gerarchia di `CS_Person` non contiene l' informazione sulla facoltà perchè è implicito il fatto che tutte le entità in `CS_Person` facciano riferimento alla facoltà di *CS*.

L' importanza di non trascurare questo tipo di informazione appare nell' esempio di `Q2`: la condizione *faculty="Economics"* non può essere posta alle classi `CS_Person` e `Student` perché non hanno l' attributo *faculty*. Le due subquery `QCSP` e `QCSPS` sono comunque formulate senza la condizione, ma in questo caso la risposta fornita sarà errata. Un errore analogo è generato

per `Q4'`. Per essa il sistema produce immediatamente una risposta di tipo vuoto, mentre in questo caso sarebbe corretto interrogare C9, fra le cui classi di riferimento vi è appunto `Professor`.

A questo problema si deve porre rimedio già dal momento di generazione della gerarchia. Si deve prevedere l' inserimento delle informazioni sui default attribute in fase di definizione dell' Intensional overlapping. Ad esempio possiamo immaginare di precisare che `CS_Person`, `Professor` e `Student` presentino il default attribute *faculty*. La Tabella 5.4 dell e proprietà intensionali risulta conseguentemente modificata e quindi anche le relazioni estensioni-attributi cambiano come appare in Tabella 6.1.

Base Extension	1	2	3	4	5	6	7	8	9	10	11
name	x	x	x	x	x	x	x	x	x	x	x
dept	x	x	x		x	x	x				
pay	x	x	x		x	x	x				
fac	**x**	x		**x**		**x**	x	x	x	x	
year		x					x	x	x	x	
tax		x					x	x	x	x	
rank	x	x						x			
email	x				x						
relation	x				x						
takes		x						x			
title	x										
section	x				x						
studcode		x					x	x	x	x	

Tabella 6.1: Relazioni Estensioni-Attributi Modificata

Le conseguenze di questa modifica risultano in fase di generazione delle classi della gerarchia integrata in Tabella 6.2 ed in Figura 6.2, dove viene mostrata la *Gerarchia Estensionale* modificata.

Una volta ottenuta questa gerarchia si specifica che tutte le classi, le cui estensioni comprendono solo *base extension* per cui l' attributo è di default, cioè $\{1,4,6\}$ o un sottoinsieme, vengono arricchite col vincolo che *faculty="CS"* attraverso una rule. In questo caso la classe su cui imporre il vincolo è C9.

Un' altra importante osservazione relativa alla fase **d.** riguarda la necessità di reperire per tutte le istanze le chiavi necessarie alla fusione.

$$Con: = \left\{ \begin{array}{l} C1: (\{name\}, \{1,2,3,4,5,6,7,8,9,10,11\}), \\ C2: (\{name, dept, pay\}, \{1,2,3,5,6,7\}), \\ C3: (\{name, faculty, year, tax, studcode\}, \{2,7,8,9,10\}), \\ C4: (\{name, dept, pay, faculty, year, tax, studcode\}, \{2,7\}), \\ C5: (\{name, faculty, year, tax, rank, takes, studcode\}, \{2,8\}), \\ C6: (\{name, dept, pay, email, relation, section\}, \{1,5\}), \\ C7: (\{name, dept, pay, faculty, year, tax, rank, takes, studcode\}, \{2\}), \\ C8: (\{name, rank\}, \{1,2,8\}), \\ C9: (\{name, dept, pay, \textbf{faculty}, rank, email, relation, title, section\}, \{1\}), \\ C10: (\{name, faculty\}, \{1,2,4,6,7,8,9,10\}), \\ C11: (\{name, dept, pay, faculty\}, \{1,2,6,7\}) \end{array} \right\}$$

Tabella 6.2: Insieme delle Classi della Gerarchia Estensionale Modificato

Nell' esempio relativo al cluster `University_Person` ci siamo limitati al caso in cui la chiave di tutte le sorgenti fosse l' attributo *name*. Questa situazione è scarsamente verosimile, mentre risulta maggiormente credibile che fra le sorgenti le chiavi siano più di una (ad esempio *codfisc, matr, codlav*). Il modo in cui vengono riformulate le subquery non assicura che le sorgenti vengano interrogate recuperando le chiavi necessarie, per questa ragione occorre prevedere di inserire nella clausola `select` delle subquery i campi relativi alle chiavi.

