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

SI-Designer: un tool per l'integrazione di sorgenti distribuite ed eterogenee

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Rossano Guidetti

Correlatore
Ing. Alberto Corni

Controrelatore
Chiar.mo Prof. Flavio Bonfatti

Anno Accademico 1999 - 2000

Parole chiave:

Tool grafico di integrazione
Integrazione Intelligente delle Informazioni
Mediatore
Schema Globale
Mapping table

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi per l'aiuto fornito alla realizzazione della presente tesi.
Un particolare ringraziamento va all'Ing. Alberto Corni e ai componenti del gruppo MOMIS per la costante disponibilità e i preziosi consigli che mi hanno fornito.

Indice

Elenco delle figure

Capitolo 1

L'Integrazione Intelligente ed il sistema MOMIS

Grazie alla crescita delle sorgenti di dati e allo sviluppo delle tecnologie di comunicazione, tanto all'interno di un'azienda quanto sulla rete, oggi è possibile accedere facilmente ad una grande quantità di dati. Parallelamente tuttavia è aumentata la difficoltà nel recuperare i dati in tempi e modi accettabili. Questo è dovuto soprattutto alla grande eterogeneità delle sorgenti, sia per quanto riguarda la natura dei dati (testi, immagini, ecc.), sia per il modo in cui vengono descritti e organizzati nelle sorgenti.
A questi problemi si aggiunge l'*information overload*, vale a dire la difficoltà da parte dell'utente di selezionare ed isolare i dati veramente interessanti dalla enorme mole disponibile che aumenta in continuazione ed in modo disordinato.

La variegata quantità di modelli e schemi dei dati utilizzati per la modellazione delle informazioni crea una eterogeneità semantica (ovvero logica) non risolvibile utilizzando gli standard attuali di comunicazione e modalità di accesso ai dati (TPC/IP, ODBC, SQL, ecc.).
Un'importante area, sia di ricerca che di applicazione, riguarda l'integrazione di DataBase eterogenei ed i *datawarehouse* (magazzini di dati). Questi lavori studiano la possibilità di materializzare, presso l'utente finale, delle viste su porzioni di sorgenti, replicando fisicamente i dati ed affidandosi a complicati algoritmi di mantenimento per garantirne la consistenza.

1.1 L'Integrazione Intelligente delle Informazioni

Con il termine *Integrazione di Informazioni* (I^2) [1] si indicano in letteratura tutti quei sistemi che, basandosi sulle *descrizioni* dei dati, combinano tra loro informazioni provenienti da diverse sorgenti (o parti selezionate di esse) senza quindi

dover ricorrere alla duplicazione fisica dei dati. Per ottenere i risultati voluti, l'integrazione richiede *conoscenza* ed *intelligenza* volte all'individuazione delle sorgenti e dei dati nonchè alla loro fusione e sintesi. Quando l'Integrazione di Informazioni fa uso di tecniche di Intelligenza Artificiale si parla allora di Integrazione Intelligente di Informazioni (*Intelligent Integration of Information*). Questa forma di integrazione si distingue dalle tecniche tradizionali per il fatto che, prima di aggregare le informazioni, il sistema ne aumenta il valore analizzando i dati che le rappresentano, ottenendo una integrazione migliore.

1.1.1 Il programma I^3

[1] Dal 1992 è operativo il programma I^3 [2]: un progetto di ricerca fondato e sponsorizzato dall'ARPA (*Advanced Research Projects Agency*, agenzia che fa capo al Dipartimento di Difesa degli Stati Uniti) che si prefigge l'obiettivo di individuare architetture modulari, sviluppabili secondo i principi definiti da uno standard che ponga le basi dei servizi di un qualsiasi sistema intelligente di integrazione automatica di sorgenti eterogenee ed abbassi i costi di sviluppo e mantenimento. Questo permette di ovviare ai problemi di realizzazione, manutenzione, adattabilità che insorgono nella costruzione di supersistemi che comprendono una notevole quantità di sorgenti non correlate semanticamente.
Per la realizzazione di supersistemi molto ampi, è stata proposta una suddivisione dei servizi e delle risorse che si articola su due dimensioni:

- orizzontalmente su 3 livelli:
  - livello utente;
  - livello intermedio, in cui sono presenti i moduli che fanno uso di tecniche di IA;
  - livello dati, dove vengono gestite le risorse di dati;
- verticalmente su molti domini: nei vari livelli i domini non sono strettamente connessi, ma si scambiano dati ed informazioni la cui combinazione avviene a livello dell'utilizzatore, riducendo la complessità totale del sistema e permettendo lo sviluppo di applicazioni con finalità diverse.

I^3 si concentra sul livello intermedio della partizione, quello che media tra gli utenti e le sorgenti. In questo livello sono presenti vari moduli, tra i quali si possono evidenziare:

[1] Nell'Appendice A è presente un glossario dei termini comunemente usati in ambito I^3, con lo scopo di spiegare quei termini che dovessero risultare ambigui o poco chiari.

- **Facilitator** e **Mediator** (le differenze tra i due sono sottili ed ancora ambigue in letteratura): ricercano le fonti *interessanti* e combinano i dati da esse ricevuti;

- **Query Processor**: riformula le query aumentando le loro probabilità di successo;

- **Data Miner**: analizza i dati per estrarre informazioni intensionali implicite.

1.1.2 Architettura di riferimento per sistemi I^3

L'architettura di riferimento per la realizzazione di sistemi integratori definita dal programma I^3 è composta da cinque famiglie di attività omogenee, illustrate in Figura 1.1 unitamente ai loro legami. Più in dettaglio, percorrendo l'asse orizzontale, si nota il rapporto tra i servizi di Coordinamento ed Amministrazione, che hanno il compito di individuare le sorgenti e i servizi che occorrono per soddisfare una qualsiasi richiesta dell'utente o di altri servizi. I servizi Ausiliari invece, responsabili dei servizi di arricchimento semantico delle sorgenti, forniscono le funzionalità di supporto.
Seguendo l'asse verticale si capisce come è stato progettato lo scambio di informazioni nel sistema: i servizi di Wrapping provvedono ad estrarre le informazioni dalle sorgenti, queste saranno poi integrate dai servizi di Integrazione e Trasformazione semantica, per poi essere infine passate ai servizi di Coordinamento che ne avevano fatto richiesta.

Nell'architettura di riferimento I^3 , i **Servizi di Coordinamento** hanno un'importanza centrale. Grazie alle funzionalità messe a loro disposizione dalle altre famiglie di servizi, riescono a coordinare le operazioni da eseguire tanto in fase di progettazione dei collegamenti fra livelli e domini informativi quanto in fase di esecuzione in tempo reale di una richiesta dell'utente.

I **Servizi di Amministrazione** sono utilizzati da quelli di Coordinamento per individuare le sorgenti, determinarne le capacità, creare ed interpretare i *template* (strutture dati che descrivono i servizi e i moduli da utilizzare per trattare un insieme di sorgenti). I template servono per ridurre al minimo le possibilità di decisione del sistema, consentendo di definire a priori le azioni da eseguire a fronte di una determinata richiesta. In alternativa ad essi si possono utilizzare le *yellow pages*: servizi di directory che mantengono le informazioni sul contenuto delle sorgenti e sul loro stato (attiva, inattiva, occupata), consentendo ai moduli preposti di inviare la richiesta di informazioni alla sorgente giusta o, se non fosse disponibile, ad una equivalente.

I **Servizi di Integrazione e Trasformazione Semantica** hanno come input una o più sorgenti dati, tradotte dai servizi di Wrapping, e generano una vista

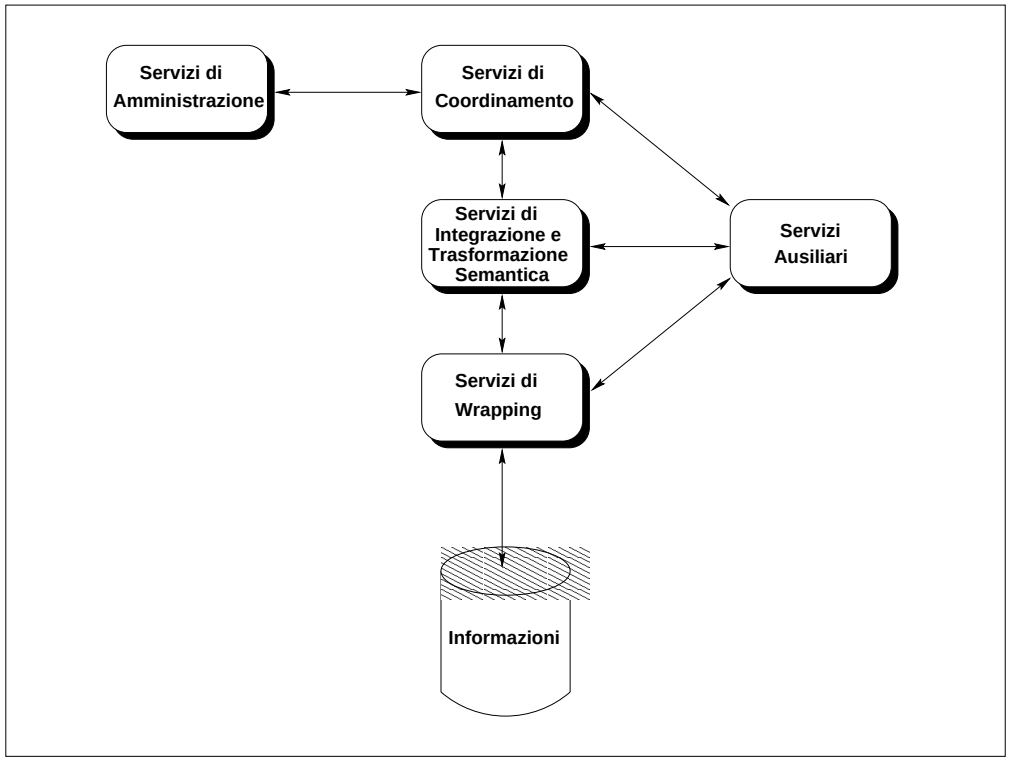


Figura 1.1: Diagramma dei servizi I^3

integrata o trasformata di queste informazioni. Essi vengono indicati spesso come servizi di mediazione. I principali sono:

- *Servizi di Integrazione di Schemi*: creano il vocabolario e le ontologie condivise dalle sorgenti, integrano gli schemi in una vista globale, mantengono il mapping tra schemi globali e sorgenti;

- *Servizi di Integrazione di Informazioni*: aggregano, riassumono ed astraggono le risposte di più sottoquery per fornire un'unica risposta alla query originale;

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

I **Servizi di Wrapping** fungono da interfaccia tra il sistema integratore e le singole sorgenti rendendo omogenee le informazioni provenienti dalle sorgenti eterogenee traducendole in un linguaggio standard. Il processo di realizzazione di

un wrapper dovrebbe essere standardizzato in modo da essere riutilizzato per altre fonti di informazione.

I **Servizi Ausiliari** aumentano le funzionalità degli altri servizi. Essi vanno dai semplici servizi di monitoraggio del sistema ai servizi di propagazione degli aggiornamenti (mantenimento della consistenza dei dati), dai servizi di arricchimento semantico (già accennati in precedenza) a quelli di ottimizzazione.

Concludendo questa rapida descrizione dell'architettura di riferimento per sistemi I^3 è opportuno sottolineare come non sia necessario che ogni sistema integratore di informazioni comprenda l'intero insieme di funzionalità descritte, anzi sarà molto improbabile che ciò avvenga vista la quantità di servizi che occorrerebbe implementare. É possibile realizzare un sistema integratore funzionante selezionando solo i servizi necessari allo svolgimento del compito che si prefigge.

1.1.3 Il mediatore

Questa tesi fa parte di un progetto di ricerca più ampio che ha come obiettivo la progettazione e realizzazione di un *mediatore* [3]. Tale modulo si pone a livello intermedio tra l'utente e le risorse dei dati nell'architettura precedentemente descritta. Secondo la definizione proposta in [4] "un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore . . . Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno o, al più, pochi esperti."
Un mediatore deve assolvere ai seguenti compiti:

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

La prima ipotesi che è stata formulata per restringere il campo applicativo del sistema da progettare (e, di conseguenza, il campo dei problemi a cui dare risposta) è di avere a che fare, per il momento, esclusivamente con sorgenti di dati testuali, strutturate o semistrutturate: quindi basi di dati relazionali, ad oggetti, file di testo, ma anche sorgenti XML che sono sorgenti semistrutturate standard.