Un aspetto da non trascurare, per evitare di connettersi con un numero eccessivo di sorgenti via rete e di eseguire join inutili e costosi, riguarda la capacità di riconoscere le subquery *ridondanti*, cioè la cui risposta sia un sottoinsieme delle risposte ottenute dalle altre, non solo da un punto di vista estensionale, ma anche in relazione alle proprietà recuperate. Un esempio in questo caso si ricava esaminando la query `Q2'`. Essa viene posizionata da ODB-Tools al di sotto della classe C7, come appare in Figura 6.3. Essa è collegata alla *base extension* {2} che corrisponde alle classi sorgente `School_Member`, `University_Student`, `University_Worker`, `CS_Person`, `Student`. Le subquery che vengono formulate sono:

```
Q2'   select name, pay, rank
      from University_Person
      where faculty="CS"
      and year="1973"

QSM   select name
      from School_Member

QUW   select name, pay
      from University_Worker

QCSP  select name
      from CS_Person
```

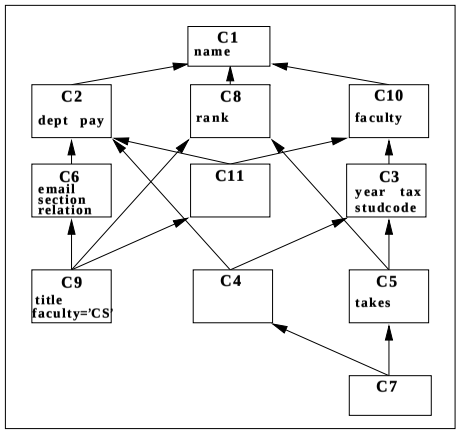



Figura 6.2: Gerarchia Integrata Modificata

```
      where faculty="CS"
      and year="1973"

QCSPS select name, rank
      from Student
      where year="1973"

QUS   select name
      from University_Student
      where faculty="CS"
      and year="1973"
```

Sulla base della conoscenza estensionale e dagli attributi specificati dalla `select` si evince che `QUS` e `QSM` aggiungono informazioni identiche. Il sistema dovrebbe essere in grado di comprendere questo fatto, usufruendo delle informazioni precedentemente codificate, ed evitare le subquery a `School_Member` o ad `University_Student`.

Dopo aver generato le subquery, per capire se una subquery è *ridondante* il sistema deve confrontare le clausole `select` e `where` con quelle delle altre subquery per cercare quelle che recuperano un sottoinsieme delle proprietà specificate, senza verificare ulteriori condizioni. Per ogni subquery che verifica questa condizione si deve poi verificare se valga la medesima proprietà per le estensioni. Una proposta per risolvere il problema si basa sull' utilizzo di ODB-Tools e delle rule estensionali. Ad esempio, date le subquery relative alla query `Q2'`, si evidenzia che la clausola `select` di `QUS` è più generale di quella di `QSM`, avendo in input all' ottimizzatore `QUS` e le rule estensiona-

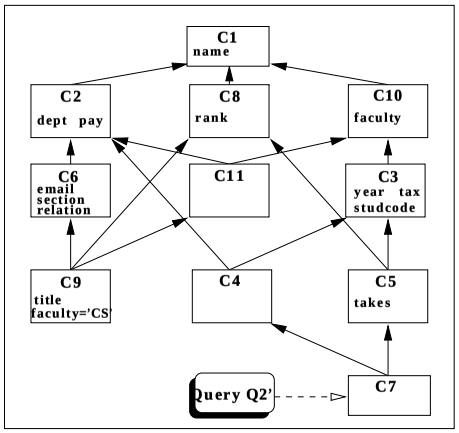


Figura 6.3: Espansione della Query Q2'

li di inclusione ed equivalenza si evidenzia come `QUS` sia sussunta anche da `School_Member`, significando che `QUS` può essere considerata superflua.

In Figura 6.4 si mostra come il sistema mostra l' equivalenza delle due query.

6.3 Architettura del Query Manager

L' architettura ipotizzabile per il Query Manager è illustrata in Figura 6.5.

I rettangoli esterni al Query Manager rappresentano le informazioni generate in fase di integrazione dalle metodologie di generazione dei cluster e delle gerarchie estensionali. I due moduli interni al Query Manager eseguono rispettivamente le fasi di generazione delle subquery e di *eliminazione* delle query ridondanti descritte nella Sezione 6.2. ODB-Tools, disponendo della gerarchia estensionale, determina la classe da interrogare; il modulo detto **Subquery Generator** determina le sorgenti coinvolte dalla query disponendo delle informazioni mantenute nell' insieme *Con*, per determinare le base extension interessate, e le potenziali classi sorgenti, sulla base delle informazioni mantenute nella tabella dell' Extensional overlapping. Una volta determinate le classi da interrogare, disponendo della conoscenza intensionale, traduce la query globale nelle rispettive subquery. Il secondo modulo, **Subquery**

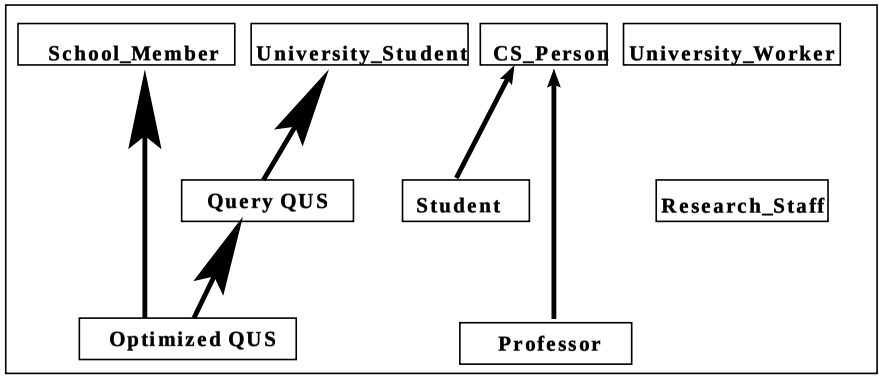


Figura 6.4: Ottimizzazione della Query QUS

Figura 6.5: Architettura del Query Manager

Selector, interagisce con ODB-Tools, come descritto nella Sezione 6.2, per eliminare le subquery equivalenti, se sono presenti.

Conclusioni

Lo studio compiuto con questa tesi ha completato la progettazione di MOMIS, un' architettura per l' integrazione di informazioni site in sorgenti autonome ed eterogenee. L' area di ricerca in cui si inserisce il progetto, l' Integrazione Intelligente di Informazioni, è piuttosto recente negli obiettivi che si prefigge, ma si avvale di tecniche di Intelligenza Artificiale ampiamente consolidate.

Obiettivo di MOMIS è fornire l' accesso a sorgenti di informazione prestabilite per **integrare, analizzare e sintetizzare** i dati che esse mettono a disposizione, aumentando il valore complessivo delle informazioni ed eliminando incongruenze e duplicati. In particolare con questo lavoro si è affrontato il problema del **Query Processing**, essendo realizzata in un momento precedente l' integrazione degli schemi delle sorgenti. L' importanza della fase di decomposizione delle interrogazione e di ricomposizione dei risultati è strategica in tutti i sistemi che non replicano le informazioni, poichè da essa dipende la correttezza e la comprensibilità dei risultati prodotti, oltre al tempo richiesto per produrli.