L'approccio architetturale scelto consta principalmente di 3 livelli:

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

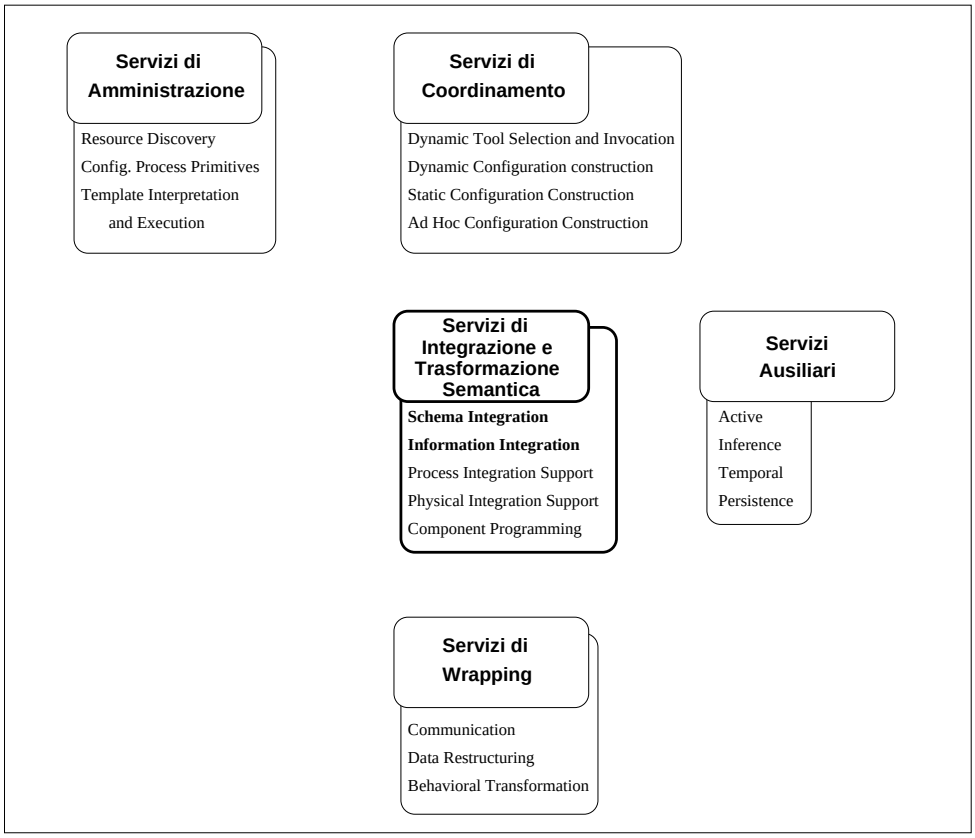


Figura 1.2: Servizi I^3 presenti nel mediatore

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

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

I servizi esaminati per la progettazione del mediatore sono quelli mostrati in Figura 1.2, evidenziati in neretto, che facilitano sia l'integrazione degli schemi sia l'integrazione delle informazioni. Parallelamente a questa impostazione, il progetto si vuole distaccare dall'approccio *strutturale* (sintattico), tuttora dominante tra i sistemi presenti sul mercato.

L'approccio sintattico è caratterizzato dall'utilizzo di un *self-describing model* per rappresentare gli oggetti da integrare, limitando così l'uso delle informazioni

semantiche a delle regole predefinite dall'operatore. In pratica, il sistema non conosce a priori la semantica di un oggetto recuperato da una sorgente (di questa dunque non possiede alcuno schema descrittivo), ma è l'oggetto stesso che, attraverso delle etichette, si autodescrive specificando tutte le volte, per ogni suo singolo campo, il significato che ad esso è associato. Le caratteristiche di questo approccio quindi sono le seguenti:

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

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

Altri progetti, e tra questi quello descritto nel presente capitolo, seguono invece un approccio definito *semantico*. Tale approccio ha le seguenti caratteristiche:

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

- le informazioni semantiche sono estratte dagli schemi concettuali;

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

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

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

1.2 Problematiche da affrontare

Pur avendo a disposizione gli schemi concettuali delle varie sorgenti, non è certamente un compito banale individuare i concetti comuni ad esse e le relazioni che

possono legarli, ne tantomeno è banale realizzare una loro coerente integrazione. Tralasciando le differenze tra i sistemi fisici (alle quali dovrebbero pensare i moduli wrapper) i problemi che si è dovuto risolvere, o con i quali si è dovuto giungere a compromessi, sono essenzialmente di due tipi:

1. **problemi ontologici**;
2. **problemi semantici**.

1.2.1 Problemi Ontologici

Nell'Appendice A si può trovare una definizione di ontologia: qui ci si limita solamente a riportare una semplice classificazione delle ontologie (mutuata da [5, 6]) per inquadrare l'ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate) sono essenzialmente le seguenti:

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

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

1.2.2 Problemi Semantici

Pur ipotizzando che anche sorgenti diverse condividano una visione similare del problema da modellare, e quindi un insieme di concetti comuni, niente assicura che i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti, ne tantomeno le stesse strutture dati. Anzi, visto che le diverse sorgenti sono state progettate e modellate da persone differenti, è molto improbabile che queste persone condividano la stessa concettualizzazione del mondo esterno: in altre parole, non esiste nella realtà una semantica univoca a cui chiunque possa riferirsi.

Ad esempio se la persona P1 disegna una fonte di informazioni (DB1) e un'altra persona P2 disegna la stessa fonte DB2, le due basi di dati probabilmente presenteranno differenze semantiche: ad esempio, le coppie sposate potranno essere rappresentate in DB1 usando degli oggetti della classe COPPIE, con attributi MARITO e MOGLIE, mentre in DB2 potrebbe esserci una classe PERSONA con un attributo SPOSATO_A.
Come riportato in [7] la causa principale delle differenze semantiche è identificabile nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma non è l'unica. Una medesima realtà può essere rappresentata su DBMS differenti che utilizzano modelli differenti: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate, ad esempio, attraverso un modello relazionale o ad oggetti.

L'obiettivo dell'integratore, che è fornire un accesso integrato ad un insieme di sorgenti, si traduce allora nel difficile compito di identificare i concetti comuni all'interno delle sorgenti e risolvere le differenze semantiche che ci possono essere. Possiamo classificare queste incoerenze semantiche in tre gruppi principali:

1. *eterogeneità tra le classi di oggetti*: benchè due classi in due differenti sorgenti rappresentino lo stesso concetto nello stesso contesto, possono usare nomi diversi per gli stessi attributi o per i metodi, oppure avere gli stessi attributi con domini di valori differenti o ancora avere regole diverse sui valori;

2. *eterogeneità tra le strutture delle classi*: comprendono le differenze nei criteri di specializzazione, nelle strutture per realizzare una aggregazione, ed anche le discrepanze schematiche, quando cioè valori di attributi sono invece parte dei metadati in un altro schema (ad esempio, l'attributo SESSO presente in uno schema può essere assente in un altro schema che rappresenta le persone attraverso le classi MASCHI e FEMMINE);

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

Occorre sottolineare la possibilità di sfruttare adeguatamente queste differenze semantiche per arricchire il nostro sistema: analizzando a fondo queste differenze e le loro motivazioni si può arrivare al cosiddetto *arricchimento semantico*, ovvero l'aggiunta esplicita ai dati di tutte quelle informazioni che erano originariamente presenti negli schemi, in un formato non interrogabile, sottoforma di metadati.

1.3 Il sistema MOMIS

Tenendo presente tutte le problematiche relative al mediatore esposte nella sezione 1.1.3, e un insieme di progetti pre-esistenti quali TSIMMIS [8, 9, 10, 11], GARLIC [12, 13] e SIMS[14, 15], si è giunti alla progettazione di un sistema intelligente di integrazione d'informazioni: MOMIS, acronimo di **M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources. Il sistema si basa sulla architettura I^3 esposta nella sezione 1.1.2 ed è stato progettato per integrare sorgenti di dati testuali, sia strutturate che semistrutturate. MOMIS nasce all'interno del progetto MURST ex-40% INTERDATA, come collaborazione fra l'Università di Modena e Reggio Emilia e l'Università di Milano.

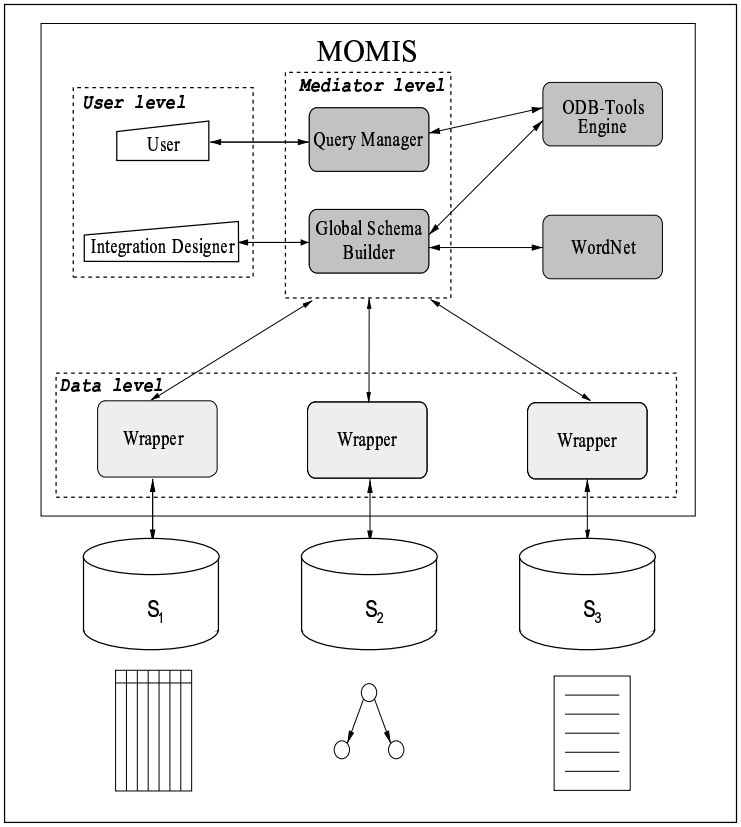


Figura 1.3: Architettura generale del sistema MOMIS

1.3.1 L'architettura di MOMIS

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

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

- **Livello Dati.** Qui si trovano i **Wrapper**. Posti al di sopra di ciascuna sorgente, sono i moduli che fungono da interfaccia tra il mediatore vero e proprio e le sorgenti locali di dati. La loro funzione è duplice:
  - in fase di integrazione, forniscono una descrizione delle informazioni contenute nelle sorgenti, utilizzando il linguaggio ODL_{I^3} (vedi appendice B);
  - in fase di query processing, traducono la query ricevuta dal mediatore (espressa nel linguaggio comune di interrogazione OQL_{I^3}, definito in analogia al linguaggio OQL) in una interrogazione comprensibile ed eseguibile 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;
- **Livello Mediatore.** Il *mediatore* è il cuore del sistema ed è composto da due moduli distinti:
  - *Global Schema Builder* (GSB), il modulo che integra gli schemi locali: partendo dalle descrizioni delle sorgenti espresse attraverso il linguaggio ODL_{I^3} ed interagendo con il progettista, genera un unico schema globale sul quale l'utente formulerà le query;
  - *Query Manager* (QM), il modulo di gestione delle interrogazioni. Servendosi delle tecniche di Logica Descrittiva, il Query Manager genera automaticamente la traduzione della query sottomessa nelle corrispondenti sub-query da sottoporre ai wrapper: query e sotto-query sono espresse in linguaggio OQL_{I^3};
- **Livello Utente.** Il progettista interagisce col Global Schema Builder e crea la vista integrata delle sorgenti: l'utente formula le interrogazioni sullo schema globale passandole come input al Query Manager , che interrogherà le sorgenti e fornirà all'utente la risposta cercata.

1.3.2 Il processo di Integrazione

Molti sistemi integratori si basano su una *vista globale* costruita sulla base degli schemi delle sorgenti che contengono le informazioni da integrare: data una query, il sistema individua ed interroga le sorgenti che contengono le informazioni cercate (eseguendo successivamente l'integrazione delle risposte ottenute) in base alla vista globale costruita inizialmente.
La novità introdotta con MOMIS risiede nella fase preliminare di costruzione della vista globale: mentre negli altri sistemi tale compito deve essere eseguito manualmente, nel sistema MOMIS viene eseguito in modo semi-automatico attraverso la costruzione di uno *schema globale* che descrive la vista.

In MOMIS, l'integrazione di sorgenti informative strutturate e semistrutturate avviene utilizzando le descrizioni degli schemi locali espresse in linguaggio ODL_{I^3} e combinando le tecniche di Description Logics e di clustering. Come mostrato in figura 1.4, le attività compiute sono le seguenti:

1. *Generazione del Thesaurus Comune*, con il supporto di ODB-Tools e di WordNet. In questa fase viene costruito un Thesaurus Comune di *relazioni terminologiche*. Tali relazioni esprimono la conoscenza intra-schema e inter-schema su sorgenti diverse e corrispondono alle asserzioni intensionali utilizzate in [16]. Le relazioni terminologiche sono derivate in modo semi-automatico a partire dalle descrizioni degli schemi in ODL_{I^3}, attraverso l'analisi strutturale (utilizzando ODB-Tools e le tecniche di Description Logics) e di contesto (attraverso l'uso di WordNet) delle classi coinvolte. Questa fase sarà discussa nella sezione 2.1.

2. *Calcolo delle affinità e generazione dei cluster di classi ODL_{I^3}*. Le relazioni contenute nel Thesaurus Comune vengono utilizzate per valutare il livello di affinità tra le classi ODL_{I^3} in modo da identificare le informazioni che devono essere integrate a livello globale. A tal fine, si utilizza il modulo ARTEMIS di SI-Designer che calcola dei coefficienti indicanti il livello di affinità delle classi ODL_{I^3}: il calcolo viene eseguito basandosi sia sui nomi delle classi che sui nomi e domini degli attributi. Le classi ODL_{I^3} con maggiore affinità sono raggruppate utilizzando le tecniche di clustering [17]. Questa fase sarà discussa nelle sezioni 2.1.6 e 2.1.7.

3. *Costruzione dello schema globale*, i cluster di classi ODL_{I^3} affini sono analizzati per costruire lo schema globale del Mediatore. Per ciascun cluster viene definita una classe globale ODL_{I^3} che rappresenta tutte le classi locali riferite al cluster ed è caratterizzata dall'unione *ragionata* dei loro attributi e da una *mapping-table* (vedere capitolo 3). L'insieme delle classi globali

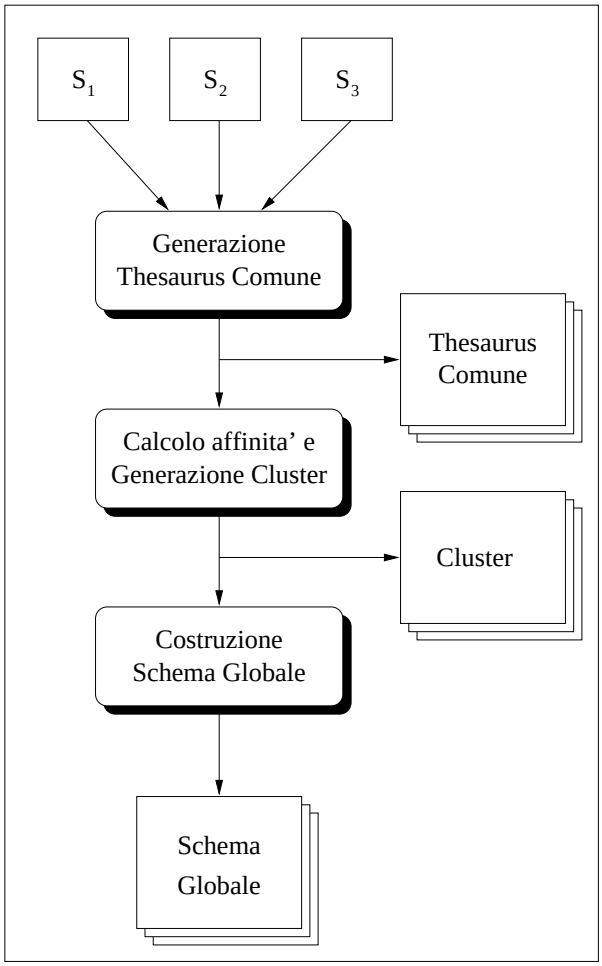


Figura 1.4: Le fasi dell'integrazione

definite costituisce lo schema globale del Mediatore che deve essere usato per porre le query alle sorgenti locali integrate.

1.3.3 Query processing e ottimizzazione

Quando l'utente pone una query sullo schema globale, MOMIS la analizza e produce un insieme di sub-query che saranno inviate a ciascuna sorgente informativa coinvolta. In accordo con altri approcci proposti in quest'ambito [14, 15], il processo consiste di due attività principali:

1. *Ottimizzazione Semantica.* L'ottimizzazione è basata sull'inferenza logica a partire dalla conoscenza contenuta nei vincoli di integrità dello schema

globale. La stessa procedura di ottimizzazione semantica si realizza in termini locali su ogni sub-query tradotta dal Mediatore nella formulazione del piano d'accesso: in questo caso ci si basa sui vincoli di integrità presenti sui singoli schemi locali.

2. *Formulazione del piano di accesso.* Il Mediatore utilizza una "mappa" (*mapping table*, generata nella costruzione dello schema globale) che definisce l'associazione tra classi globali e classi locali. La query globale viene espressa in termini degli schemi locali, considerando anche l'eventuale conoscenza costituita da regole inter-schema definite sulle estensioni delle classi locali.

Il Mediatore agisce sulla query sfruttando la tecnica di ottimizzazione semantica supportata da ODB-Tools , in modo da ridurre il costo del piano d'accesso, e, dopo aver ottenuto la query ottimizzata, genera l'insieme di sub-query relative alle sorgenti coinvolte.

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

Il linguaggio ODL (**O**bject **D**efinition **L**anguage) per la specifica di schemi ad oggetti, proposto dal gruppo di standardizzazione ODMG-93 [18, 19], è universalmente riconosciuto come standard. Le sue caratteristiche peculiari, al pari di altri linguaggi basati sul paradigma ad oggetti, possono essere così riassunte:

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

Tuttavia, nonostante l'ODL sia ben progettato per rappresentare la conoscenza relativa ad un singolo schema ad oggetti, è certamente incompleto se utilizzato in un contesto di integrazione di basi di dati eterogenee quale è quello descritto in questa tesi. Pertanto, si è reso necessario definire un'estensione di tale linguaggio, denominata ODL$_{I^3}$, in accordo con le raccomandazioni della proposta di standardizzazione per i linguaggi di mediazione, risultato del lavoro di workshop I^3 svoltosi nel 1995. Partendo dal presupposto che un sistema di mediazione

dovrebbe poter supportare sorgenti con modelli complessi (come quelli ad oggetti) e modelli più semplici (come file di strutture), si è comunque cercato di discostarsi il meno possibile dal linguaggio ODL.

Con l'estensione di ODL al linguaggio ODL_{I^3} sono stati raggiunti i seguenti obiettivi:

- per ogni classe, il wrapper può indicare nome e tipo del sorgente di appartenenza;
- per le classi appartenenti ai sorgenti relazionali è possibile definire le chiavi candidate ed eventuali foreign key;
- attraverso l'uso del costrutto *union* ogni classe può avere più strutture dati alternative, mentre il costrutto *optional* consente di indicare la natura opzionale di un attributo. Queste caratteristiche sono in accordo con la strategia utilizzata per la descrizione di dati semistrutturati;
- il linguaggio supporta la definizione di grandezze locali e di grandezze globali;
- il linguaggio supporta la dichiarazione di regole di mapping (o *mapping-rule*) fra grandezze globali e grandezze locali;
- è data la possibilità di definire regole di integrità (o *if then rule*), sia sugli schemi locali, sia sullo schema globale;
- il linguaggio supporta la definizione di relazioni terminologiche di sinonimia (SYN), ipernimia (BT), iponimia (NT) e associazione (RT);
- il linguaggio può essere automaticamente tradotto nella logica descrittiva OLCD usata da ODB-Tools, e quindi utilizzarne le capacità nei controlli di consistenza e nell'ottimizzazione semantica delle interrogazioni.

In appendice B sono presentati gli elementi sintattici di ODL_{I^3}, espressi in notazione BNF, che differiscono da ODL.

1.3.5 Gli strumenti utilizzati

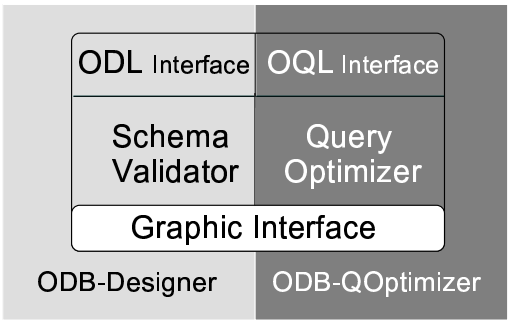
MOMIS utilizza alcuni tool esterni per svolgere il suo compito. Uno è **ODB-Tools**, che è utilizzato sia durante la fase d'integrazione degli schemi delle sorgenti per costruire il Thesaurus Comune, sia durante la fase di query processing per l'ottimizzazione di query. L'altro è **WordNet** e viene impiegato nella costruzione del Thesaurus Comune.

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 su basi di dati orientate agli oggetti (OODB), ed è stato sviluppato presso il Dipartimento di Scienze dell'Ingegneria dell'Università di Modena [20, 21]. ODB-Tools basa il suo ambiente teorico su due elementi fondamentali:

1. **OLCD** (**O**bject **L**anguage with **C**omplements allowing **D**escriptive cycles): linguaggio derivante dalla famiglia KL-ONE [22], proposto come formalismo comune per esprimere descrizioni di classi, vincoli di integrità ed interrogazioni e dotato di tecniche di inferenza basate sul calcolo della sussunzione, introdotte per le *Logiche Descrittive* nell'ambito dell'Intelligenza Artificiale;

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



Il formalismo **OLCD** deriva dal precedente **ODL**, proposto in [18, 19], che già estendeva l'espressività di linguaggi di logica descrittiva al fine di rappresentare la semantica dei modelli di dati ad oggetti complessi (*CODMs*), recentemente proposti in ambito di basi di dati deduttive e basi di dati orientate agli oggetti. In [23], ODL è stato esteso per permettere la formulazione dichiarativa di un insieme rilevante di vincoli di integrità definiti sulla base di dati. L'estensione di ODL con vincoli è stata denominata **OLCD** (**O**bject **L**anguage with **C**omplements allowing **D**escriptive cycles). Attraverso questa logica descrittiva, è possibile descrivere, oltre alle classi, anche le *regole di integrità*, permettendo 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 [24] è stato presentato il sistema OCDL-Designer, per l'acquisizione e la validazione di schemi OODB descritti attraverso **OLCD** , 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*, cioè tipi necessariamente vuoti.
In [23] l'ambiente teorico sviluppato in [24] è stato esteso per effettuare l'ottimizzazione 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 [18, 19]) in **OLCD** .

La nozione di ottimizzazione semantica di una query è stata introdotta, per le basi di dati relazionali, da King [25, 26] e da Hammer e Zdonik [27]. 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 [28].
Il calcolo dell'espansione semantica di una classe permette di rilevare nuove relazioni *isa*, cioè 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 (la query 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 a quella iniziale. In questo modo la query 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.

WordNet

WordNet [29] è un database lessicale elettronico ed è considerato la più importante risorsa disponibile per i ricercatori nei campi della linguistica computazionale, dell'analisi testuale, e in altre aree associate. É stato sviluppato dal Cognitive science Laboratory alla Princeton University, sotto la direzione del Professor George A. Miller ed è disponibile gratuitamente.

La sua architettura si ispira alle teorie psicolinguistiche attuali che sono legate alla memoria lessicale umana. Sostantivi, verbi, aggettivi e avverbi della lingua inglese vengono organizzati in insiemi di sinonimi, chiamati *synset*, ognuno dei quali rappresenta un determinato concetto lessicale. Nel database lessicale sono memorizzate delle relazioni che collegano fra loro i synset.

Il principio della semantica lessicale è il riconoscimento che esiste una associazione convenzionale tra la *forma* delle parole (il modo in cui sono pronunciate e scritte) e i *concetti* che esse esprimono. La corrispondenza tra la forma delle parole e il loro significato può essere sintetizzata in una tabella, la **Matrice lessicale** (Figura 1.5): l'elemento M_i, j che assume il valore E_i, j indica che la parola F_j può assumere il significato M_i, mentre se M_i, j è vuoto significa che la parola F_j non assume mai il significato M_i.