La proposta che è stata presentata si basa sulla manipolazione della conoscenza estensionale. Dal punto di vista dell' integrazione queste informazioni sono introdotte e interpretate in maniera semi automatica al momento della progettazione dello schema del Mediatore, grazie all' impiego di strumenti sviluppati nell' area dell' *Intelligenza Artificiale*. Dal punto di vista dell' interrogazione questa conoscenza è ampiamente utilizzata dal *Query Manager*: con l' ausilio delle tecniche di inferenza a disposizione, si riesce ad ottimizzare l' insieme delle query locali per effettuare una scelta "ragionata" delle sorgenti da interrogare, formulando un insieme minimo di interrogazioni, che preservino la completezza e la correttezza della risposta.

I risultati ottenuti, confrontati con i sistemi presentati in letteratura, hanno evidenziato l' originalità e la funzionalità della proposta: l' idea di utilizzare la conoscenza estensionale per ottimizzare la fase di interrogazione

non è solo assente in tutti gli altri sistemi, ma consente di migliorare sensibilmente l' integrazione dei risultati. Un altro aspetto positivo che emerge dal confronto con gli altri sistemi si evidenzia nella capacità di sostenere il progettista nella fase di integrazione degli schemi ed introduzione della conoscenza estensionale. Il processo è realizzato in maniera semi automatica, preservando in questo modo la componente di conoscenza a disposizione del progettista, ma supportando tutto il processo di elaborazione delle informazioni ed integrazione con sofisticati strumenti che facilitino e rendano più veloce il processo.

Nella realizzazione degli obiettivi si è rivelato *strategico* il componente *intelligente* utilizzato: sebbene la presenza del progettista rimanga essenziale, le tecniche di Intelligenza Artificiale, implementate negli ODB-Tools, sono state usate ampliamente in tutte le fasi, dall' integrazione, all' ottimizzazione semantica ed alla riformulazione delle interrogazioni.

L' architettura di MOMIS, completata attraverso la specifica del modulo del Query Manager, soddisfa i requisiti specifici delle architetture di mediazione così come posti nel Capitolo 1. Ulteriori approfondimenti dovranno essere svolti in almeno tre direzioni. In primo luogo è necessario prevedere di aumentare le potenzialità del sistema "agganciando" MOMIS ad un DBMS che fornisca le operazioni classiche di gestione degli indici, degli archivi, efficienza nell' esecuzione dei join, etc... . Inoltre rimane da approfondire l' interazione fra l' utente ed il sistema in fase di interrogazione: sfruttando la semantica introdotta in fase di integrazione è infatti possibile arricchire questa fase consentendo al sistema di suggerire l' eventuale introduzione o rilassamento di vincoli, in modo da ottenere una risposta più adeguata alle richieste dell' utente. 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 è riportato, a fianco del termine, il corrispettivo inglese, quando la traduzione dal termine originale in 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 organizzazione responsabile di un servizio.
- database = risorsa che comprende un insieme di dati con uno schema descrittivo.
- warehouse = database che contiene o dá accesso a dati selezionati, astratti e integrati da una molteplicitá di sorgenti. Tipicamente ridondante rispetto alle sorgenti di dati.
- base di conoscenza = risorsa comprendente un insieme di conoscenze trattabili in modo automatico, spesso nella forma di regole e di metadata; permettono l'accesso alle risorse.
- simulazione = risorsa in grado di fare proiezioni future sui dati e generare nuove informazioni, basata su un modello.
- amministrazione della transazione = assicurare che la consistenza temporale del database non sia compromessa dagli aggiornamenti.
- impatto della transazione = riporta le risorse che sono state coinvolte in un aggiornamento.
- schema = lista dedlle relazioni, degli attributi e, quando possibile, degli oggetti, delle regole, e dei metadata di un database. Costituisce la base dell'ontologia della risorsa.
- dizionario = lista dei termini, fa parte dell'ontologia.
- modello del database = descrizione formalizzata della risorsa database, che include lo schema.
- interoperabilitá = capacitá di interoperare.
- eterogeneitá = incompatibilitá trovate tra risorse e servizi sviluppati autonamamente, che vanno dalla paittaforma utilizzata, sistema operativo, modello dei dati, alla semantica, ontologia,...
- costo = prezzo per fornire un servizio o un accesso ad un oggetto.
- database deduttivo = database in grado di utilizzare regole logiche per trattare i dati.
- regola = affermazione logica, unitá della conoscenza trattabile in modo automatico.