Dalla figura si nota il tipo di corrispondenza molti-a-molti, da cui emergono due proprietà:

- *Polisemia*: una stessa parola può avere due o più significati. Nella matrice compaiono due o più elementi in colonna, come, ad esempio, accade per la forma F_2;

- *Sinonimia*: significato che può avere due o più parole in grado di esprimerlo. Nella matrice compaiono due o più elementi in riga, ad esempio in corrispondenza di M_1.

La *Polisemia* e la *Sinonimia* costituiscono problemi lessicografici tra loro complementari. Mentre è in atto una comunicazione in forma scritta/orale, da un lato il lettore/ascoltatore riceve la parola, cercando di capirne il significato tra tutti quelli che la stessa può esprimere, dall'altro lo scrittore/oratore conosce il concetto che

Word Meanings	Word Forms				
	F_1	F_2	F_3	. . . . .	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
. . .				. . . . .	
M_m					$E_{m,n}$

Figura 1.5: La Matrice Lessicale

vuole esprimere e si trova nella situazione di dover scegliere una fra le varie forme che possono esprimerlo.

WordNet rappresenta i concetti seguendo la teoria cosiddetta *differenziale*, secondo cui i diversi significati non sono necessariamente denotati da definizioni scritte, ma vengono rappresentati e distinti tra loro attraverso l'uso di differenti simboli conosciuti dall'utente, ai quali l'utente stesso associa il concetto. In altre parole, questa scelta parte dal presupposto che l'utente conosca già sufficientemente bene la lingua (nella fattispecie l'inglese) e sia pertanto in grado di riconoscere il significato di una determinata parola in base ai vari sinonimi che la affiancano all'interno di un *synset*, senza che ne venga data esplicitamente una definizione.

Tipi di relazioni La distinzione fondamentale operata da WordNet sui termini consiste nel suddividerli in sostantivi, verbi, aggettivi e avverbi. Non si vedranno pertanto mai accostate nel medesimo *synset*, parole appartenenti a categorie differenti.

Il database lessicale collega i termini in base a relazioni semantiche. Poiché una relazione semantica è una relazione fra significati, e considerato che i significati, a causa della sinonimia, sono associati a set di termini sinonimi, è naturale pensare alle relazioni semantiche come a relazioni tra insiemi di sinonimi. Da ciò emerge una distinzione tra la relazione di sinonimia e le altre relazioni semantiche, denotata anche dal fatto che ogni insieme di termini sinonimi è racchiuso fra parentesi graffe {}, mentre gli insiemi prodotti da tutte le altre relazioni sono racchiusi fra parentesi quadre [ ].

Le relazioni semantiche godono di due proprietà:

- Se esiste una relazione R fra gli insiemi $\{x,x', ...\}$ e $\{y,y', ...\}$, allora deve esistere una relazione inversa R' fra $\{y,y', ...\}$ e $\{x,x', ...\}$. R e R' possono, non necessariamente, coincidere.

- Se esiste una relazione R fra gli insiemi $\{x,x', ...\}$ e $\{y,y', ...\}$, allora vale $[x\ R\ y]$, $[x\ R\ y']$, ... , $[x'\ R\ y]$, etc...

Vediamo i principali tipi di relazioni che sono codificate in WordNet:

- **Sinonimia.** *Due termini si definiscono sinonimi se possono essere indifferentemente scambiati senza che, nel contesto in cui si trovano, il significato cambi.*
  É opportuno osservare che nella realtà sono pochi i sinonimi in senso stretto: più comunemente si parla di sinonimi riferiti ad un particolare contesto. Ad esempio, 'versare' può assumere il significato di 'travasare' un liquido ovvero di 'depositare' soldi ed è sinonimo di entrambi i termini, ma se viene utilizzato col primo significato certamente non potrà essere sostituito con 'depositare'. La sinonimia tra termini è la relazione più importante, perché ogni insieme *synset* che ne scaturisce rappresenta la semantica di un concetto.

- **Antonimia.** *Due termini sono in relazione di antonimia se uno è il contrario dell'altro.*
  Da osservare che l'antonimo del termine x non sempre coincide con '$non\ x$': ad esempio 'fradicio' e 'asciutto' sono in relazione di antonimia, ma 'non bagnato' non coincide con 'asciutto', dal momento che un oggetto può essere solamente 'umido', vale a dire contemporaneamente 'non fradicio' e 'non asciutto'. Tra le relazioni citate, l'antonimia è l'unico tipo di relazione lessicale che si applica fra singoli termini e non fra concetti da *synset*.
  Ad esempio non si può affermare che {rise,ascend} e {fall,discend} siano antonimi, pur essendolo singolarmente [rise/fall] (e anche [ascend/discend]).

- **Iponimia.** *Un concetto è iponimo di un altro quando lo specializza: tra i due esiste un rapporto di tipo ISA.*
  Ad esempio 'acero' è iponimo di 'albero' e 'albero' è a sua volta iponimo di 'pianta'. L'iponimia gode della proprietà transitiva: nell'esempio si deduce che 'acero' è iponimo di 'pianta'. Questa proprietà consente la costruzione si sistemi ereditari, gerarchie nelle quali ogni concetto iponimo eredita tutte le caratteristiche del suo superconcetto e ne aggiunge almeno una che lo distingue dallo stesso superconcetto e da qualsiasi suo altro iponimo. Inoltre, l'iponimia è una relazione asimmetrica: la sua relazione duale è **Ipernimia**.

- **Olonimia.** *La Olonimia è una relazione semantica che si esprime fra due concetti x e y ("x olonimo di y") quando y* **is a part of** x.
  Per esempio, 'riempire' è olonimo di 'versare' perché certamente il concetto di "riempire qualcosa" implica il concetto di "versare qualcosa", ma "versare qualcosa" da solo non basta a rappresentare il concetto di riempimento (per 'riempire' qualcosa bisogna 'versare' qualcos'altro in un recipiente), cioè il concetto di riempimento è un concetto composto e 'versare' è solo una parte (*is a part of*) di tale concetto.
  Come la relazione precedente, anche la Olonimia gode della proprietà transitiva, ed è asimmetrica: la relazione duale è **Meronimia**.
  Come nel caso precedente si possono realizzare gerarchie di concetti olonimi/meronimi. In questo caso però uno stesso meronimo può avere più olonimi: uno stesso componente può contemporaneamente far parte di differenti concetti composti.

- **Correlazione.** *Due termini sono correlati quando condividono uno stesso ipernimo.* Questa relazione dunque è indiretta poiché è derivata da altre relazioni.

Nel database implementato da WordNet, l'insieme di tutte le relazioni tra le parole, dei diversi tipi appena descritti, formano una rete complessa. In questo modo, secondo la teoria *differenziale* adottata, il significato di una parola data può essere determinato in base alla collocazione che la stessa ha all'interno della rete.

Morfologia delle parole Un aspetto che finora non è stato considerato, riguarda la morfologia delle parole che, nella lingua inglese, si manifesta come segue:

- declinazioni per i sostantivi, ad esempio distinzione fra singolare e plurale;
- coniugazioni per i verbi.

Le prime versioni di WordNet non tenevano conto di questo aspetto, con ovvie limitazioni sulla funzionalità del sistema: basti pensare all'inserimento di un termine plurale che non viene riconosciuto. La versione attuale di WordNet rivolve il problema attraverso un componente software che, interfacciandosi fra l'utente e il database, converte ogni termine in una forma canonica, che è quella memorizzata nel database. Ad esempio, il termine 'trees' verrà convertito in 'tree'.

Come si vedrà nel capitolo 2, il sistema Momis sfrutta le relazioni terminologiche di Sinonimia (SYN), Iponimia (NT), Ipernimia (BT) e Olonimia (considerata come relazione associativa RT). Attualmente la relazione di Antonimia non viene considerata: si potrebbe tuttavia pensare di sfruttarla in futuro come

ulteriore conoscenza, interpretando l’antonimia tra due classi come relazione di disgiunzione estensionale.

Capitolo 2

Il processo di integrazione di sorgenti in MOMIS

2.1 Generazione del Thesaurus Comune

Nell'ambito di un approccio semantico alla integrazione di dati (quale è quello adottato nel progetto **MOMIS**) è di fondamentale importanza la conoscenza delle informazioni semantiche relative al contesto e alla struttura dei vari schemi sorgenti. Tale conoscenza è contenuta nel cosiddetto *Thesaurus Comune*, un dizionario all'interno del quale sono presenti un insieme di relazioni terminologiche che legano tra loro classi ed attributi. In figura 2.1 sono evidenziate le fasi del processo che porta alla generazione del Thesaurus Comune.

Lo sforzo compiuto a questo livello è quello di rendere il processo il più possibile automatico, minimizzando l'intervento del progettista, la cui partecipazione si rende comunque indispensabile.

2.1.1 Estrazione delle relazioni intra-schema

La struttura di ogni schema locale contiene delle informazioni semantiche ricavabili dalle gerarchie di ereditarietà e aggregazione. Terminata la fase di acquisizione degli schemi locali, il sistema è in grado di estrarre in modo automatico alcune relazioni.
Da un'analisi delle sorgenti relazionali sono estratte le relazioni terminologiche definite dalle foreign key:

- se la foreign key è anche chiave primaria nella relazione di partenza e in quella referenziata allora la relazione terminologica è di tipo BT/NT;
- altrimenti è semplicemente di tipo RT.

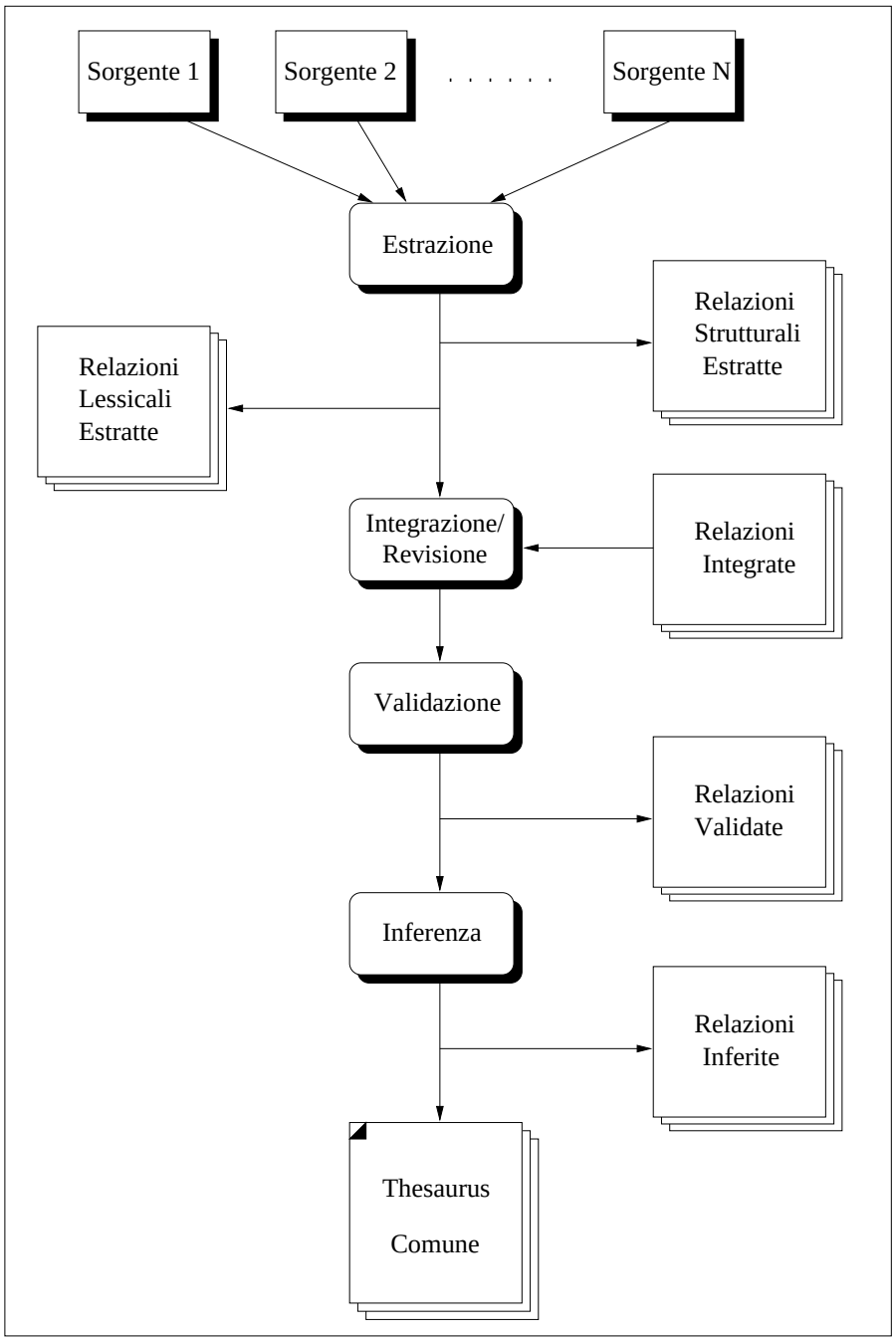


Figura 2.1: Il processo di generazione del Thesaurus Comune

L'analisi delle sorgenti ad oggetti invece produce:

- relazioni BT/NT ogni volta che c'è ereditarietà tra classi,
- relazioni RT quando si è in presenza di gerarchie di aggregazione.

Esempio 1 Dall'esempio di riferimento, si ricavano le seguenti relazioni terminologiche intra-schema:

⟨`CS.Professor` NT `CS.CS_Person`⟩
⟨`CS.Student` NT `CS.CS_Person`⟩
⟨`CS.Professor` RT `CS.Division`⟩
⟨`CS.Student` RT `Cs.Course`⟩
⟨`CS.Division` RT `CS.Location`⟩
⟨`CS.Course` RT `CS.Professor`⟩
⟨`UNI.Research_Staff` RT `UNI.Department`⟩
⟨`UNI.Research_Staff` RT `UNI.Section`⟩
⟨`UNI.Section` RT `UNI.Room`⟩

Se nella relazione `UNI.Research_Staff` avessimo avuto anche la foreign key:

```
foreign_key (first_name, last_name) references School_Member
```

allora avremmo avuto anche la relazione ⟨`UNI.Research_Staff` BT `UNI.School_Member`⟩.

2.1.2 Estrazione delle relazioni inter-schema

Le relazioni inter-schema, terminologiche ed estensionali, sono estratte analizzando gli schemi ODL_{I^3} insieme. L'estrazione di queste relazioni è basata sulle relazioni lessicali che sussistono tra i nomi di classi ed attributi, derivanti dai significati delle parole. Questo tipo di conoscenza non è basato sulle regole di un linguaggio di definizione di dati ma deriva dal nome assegnato dal progettista: è compito del progettista assegnare nomi esplicativi e significativi. L'incertezza nell'interpretazione è comunque dovuta all'ambiguità del linguaggio: Bates [30] scrive "*la probabilità che due persone usino lo stesso termine nel descrivere la stessa cosa è inferiore al 20%*".

Qualunque conoscenza associata ai nomi dello schema è un'opportunità che deve essere presa in considerazione. Essendo impossibile compiere questo lavoro manualmente, diventa indispensabile l'uso di un database lessicale, quale WordNet [29], per estrarre e proporre al progettista relazioni inter-schema intensionali.

Le tipologie di relazione implementate da WordNet vengono tradotte in relazioni del Thesaurus in questo modo:

- **Sinonimia**: corrisponde ad una relazione SYN;
- **Ipernimia**: corrisponde ad una relazione BT;
- **Olonimia**: corrisponde ad una relazione RT;
- **Correlazione**: corrisponde ad una relazione RT;

Non tutte le relazioni estratte da WordNet rappresentano un ulteriore arricchimento del Thesaurus: questo accade spesso quando negli schemi ci sono dei nomi composti. In questi casi, è il progettista che deve intervenire ed eliminare le relazioni errate altrimenti si corre il rischio di creare uno schema globale non corretto.

Esempio 2 Applicando l'uso di WordNet all'esempio di riferimento, le relazioni ottenute, dopo essere state filtrate dal progettista, sono le seguenti:

⟨CS.CS_Person.name SYN TP.Un_Student.name⟩
⟨CS.CS_Person.name BT UNI.Research_Staff.first_name⟩
⟨CS.CS_Person.name BT UNI.School_Member.first_name⟩
⟨CS.CS_Person.name BT UNI.Research_Staff.last_name⟩
⟨CS.CS_Person.name BT UNI.School_Member.last_name⟩
⟨TP.Un_Student.name BT UNI.Research_Staff.first_name⟩
⟨TP.Un_Student.name BT UNI.School_Member.first_name⟩
⟨TP.Un_Student.name BT UNI.Research_Staff.last_name⟩
⟨TP.Un_Student.name BT UNI.School_Member.last_name⟩
⟨UNI.Research_Staff.last_name SYN UNI.School_Member.last_name⟩
⟨UNI.Research_Staff.first_name SYN UNI.School_Member.first_name⟩
⟨CS.Professor RT TP.Un_Student.faculty_name⟩
⟨CS.Professor RT UNI.School_Member.faculty⟩
⟨CS.Professor.rank SYN CS.Student.rank⟩
⟨CS.Student.year SYN UNI.School_Member.year⟩
⟨UNI.School_Member.faculty SYN TP.Un_Student.faculty_name⟩
⟨UNI.Section.room_code SYN University.Room⟩

2.1.3 Integrazione dell'insieme di relazioni

Nuove relazioni possono essere fornite direttamente dal progettista per aggiungere al Thesaurus una conoscenza specifica non ricavata automaticamente nelle fasi precedenti. Questa è una operazione cruciale, perché le nuove relazioni sono forzate ad entrare nel Thesaurus Comune, contribuendo così alla creazione dello schema globale. Ciò significa che se viene inserita una relazione sbagliata o senza senso, si può giungere ad uno schema globale errato. Nella fase di Validazione il progettista viene aiutato per evitare di incombere in una situazione di questo tipo.

Esempio 3 Nell'esempio di riferimento il progettista aggiunge le seguenti relazioni:

⟨`UNI.Research_Staff` BT `CS.Professor`⟩
⟨`UNI.School_Member` BT `CS.Student`⟩
⟨`TP.Un_Student` BT `CS.Student`⟩
⟨`UNI.Department` BT `CS.Division`⟩
⟨`UNI.Section` SYN `CS.Course`⟩
⟨`UNI.Research_Staff.dept_code` BT `CS.Professor.belongs_to`⟩
⟨`UNI.Department.dept_name` SYN `CS.Division.description`⟩
⟨`UNI.Section.section_name` SYN `CS.Course.course_name`⟩
⟨`UNI.School_Memeber.faculty` SYN `TP.Un_Student.faculty_name`⟩
⟨`CS.Division.fund` SYN `UNI.Department.budget`⟩
⟨`UNI.Department.dept_area` SYN `CS.Division.sector`⟩

2.1.4 Validazione del Thesaurus Comune

La relazioni intensionali tra attributi e quelle estensionali tra classi sono validate utilizzando ODB-Tools . La validazione di relazioni intensionali tra nomi di attributi è basata sulla compatibilità dei domini associati agli attributi stessi: in questo modo, le relazioni intensionali sono classificate in *valide* e *non valide*.
Siano $a_t = \langle n_t, d_t \rangle$ e $a_q = \langle n_q, d_q \rangle$ una coppia di attributi posti in relazione tra loro. A seconda del tipo di relazione, per la validazione sono adottati i seguenti criteri:

- $\langle n_t$ SYN $n_q \rangle$: la relazione è *valida* se i domini d_t e d_q sono equivalenti o se uno dei due è più specializzato dell'altro;
- $\langle n_t$ BT $n_q \rangle$: la relazione è *valida* se d_t contiene o è equivalente a d_q;
- $\langle n_t$ BT $n_q \rangle$: analogamente al caso precedente la relazione è *valida* se d_t è un sottoinsieme non proprio di d_q;

Quando il dominio di un attributo d_t (d_q) è definito utilizzando il costruttore `union`, la relazione che coinvolge quell'attributo è *valida* se almeno uno dei suoi domini rispetta i criteri sopra elencati.

Esempio 4 Il risultato della validazione delle relazioni aggiunte direttamente dal progettista è il seguente:

⟨CS.CS_Person.name SYN TP.Un_Student.name⟩ [1]
⟨CS.CS_Person.name BT UNI.Research_Staff.first_name⟩ [1]
⟨CS.CS_Person.name BT UNI.School_Member.first_name⟩ [1]
⟨CS.CS_Person.name BT UNI.Research_Staff.last_name⟩ [1]
⟨CS.CS_Person.name BT UNI.School_Member.last_name⟩ [1]
⟨TP.Un_Student.name BT UNI.Research_Staff.first_name⟩ [1]
⟨TP.Un_Student.name BT UNI.School_Member.first_name⟩ [1]
⟨TP.Un_Student.name BT UNI.Research_Staff.last_name⟩ [1]
⟨UNI.Research_Staff.last_name SYN UNI.School_Member.last_name⟩ [1]
⟨UNI.Research_Staff.first_name SYN UNI.School_Member.first_name⟩ [1]
⟨CS.Professor.rank SYN CS.Student.rank⟩ [1]
⟨CS.Student.year SYN UNI.School_Member.year⟩ [1]
⟨UNI.School_Member.faculty SYN TP.Un_Student.faculty_name⟩ [1]
⟨UNI.Research_Staff.dept_code BT CS.Professor.belongs_to⟩ [0]
⟨UNI.Department.dept_name SYN CS.Division.description⟩ [1]
⟨UNI.Section.section_name SYN CS.Course.course_name⟩ [1]
⟨UNI.School_Memeber.faculty SYN TP.Un_Student.faculty_name⟩ [1]
⟨CS.Division.fund SYN UNI.Department.budget⟩ [1]
⟨UNI.Department.dept_area SYN CS.Division.sector⟩ [1]

dove ”[1]” indica che la relazione è valida mentre ”[0]” indica il contrario.
L’unica relazione non validata è ⟨UNI.Research_Staff.dept_code BT CS.Professor.belongs_to⟩: infatti il dominio di dept_code è un integer mentre quello di belongs_to è una oggetto UNI.Division.

2.1.5 Inferenza di nuove relazioni intensionali

Sulla base delle relazioni fin qui ottenute, attraverso un procedimento automatico che fa uso di tecniche di inferenza, viene generato un nuovo insieme di relazioni tra classi, che contribuiscono ad arricchire ulteriormente (e completare) il Thesaurus.
Per poter agire in questo senso occorre effettuare alcune modifiche agli schemi locali: è importante precisare che tali modifiche hanno carattere provvisorio e servono solamente per il calcolo di deduzione di nuove relazioni inferite.

Le **classi** sono inserite nello schema e collegate tra loro in modo da conformarsi alle relazioni esistenti:

- ogni relazione BT e NT dà luogo a una gerarchia di ereditarietà;
- la relazione SYN produce una doppia ereditarietà, da cui emerge come le due classi siano vicendevolmente in rapporto *is_a*;
- ogni relazione RT produce un’aggregazione.

Notare come, oltre a ricomporsi i singoli schemi locali, le relazioni lessicali aggiungano nuovi collegamenti inter-schema, in modo tale che alla fine risulti uno schema unico.
Gli attributi coinvolti nelle relazioni validate vengono organizzati in gerarchie nelle quali la sinonimia pone due attributi allo stesso livello, mentre le relazioni BT e NT pongono a livello superiore il termine più generale. Alla fine risultano diversi alberi gerarchici isolati e ogni attributo viene rinominato, assumendo il nome del termine di più alto livello dell'albero a cui appartiene.
Quest'ultima operazione ha lo scopo di rendere il più possibile confrontabili le intensioni dello schema prodotto dall'analisi delle relazioni tra le classi. In questo modo l'intervento di ODB-Tools, attraverso tecniche di intelligenza artificiale basate sul calcolo di *sussunzione*, è in grado di inferire tra le classi locali nuove gerarchie di aggregazione ed ereditarietà, che si traducono facilmente in nuove relazioni per il Thesaurus.