- sistema di amministrazione delle regole = software indipendente dal dominio che raccoglie, seleziona ed agisce sulle regole.
- database attivo = database in grado di reagire a determinati eventi.
- dato virtuale = dato rappresentato attraverso referenze e procedure.
- stato = istanza o versione di una base di dati o informazioni.
- cambiamento di stato = stato successivo ad una azione di aggiornamento, inserimento o cancellazione.
- vista = sottoinsieme di un database, sottoposto a limiti, e ristrutturato.
- server di oggetti = fornisce dati oggetto.
- gerarchia = struttura di un modello che assegna ogni oggetto ad un livello, e definisce per ogni oggetto l'oggetto da cui deriva.
- network = struttura di un modello che fa uso di relazioni relativamente libere tra oggetti.
- ristrutturare = dare una struttura diversa ai dati seguendo un modello differente dall'originale.
- livello = categorizzazione concettuale , dove gli oggetti di un livello inferiore dipendono da un antenato di livello superiore.
- antenato (ancestor) = oggetto di livello superiore, dal quale derivano attributi ereditabili.
- oggetto root = oggetto da cui tutti gli altri derivano, all'interno di una gerarchia.
- datawarehouse = deposito di dati integrati provenienti da una molteplicitá di risorse.
- deposito di metadata = database che contiene metadata o metainformazioni.

A.4 Ontologia

- Ontologia = descrizione particolareggiata di una concettualizzazione, i.e. l'insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti, spesso ambigui tra domini diversi.
- concetto = definisce una astrazione o una aggregazione di oggetti per il cliente.
- semantico = che si riferisce al significato di un termine, espresso come un insieme di relazioni.
- sintattico = che si riferisce al formato di un termine, espresso come un insieme di limitazioni.
- classe = definisce metaconoscenze come metodi, attributi, ereditarietá, per gli oggetti in essa istanziati.
- relazione = collegamento tra termini, come *is-a*, *part-of*,...
- ontologia unita (merged) = ontologia creata combinando diverse ontologie, ottenuta mettendole in relazione tra loro (mapping).
- ontologia condivisa = sottoinsieme di diverse ontologie condiviso da una molteplicitá di utenti.
- comparatore di ontologie = strumento per determinare relazioni tra ontologie, utilizzato per determinare le regole necessarie per la loro integrazione.
- mapping tra ontologie = trasformazione dei termini tra le ontologie, attraverso regole di accoppiamento, utilizzato per collegare utenti e risorse.
- regole di accoppiamento (matching rules) = dichiarazioni per definire l'equivalenza tra termini di domini diversi.
- trasformazione dello schema = adattamento dello schema ad un'altra ontologia.
- editing = trattamneto di un testo per assicurarne la conformitá ad una ontologia.
- algebra dell'ontologia = insieme delle operazioni per definire relazioni tra ontologie.

- consistenza temporale = é raggiunta se tutti i dati si riferiscono alla stessa istanza temporale ed utilizzano la stessa granularitá temporale.
- specifico ad un dominio = relativo ad un singolo dominio, presuppone l'assenza di incompatibilitá semantiche.
- indipendente dal dominio = software, strumento o conoscenza globale applicabile ad una molteplicitá di domini.

Appendice B

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

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

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

Si veda ora la sintassi proposta per la dichiarazione dei mapping fra i nomi delle classi globali e quelli nelle sorgenti:

⟨attr_dcl⟩ ::= [**readonly**] **attribute**
⟨domain_type⟩ ⟨attribute_name⟩
[⟨fixed_array_size⟩] [⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ ::= **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ ::= ⟨rule⟩ | ⟨rule⟩,⟨rule_list⟩
⟨rule⟩ ::= ⟨local_attr_name⟩ | '⟨identifier⟩'
⟨and_expression⟩ | ⟨or_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and** ⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **and** ⟨and_list⟩
⟨or_expression⟩ ::= **(** ⟨local_attr_name⟩ **or** ⟨or_list⟩ **)**
⟨or_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **or** ⟨or_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩.⟨class_name⟩.⟨attribute_name⟩
...