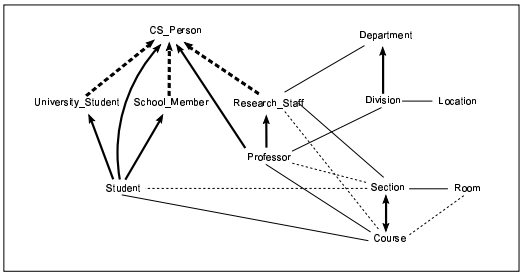


Figura 2.2: Thesaurus Comune per le sorgenti dell'esempio

Esempio 5 Le relazioni inferite sono le seguenti:

⟨`CS.CS_Person` BT `UNI.Research_Staff`⟩
⟨`CS.CS_Person` BT `UNI.School_Member`⟩
⟨`UNI.Section` RT `CS.Professor`⟩
⟨`UNI.Research_Staff` RT `CS.Course`⟩
⟨`CS.Professor` RT `UNI.Department`⟩
⟨`CS.Professor` RT `UNI.Section`⟩
⟨`CS.Course` RT `UNI.Room`⟩
⟨`CS.Student` RT `UNI.Section`⟩
⟨`CS.CS_Person` BT `TP.Un_Student`⟩

In Figura 2.2 sono riportate tutte le classi di tutte le sorgenti, così come risultano riorganizzate alla fine della fase di generazione del Thesaurus comune, facendo quindi riferimento alle classi modificate. È quindi una rappresentazione grafica delle relazioni che sussistono tra queste: le linee tratteggiate mettono in evidenza le relazioni inferite, le

linee unite rappresentano invece quelle esplicite; le linee dotate di frecce rappresentano inoltre relazioni di generalizzazione, mentre quelle in cui le frecce sono assenti denotano le relazioni di aggregazione.

2.1.6 Calcolo delle affinità

La conoscenza semantica contenuta nel Thesaurus comune appena generato rappresenta il punto di partenza per il successivo processo che porta alla definizione dello schema globale integrato. Scopo di questa fase è quantificare il grado di *affinità* mutuo fra le classi locali, al fine di raggrupparle in cluster.

Il livello di similarità tra le classi viene calcolato su due dimensioni:

- da un punto di vista strutturale, attraverso lo *Structural Affinity Coefficient*, calcolato tra due classi in base alle relazioni tra i loro attributi;
- sul piano dei nomi, attraverso il *Name Affinity Coefficient*, calcolato a partire dalle relazioni che legano coppie di classi.

La combinazione di questi due coefficienti produce il *Global Affinity Coefficient*, che rappresenta il vero indicatore di similarità fra due classi.

Per effettuare il calcolo di affinità, il Thesaurus viene organizzato in una struttura simile alle Associative Networks [31], dove i nodi (ciascuno dei quali rappresenta genericamente un termine, sia esso il nome di una classe o il nome di un attributo) sono uniti attraverso relazioni terminologiche. A loro volta, tutte le relazioni presenti in questa rete sono percorribili in entrambi i sensi (dunque anche le BT e NT): due termini sono quindi affini se esiste un percorso che li unisce, formato da qualsivoglia relazioni. Per dare una valutazione numerica dell'affinità tra due termini, a ogni tipo di relazione viene associato un peso $\sigma_{\Re}$ che sarà tanto maggiore quanto più questo tipo di relazione contribuisce a legare due termini: sarà quindi $\sigma_{syn} \geq \sigma_{bt/nt} \geq \sigma_{rt}$).

In questa sezione si userà $\sigma_{ij_{\Re}}$ per denotare il peso della relazione terminologica $\Re$ definita tra i termini t_i e t_j. Nel nostro esempio, e nelle sperimentazioni precedentemente realizzate presso l'Università di Milano, si è adottato:

$\sigma_{syn} = 1$;
$\sigma_{bt} = \sigma_{nt} = 0.8$;
$\sigma_{rt} = 0.5$.

Definizione 1 (Funzione di Affinità) Presi due termini, t_i e t_j, possono essere presenti nel grafo zero o più cammini che li uniscono, formati da relazioni. Ad ognuno di questi cammini corrisponde un valore, dato dal prodotto dei pesi delle relazioni in esso coinvolte. La Funzione di Affinità $A(t_i, t_j)$ tra due termini, t_i e

t_j, restituisce il valore maggiore tra questi, corrispondente al cammino più *stringente*, che unisce questi termini (che non sempre coincide col cammino più breve), definito come segue:

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

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

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

Esempio 6 Si consideri il Thesaurus illustrato in Figura 2.2. Esiste un cammino tra le classi `University_Student` e `School_Member`, ovvero, `University_Student` $\rightarrow^{nt}$ `CS_Person` $\rightarrow^{bt}$ `School_Member`. Da questo si deduce che $A($`University_Student`, `School_Member`$) = 0.8 \cdot 0.8 = 0.64$.

Definizione 2 (Termini Affini) Due termini t_i, t_j si dicono *affini*, e si denotano con $t_i \sim t_j$, se la loro Funzione di Affinità restituisce un valore maggiore o uguale ad un predefinito valore di soglia $\alpha > 0$, cioè:

$$t_i \sim t_j \Longleftrightarrow A(t_i, t_j) \geq \alpha$$

Coefficienti di Affinità

In questo paragrafo, vengono date le definizioni dei parametri attraverso i quali si determina l'affinità tra due classi, cioè i coefficienti *Name Affinity Coefficient*, *Structural Affinity Coefficient* e *Global Affinity Coefficient* facendo riferimento a due classi ODL_{I^3} c e c' appartenenti rispettivamente alle sorgenti S e S'.

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

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

In sostanza il *Name Affinity Coefficient* tra due classi coincide esattamente con la Funzione di affinità solo se il suo valore supera la soglia α, altrimenti è nullo.

Esempio 7 Si considerino le classi `University_Student` e `School_Member`. Si avrà che $NA(\texttt{University\_Student}, \texttt{School\_Member}) = 0.64$

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

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

dove:

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

$$\text{C} = \{(a_t, a_q) \mid a_t \in A(c), a_q \in A(c'), \langle n_t \text{ SYN } n_q\rangle \vee \langle n_t \text{ BT } n_q\rangle \vee \langle n_t \text{ NT } n_q\rangle\}$$

Lo Structural Affinity Coefficient tra due classi è un valore compreso nell'intervallo [0,1]: viene valutato utilizzando la funzione di Dice e raffinato da un fattore di controllo F_c, che indica la percentuale di attributi aventi una affinità data da relazioni validate. Un valore pari a 0 indica l'assenza di attributi affini nelle classi considerate, mentre il valore 1 indica che tutti gli attributi defini nelle due sono affini e validi. Più grande è il numero di attributi affini nelle classi considerate, e più grande è il numero dei controlli sulla validità che hanno dato esito positivo, maggiore è il valore di SA(c,c').

Esempio 8 Date classi `University_Student` della sorgente `Tax_Position` e `School_Member` della sorgente `University`, abbiamo che $SA(\texttt{University\_Student}, \texttt{School\_Member}) = \frac{2 \cdot 3}{4+4} \cdot \frac{1}{1} = 0.75$.

In generale, date due classi, un attributo di una classe potrebbe essere affine con più di un attributo dell'altra classe. Nella valutazione del coefficiente SA(c,c'), queste affinità multiple vengono considerate come una singola corrispondenza tra un attributo e un insieme di attributi.
Per la valutazione dello SA, gli attributi opzionali rappresentano un caso a parte. In funzione degli attributi che vengono considerati, il coefficiente SA può essere calcolato in diversi modi:

- *In base a tutti gli attributi.* Gli attributi opzionali sono considerati come gli altri e vengono sempre presi in considerazione per valutare l'affinità delle classi.
- *In base ad un attributo comune.* Gli attributi opzionali non vengono presi in considerazione.
- *In base ad una soglia.* Gli attributi opzionali vengono considerati solo se sono presenti in un certo numero di oggetti.

La terza opzione è difficile da applicare dal momento che richiede un valore di soglia che può dipendere dalla singola classe oppure dalla sorgente. Si osservi inoltre come, a parità di numero di attributi affini, la seconda opzione fornisca un valore maggiore rispetto alla prima: la scelta tra la prima o la seconda opzione dipende dal tipo di sorgente che si sta analizzando.

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

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

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

Esempio 9 Partendo dai coefficienti riportati negli Esempi 7 e 8, il Global Affinity Coefficient delle classi `University_Student` e `School_Member` è calcolato nel seguente modo:

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

2.1.7 Generazione dei Cluster

Per l'identificazione degli insiemi di classi affini negli schemi considerati sono utilizzate, all'interno del mediatore, tecniche di clustering, attraverso le quali le classi sono automaticamente classificate in gruppi caratterizzati da differenti livelli di affinità, formando un albero.

La procedura di clustering utilizza una matrice M di rango r, con r uguale al numero totale di classi ODL$_{I^3}$ che devono essere analizzate. In ogni casella $M[h,k]$ della matrice è rappresentato il coefficiente globale $GA()$ riferito alle classi c_h e c_k.
La matrice M ha le seguenti proprietà:

- $\forall h,k < r$, con $h \neq k$, $M[h,k] = M[k,h] \Longleftrightarrow$ la matrice è simmetrica. Infatti, poiché le relazioni sono simmetriche (e quindi il grafo non è orientato) il *Name Affinity Coefficient* è a sua volta simmetrico; lo stesso si può dire per lo *Structural Affinity Coefficient* osservando la formula che lo genera. Di conseguenza anche $GA()$ gode della proprietà di simmetria.

- $\forall h < r$, $M[h,h]$ non ha valore: la diagonale principale, cioè, non deve essere presa in considerazione poiché non ha senso mettere a confronto una classe locale con se stessa

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

L'output di questa fase non è comunque il cluster finale, contenente tutte le classi: ben più importante è l'albero che si è definito attraverso questa procedura

di clustering, riportato in Figura 2.4. In questo albero, le foglie rappresentano tutte le classi locali rappresentate: foglie contigue sono caratterizzate da alta affinità, foglie tra loro molto lontane rappresenteranno invece concetti differenti.
Ogni nodo rappresenta un livello di clusterizzazione, ed ha associato il coefficiente di affinità tra i due sottoalberi (cluster) che unisce. In questo modo, scegliendo un valore di soglia di riferimento, si possono formare non un unico, bensì un insieme di cluster, all'interno dei quali sono raggruppate tutte le classi tra le quali esiste una affinità (rappresentata dal valore di GA) maggiore del valore soglia predefinito. Come mostrato in figura, abbiamo scelto come soglia il valore 0.5: tutte le classi di partenza delle sorgenti dell'esempio di riferimento sono state raggruppate iterativamente (attraverso la procedura esposta sopra) in cinque cluster finali.

2.2 Costruzione delle classi globali

L'ultimo passo che porta alla generazione dello schema globale, è quello che associa una classe globale ad ogni cluster determinato nella fase precedente; lo schema globale, visibile dall'utente finale e sul quale l'utente stesso porrà le interrogazioni, è formato dall'insieme di tutte queste classi.

Il processo di trasformazione di ogni cluster (definito come gruppo di classi locali ritenute affini) in classe globale, è articolato nelle seguenti fasi:

- Unione degli attributi di tutte le classi appartenenti al cluster;
- Fusione degli attributi "simili".

Questa cosiddetta "Unione ragionata" degli attributi locali, è un'operazione importante e delicata: importante perché attraverso lo schema di attributi visibile all'utente, si deve dare a quest'ultimo la possibilità di porre query semplici ma espressive; delicata perché non è immediato stabilire quali attributi debbano essere collassati in uno solo.

Le uniche informazioni che il sistema ha in merito ai rapporti tra attributi appartenenti a diverse classi (o relazioni), sono quelle memorizzate nel Thesaurus Comune (vedi sezione 2.1) generato prima della fase di creazione dei cluster.

Al termine di questa fase dell'integrazione, sviluppata nel capitolo 3, si ottengono le classi che andranno a costituire lo schema globale. L'ultimo passo di questa fase è la traduzione in ODL_{I^3} dello schema globale: in Figura 2.3 viene mostrata una parte della descrizione ODL_{I^3} di una classe globale.

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

Figura 2.3: Esempio di classe globale in ODL$_{I^3}$

2.3 Le relazioni estensionali

In questa sezione viene spiegato come le relazioni estensionali possono essere utilizzate nella integrazione di schemi in MOMIS.

Esempio di riferimento

L'esempio di riferimento utilizzato in questa sezione è costituito da due sorgenti che contengono informazioni sui ristoranti. La Figura 2.5 mostra la descrizione ODL$_{I^3}$ delle due sorgenti: la sorgente `Eating Source` (`ES`) è semistrutturata e contiene informazioni sui fast-food, con il loro menu, la qualità eccetera; la sorgente `Food Guide` (`FD`) invece è un database relazionale che contiene informazioni su vari tipi di ristoranti. Lo schema della sorgente `FD` è composto da quattro relazioni: `Restaurant`, `Bistro`, `Person` e `Brasserie`. Le istanze di `Bistro` sono un sottoinsieme delle istanze di `Restaurant` e danno informazioni sui piccoli ristoranti che servono vino. Ogni `Restaurant` e `Bistro` è gestito da una `Person`.

La traduzione in ODL$_{I^3}$

Il significato delle relazioni intensionali SYN, BT e NT tra due classi C_1 e C_2 può essere "esteso" per indicare che la relazione è anche *estensionale*. Di conseguenza, in ODL$_{I^3}$ si possono avere le seguenti relazioni estensionali:

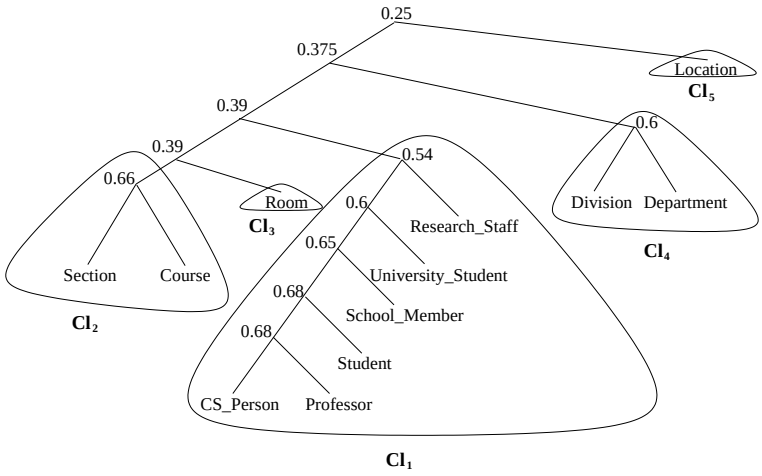


Figura 2.4: Albero di affinità

- C_1 SYN$_{ext}$ C_2: significa che le istanze di C_1 sono le stesse di C_2;
- C_1 BT$_{ext}$ C_2: significa che le istanze di C_1 sono un sovrainsieme delle istanze di di C_2;
- C_1 NT$_{ext}$ C_2: significa che le istanze di C_1 sono un sottoinsieme delle istanze di C_2;

Inoltre, le relazioni estensionali ”vincolano” le strutture di due classi C_1 e C_2, cioè C_1 NT$_{ext}$ C_2 è semanticamente equivalente a una relazione ”isa”. Riassumendo:

- una relazione estensionale C_1 NT$_{ext}$ C_2 è equivalente alla relazione C_1 ISA C_2 *più* la relazione intensionale C_1 NT C_2;
- una relazione estensionale C_1 BT$_{ext}$ C_2 è equivalente alla relazione C_1 ISA C_2 *più* la relazione intensionale C_1 BT C_2;
- una relazione estensionale C_1 SYN$_{ext}$ C_2 è equivalente alle due relazioni C_1 ISA C_2, C_2 ISA C_1 *più* la relazione intensionale C_1 SYN C_2;

In ODL$_{I^3}$una relazione ”isa” può essere espressa dal seguente vincolo di integrità:

```
rule Rule2 forall X in C1 then X in C2
```

La traduzione in OLCD

Una relazione C_1 ISA C_2 relativa ad una relazione estensionale viene espressa in ODL$_{I^3}$ dalla rule:

```
Eating_Source (ED):

 interface Fast-Food                          interface Address
 ( source semistructured Eating_Source )      ( source semistructured Eating_Source )
 { attribute string         name;             { attribute string city;
   attribute Address        address;            attribute string street;
   attribute integer        phone*;             attribute string zipcode; };
   attribute set<string>    specialty;        union
   attribute string         category;         { string; };
   attribute Restaurant     nearby*;
   attribute integer        midprice*;
   attribute Owner          owner*; };

  interface Owner
  ( source semistructured Eating_Source )
  { attribute string   name;
    attribute Address  address;
    attribute string   job; };


Food_Guide_Source (FD):

 interface Restaurant                           interface Person
 ( source relational Food_Guide                 ( source relational Food_Guide
   key r_code                                     key pers_id)
   foreign_key(pers_id) references Person )     {  attribute integer      pers_id;
 { attribute string       r_code;                 attribute string       first_name;
   attribute string       name;                   attribute string       last_name;
   attribute string       category;               attribute integer      qualification;};
   attribute string       street;
   attribute string       zip_code;
   attribute integer      pers_id;
   attribute integer      tourist_menu_price;
   attribute string       special_dish; };

 interface Bistro
 ( source relational Food_Guide                 interface Brasserie
   key r_code                                   ( source relational Food_Guide
   foreign_key(r_code) references Restaurant,     key b_code )
   foreign_key(pers_id) references Person)      {  attribute string     b_code;
 { attribute string      r_code;                   attribute string     name;
   attribute set<string> type;                     attribute string     address; };
   attribute integer     pers_id;};
```

Figura 2.5: Le sorgenti di esempio per la trattazione delle relazioni estensionali.

```
rule Rule2 forall X in C1 then X in C2
```

e viene integrata nella descrizione OLCD della classe C_1 con il costrutto $\sqcap$:

$\sigma_P(C_1) = C_2 \sqcap \ldots$

La validazione

Si consideri il seguente insieme delle relazioni estratte dagli schemi delle sorgenti di esempio e integrato in parte dal progettista:

⟨ES.Fast-Food RT ES.Owner⟩,
⟨ES.Fast-Food RT ES.Address⟩,
⟨ES.Fast-Food RT ES.Fast-Food⟩,
⟨FD.Restaurant RT FD.Person⟩,
⟨FD.Restaurant BT FD.Bistro⟩,
⟨FD.Bistro RT FD.Person⟩,
⟨ES.Fast-Food SYN_{ext} FD.Restaurant⟩,
⟨ES.Fast-Food.category BT FD.Bistro.type⟩,
⟨ES.Fast-Food.specialty BT FD.Bistro.special_dish⟩,
⟨FD.Restaurant BT FD.Brasserie⟩,
⟨FD.Person BT ES.Owner⟩,
⟨ES.Owner.name BT FD.Person.first_name⟩,
⟨ES.Owner.name BT FD.Person.last_name⟩,
⟨ES.Fast-Food.name BT FD.Person.first_name⟩,
⟨ES.Fast-Food.name BT FD.Person.last_name⟩,
⟨ES.Fast-Food.category BT FD.Restaurant.category⟩,

Il risultato delle validazione è il seguente:

⟨ES.Fast-Food.category BT FD.Bistro.type⟩	[0]
⟨ES.Owner.name BT FD.Person.first_name⟩	[1]
⟨ES.Owner.name BT FD.Person.last_name⟩	[1]
⟨ES.Fast-Food.specialty BT FD.Bistro.special_dish⟩	[1]
⟨ES.Fast-Food.name BT FD.Person.first_name⟩	[1]
⟨ES.Fast-Food.name BT FD.Person.last_name⟩	[1]
⟨ES.Fast-Food.category BT FD.Restaurant.category⟩	[0]

dove ”[1]” indica che la relazione è valida mentre ”[0]” indica il contrario.
La validazione di una relazione estensionale tra due classi C_1 e C_2 viene fatta valutando la consistenza della classe C_1. Per esempio, la relazione estensionale ⟨FD.Restaurant BT_{ext} FD.Bistro⟩ viene espressa nella classe FD.Bistro in questo modo:

$$\sigma_P(\texttt{FD.Bistro}) = \texttt{FD.Restaurant} \sqcap \triangle[\texttt{r\_code:String, type:String, pers\_id:Integer}]$$

Dal momento che la descrizione della classe FD.Bistro è consistente, la relazione tra FD.Bistro e FD.Restaurant è valida. D’altra parte, la relazione estensionale ⟨ES.Fast-Food SYN_{ext} FD.Restaurant⟩ viene rifiutata, perché la descrizione della classe:

$$\sigma_P(\texttt{ES.Fast-Food}) = \texttt{FD.Restaurant} \sqcap \ldots$$

è inconsistente (l’attributo category è definito in entrambe le classi ma in domini disgiunti).

Capitolo 3

La costruzione delle classi globali in MOMIS

In questa fase viene eseguita l'integrazione vera e propria degli schemi. Per ogni cluster individuato nella fase precedente viene costruita una classe globale.
Ogni classe è caratterizzata da:

- un **nome** che la identifica univocamente dalle altre;
- un insieme di **attributi globali**;
- una **mapping table** che indica la corrispondenza tra gli attributi globali e le informazioni contenute nelle sorgenti.

La costruzione delle classi avviene in due momenti: in un primo tempo il sistema costruisce, in modo semi-automatico, una prima versione di classi globali, con nomi, attributi globali e mapping table; successivamente il progettista interviene per apportare le modifiche che ritiene necessarie.

Dato un cluster, inizialmente viene creato un insieme d'attributi globali eseguendo l'unione di tutti gli attributi delle classi locali appartenenti al cluster: chiamiamo questo insieme iniziale *insieme-unione*. Indicando con Cl_i il cluster in esame, con c_j una classe locale appartenente ad esso e con $A(Cl_i)$ gli attributi del cluster Cl_i, l'insieme-unione sarà dato da:

$$A(Cl_i) = \bigcup^{j} A(c_j), \forall c_j \in Cl_i$$

Esempio 10 Riferendoci al cluster Cl_1 di Figura 2.4, l'unione degli attributi delle classi locali che raggruppa fornisce il seguente insieme-unione:

$A(Cl_1)$ = { `first_name, last_name, relation, e_mail, home_email, phd_email, dept_code, section_code, name, student_code, faculty_name, tax_fee, year, takes, title, belongs_to, faculty` }

Si noti come vi sia una certa ridondanza tra gli attributi: `faculty` e `faculty_name` in realtà hanno lo stesso significato di ”nome della facoltà”, così come gli attributi `first_name`, `last_name` e `name` rappresentano nomi di persona.

Alla creazione dell'insieme-unione seguono due fasi:

- *Fusione degli attributi simili*,
- *Creazione della mapping table*.

Nella prima, il sistema riconosce le affinità degli attributi in base alle relazioni del Thesaurus Comune che li legano: tutti gli attributi affini vengono *fusi* in uno solo, riducendo così le ridondanze presenti nell'insieme-unione. Nella fase successiva, vengono definite delle regole di mapping che indicano al Query Manager quali valori assume ogni attributo globale quando si accede ad una determinata classe locale del cluster.
La definizione delle regole di mapping nella seconda fase dipende da quali fusioni sono state fatte nella prima: è per questo motivo che MOMIS esegue queste due fasi contemporaneamente. Per motivi di chiarezza tuttavia verranno presentate in sezioni separate.

3.1 Fusione degli attributi simili

Il sistema individua nell'insieme-unione U dei sottoinsiemi di attributi *simili*: vale a dire attributi che, in base al Thesaurus Comune, rappresentano informazioni simili. In ogni sottoinsieme S di U viene scelto un attributo a_g e da U viene eliminato l'insieme di attributi $\{a : a \in S, a \neq a_g\}$, ovvero l'insieme di attributi simili ad a_g che per la classe globale sarebbero ridondanti: questa operazione prende il nome di *fusione* poiché è come se gli attributi a eliminati da U si siano *fusi* con a_g.
Il nome e il dominio di a_g possono essere cambiati in funzione dei nomi e dei domini degli attributi $a \in S$ simili ad a_g. In particolare, la scelta del nome può ricadere tra i nomi degli attributi $a \in S$ oppure essere il concatenamento dei nomi medesimi oppure ancora essere inserito direttamente dal progettista: deve essere significativo e univoco per la classe globale che si sta costruendo. Il dominio può cambiare, come si capirà nel seguito, in funzione del tipo di fusione attuata e se la sorgente a cui appartiene è semistrutturata.

Al termine di questa fase, U è l'insieme di attributi non ridondanti della classe globale che si sta costruendo.