La seguente sintassi è stata introdotta per definire le relazioni terminologiche:

⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** | ⟨relationship_dcl⟩**;** ⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
⟨relationship_type⟩ ::= **syn** | **bt** | **nt** | **rt**
...

Di seguito si mostra la sintassi per la dichiarazione di vincoli e rule estensionali:

⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_pre⟩ **then** ⟨rule_post⟩
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨rule_identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_identifier⟩ ::= ⟨identifier⟩ | **(** ⟨identifier⟩ **and** ⟨identifier⟩ **)**
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** | ⟨rule_body⟩ |
⟨rule_body_list⟩ **and** ⟨rule_body⟩ |
⟨rule_body_list⟩ **and (** ⟨rule_body_list⟩ **)**
⟨rule_body⟩ ::= ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in** ⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
⟨rule_const_op⟩ ::= = | ≥ | ≤ | > | <
⟨rule_cast⟩ ::= (⟨simple_type_spec⟩)
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩.⟨dotted_name⟩
⟨forall⟩ ::= **for all** | **forall**

Appendice C

Esempio in ODL$_{I^3}$

Di seguito é riportata la descrizione, attraverso il linguaggio ODL$_{I^3}$, dell'esempio di riferimento. Si introducono inoltre le rule estensionali relative all' insieme di classi che formano il cluster `University_Person`.

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

  attribute integer pay; };
```

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

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



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

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

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

```
COMPUTER_SCIENCE source:
```

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

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

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

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

```
interface Location
( source object Computer_Science
  extent Locations
```

```
interface Course
( source object Computer_Science
  extent Courses
```

```
  keys city, street, county, number)
{ attribute string city;
  attribute string street;
  attribute string county;
  attribute integer number; };
```

```
Tax_Position source:

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

Le rule estensionali sul cluster `University_Person`, così come definito nel Capitolo 4 sono:

```
rule RE1a forall x in School_Member then x in University_Student;

rule RE1b forall x in University_Student then x in School_Member;

rule RE2 forall x in Research_Staff then x in University_Worker;

rule RE3 forall x in Student then x in School_Member;

rule RE4 forall x in Professor then x in Research_Staff;

rule RE5 forall x in (Professor and School_Member)
                                    then x in bottom;

rule RE6 forall x in (Research_Staff and University_Student)
                                    then x in bottom;

rule RE7 forall x in (Research_Staff and Student)
                                    then x in bottom;
```



```
  key course_name )
{ attribute string course_name;
  attribute Professor taught_by; };
```

Bibliografia

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

[2] Simone Montanari. Un approccio intelligente all' integrazione di sorgenti eterogenee di informazione, 1996-1997.

[3] S.Bergamaschi, S.Castano, S.De Capitani di Vimercati, S.Montanari, and M.Vincini. An intelligent approach to information integration. *Accepted for: Formal Ontology in Information Systems FOIS98.*

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

[5] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

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

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

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

[9] 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.

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

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

[12] *"The Object Database Standard. Release 2.0"*. "Morgan Kauffman Inc.", "1998".

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

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

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

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

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

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

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

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

[21] 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.

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

[23] Y.Sagiv J.Ullman D.Quass, A.Rajaraman and J.Widom. Querying semistructured heterogeneous informations. Technical report, Stanford University, 1996.

[24] 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.

[25] 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.

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

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

[28] M. Ginsberg, editor. *Essentials of Artificial Intelligence*. Morgan Kauffman Pubblisher, Inc, 1993.

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

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

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

[32] 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.

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

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

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

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

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

[38] 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.

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

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

[41] S.Castano and Valeria De Antonellis. Semantic dictionary design for database interoperability. 1997.

[42] 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.

[43] R. Hull and M. Yoshikawa. Ilog: Declarative creation and manipulation of object identifiers (extended version). In *Proceedings of the 16th VLDB Conference*, Brisbane, Australia, August 1990.

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