3.1.1 Attributi legati da relazioni validate

Una relazione validata che lega due attributi indica che c'è compatibilità tra i domini: il dominio di un attributo coincide o contiene il dominio dell'altro oppure, se gli attributi possono avere più domini (è il caso delle sorgenti semistrutturate), almeno un dominio dell'uno coincide o contiene un dominio dell'altro.
Sono eseguiti due tipi di fusione: la prima considera gli attributi legati da relazioni validate di sinonimia (SYN) mentre la seconda considera quelle validate di specializzazione (BT o NT). Le relazioni di associazione (RT), anche se validate non vengono considerate, perché rappresentano legami molto *flebili* oppure legami di tipo *is a part of*. Pertanto se due attributi sono legati da una relazione di questo tipo non si può concludere automaticamente che le informazioni relative rappresentano concetti simili: solo il progettista può stabilirlo.

Esempio 11 Si consideri la relazione `Studenti.esami_dati RT Corso_di_laurea.tot_esami` dove i due attributi `esami_dati` e `tot_esami` sono entrambi integer: la relazione viene validata e, ai fini del calcolo delle affinità tra classi, rafforza il legame tra le classi `Studenti` e `Corso_di_laurea` tuttavia il significato delle due informazioni è diverso.

Premessa Il sistema non contempla la fusione tra due o più attributi appartenenti ad una *medesima classe*. D'altro canto, una tale situazione potrebbe significare due cose:

- o la classe comune agli attributi è stata progettata male,
- oppure le relazioni che legano gli attributi in questione sono sbagliate.

Grafi di relazioni

Per eseguire la ricerca degli attributi simili il sistema costruisce dei grafi in cui i vertici sono gli attributi e gli archi sono relazioni validate del Thesaurus Comune che legano attributi appartenenti al cluster.
Il numero di grafi costruiti dipende dal numero degli attributi dell'insieme-unione U iniziale e dalle relazioni validate (x, t, y) del Thesaurus Comune T, dove x, y sono attributi e t è il tipo della relazione (SYN, BT, NT, RT).
Considerato un qualsiasi attributo $a \in U$, il grafo $G(V, A)$ costruito per a è così definito:

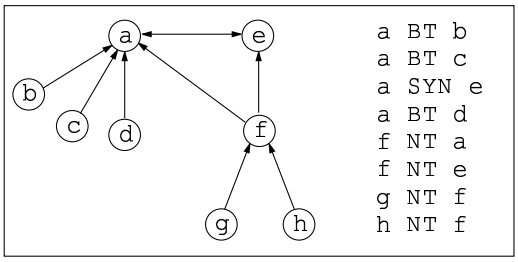


Figura 3.1: Grafo di relazioni: esempio

$$\begin{aligned} \text{V} &= \{v \in U : \exists C(a,v) \vee C(v,a)\} \\ \text{A} &= \{(x,y) : (x,y) \in C(r,s), \forall r,s \in V\} \end{aligned}$$

dove $C(a,v)$ è il cammino costituito da relazioni validate SYN, BT o NT che va dall'attributo a all'attributo v, V è l'insieme dei vertici e A l'insieme degli archi.

Esempio 12 La Figura 3.1 mostra un grafo di esempio: il grafo è stato costruito sulla base di relazioni date, partendo da un attributo qualsiasi. Nella rappresentazione grafica, ogni relazione di tipo BT è stata trasformata in una relazione equivalente NT, rappresentate con frecce unidirezionali che vanno dall'attributo specializzato a quello generalizzante, mentre le relazioni SYN sono rappresentate da una doppia freccia.

Attributi legati da relazioni validate di sinonimia.

Quando due o più attributi, appartenenti ciascuno a una classe locale diversa, sono legati da relazioni validate di sinonimia (SYN) significa che i nomi dei due attributi possono essere scambiati poiché identificano lo stesso concetto del mondo reale: essendoci anche compatibilità tra i domini, gli attributi possono essere fusi. L'insieme di tali attributi costituisce dunque un sottoinsieme di U di attributi simili.
Il dominio assunto dall'attributo a_g scelto nel sottoinsieme è determinato dai domini degli attributi fusi:

- se gli attributi hanno un unico dominio la scelta ricade su quello che li contiene tutti;
- se ci sono attributi con più domini, il sistema propone al progettista di scegliere il dominio da un elenco che contiene tutti i possibili domini che generalizzano quelli degli attributi fusi.

Esempio 13 Si supponga che il sistema abbia individuato tre attributi sinonimi legati dalle relazioni:

```
faculty_name SYN faculty
faculty SYN name_of_faculty
```

Nell'ipotesi che `faculty_name` sia di tipo `string[30]`, `faculty` di tipo `string[40]` e `name_of_faculty` di tipo `string[100]`, si ottiene la fusione dei tre attributi in uno solo il cui dominio dovrà essere `string[100]`.

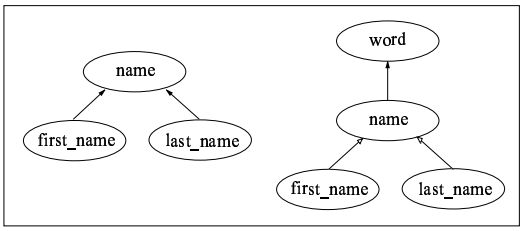


Figura 3.2: Esempi di gerarchia tra attributi

Attributi legati da relazioni validate di specializzazione.

Dai grafi vengono estratte tutte quelle gerarchie di attributi costruite sulla base delle relazioni di specializzazione BT o NT: gli attributi dei livelli più bassi sono fusi con l'attributo che sta in cima alla gerarchia.

Esempio 14 Nella Figura 3.1.1 si vedono due esempi di queste gerarchie:

- nel primo caso gli attributi `first_name` e `last_name` saranno fusi con l'attributo `name`;
- nel secondo gli stessi attributi verranno fusi in `word` assieme all'attributo `name`.

Per la fusione di gerarchie di attributi rimangono valide le considerazioni sui domini enunciate nella sezione dedicata alla fusione di attributi sinonimi.

Problema degli attributi sinonimi specializzati

Se più attributi legati da relazioni SYN compaiono in relazioni di specializzazione di tipo NT, allora si presenta il problema di quale fusione attuare.

Esempio 15 Si consideri il caso presentato in Figura 3.1.1. Non è possibile stabilire automaticamente il tipo di fusione che porta alla migliore integrazione solamente sulla base delle relazioni che legano gli attributi: occorre l'intervento del progettista. In particolare, dall'insieme d'attributi iniziale {`first_name`, `last_name`, `name`}:

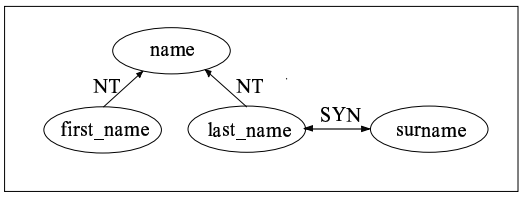


Figura 3.3: Esempio di ambiguità nella scelta della modalità di fusione

- se si esegue la fusione degli attributi legati da relazioni si specializzazione allora è possibile eliminare gli attributi `first_name` e `last_name`, che verrebbero fusi nell'attributo `name`:

  $$\left.\begin{array}{l}\texttt{first\_name}\\ \texttt{last\_name}\end{array}\right\} \longrightarrow \texttt{name}$$

  e si ottiene l'insieme di attributi {`name`, `surname`};

- se invece si esegue la fusione degli attributi legati da una relazione di sinonimia, `last_name` con `surname`, si ottiene l'eliminazione di un solo attributo (`surname` o `last_name`): questo perché la fusione dei due attributi rimanenti, `first_name` e `name`, anche se legati da una relazione di specializzazione, non sarebbe corretta, in quanto esprimono concetti diversi. L'insieme risultante è {`name`, `first_name`, `surname`}.

Il sistema risolve il problema eseguendo comunque la fusione tra attributi sinonimi: questo perché le relazioni di sinonimia costituiscono un legame più forte delle relazioni di specializzazione. In ogni caso, terminata la definizione semi-automatica della classe globale, il progettista può sempre intervenire per apportare delle modifiche.

3.1.2 Attributi legati da relazioni non validate

La fusione di attributi coinvolti in relazioni validate è un'operazione automatica e solo in determinati casi è richiesto l'intervento del progettista. Lo stesso non si può dire nel caso di relazioni non validate, per le quali il problema è più complesso e l'intervento del progettista è indispensabile.
Le relazioni non validate del Thesaurus indicano che nonostante i concetti espressi dai nomi degli attributi siano correlati, l'analisi sui domini mostra che gli attributi in questione sono incompatibili. Eseguire la fusione automatica di attributi sulla base di queste relazioni diventerebbe quantomai azzardato.
Per capire meglio il problema, prendiamo in esame l'esempio di riferimento. Una delle relazioni non validate è la seguente:

`UNI.Research_Staff.dept_code BT CS.Professor.belongs_to`

Questa relazione non è valida perché l'attributo `dept_code` appartiene all'insieme degli interi, mentre `belongs_to` è una collezione di OID che realizza una associazione tra le classi `CS.Professor` e `CS.Division`. Tuttavia, un'analisi attenta dei sorgenti, mostra che l'attributo `dept_code` è definito come foreign key nella relazione `UNI.Research_Staff` (`UNI` è una sorgente relazionale) e realizza una aggregazione fra le relazioni `UNI.Research_Staff` e `UNI.Department`.

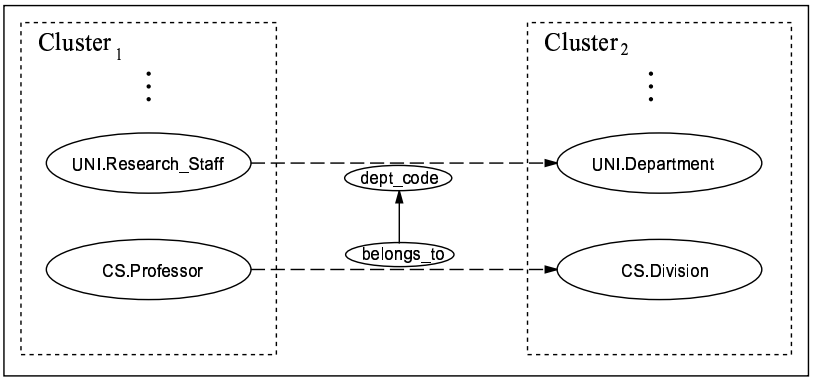


Figura 3.4: Fusione di attributi contenuti in relazioni non valide

Dalla figura 3.4, che mostra graficamente il significato complesso dei due attributi, si nota che le classi *puntate* dai due attributi appartengono entrambe allo stesso cluster. In tal caso è opportuno che i due attributi vengano fusi in uno solo:

$$\texttt{dept\_code, belongs\_to} \longrightarrow \texttt{works}$$

In questo caso, il dominio assunto dall'attributo globale sarà un dominio particolare che il gestore delle interrogazioni riconoscerà e si comporterà di conseguenza.

Possiamo quindi definire le condizioni affinché sia possibile raggruppare attributi appartenenti a relazioni non valide:

1. gli attributi devono rappresentare entrambi una gerarchia di aggregazione (collezioni di attributi complessi o foreign key);

2. le rispettive classi locali mappate devono appartenere allo stesso cluster e la stessa condizione deve valere anche per le classi di appartenenza, ma questo è sempre vero perché, per la fusione, vengono considerate solamente le relazioni che legano attributi appartenenti ad un medesimo cluster;

3. gli attributi devono avere lo stesso significato semantico: condizione garantita se esiste una relazione di sinonimia o di specializzazione che li lega.

3.2 Generazione della *mapping table*

Il Query Manager, per eseguire una query formulata sullo schema globale, deve sapere a quali sorgenti deve accedere: questa conoscenza viene rappresentata in MOMIS attraverso le *mapping-table*.
Ogni mapping-table è una tabella $MT[C][A]$ dove C é l'insieme delle classi locali che appartengono al cluster a cui la mapping table si riferisce e A è l'insieme degli attributi globali ottenuto dopo la fusione degli attributi simili. Indicando con $c_l \in C$ il nome di una classe locale e con $a_g \in A$ il nome di un attributo globale, ogni elemento $MT[c_l][a_g]$ della tabella contiene una *mapping-rule* ODL_{I^3}: una regola che indica il tipo di legame tra l'attributo globale a_g e le informazioni della classe c_l ovvero quali valori assume a_g ogni volta che il Query Manager accede a c_l. Non è detto tuttavia che venga definita una mapping-rule per ogni coppia attributo globale-classe locale: può anche essere che un attributo globale non assuma alcun valore in corrispondenza ad una certa classe locale.

Come si vedrà meglio nel seguito, le mapping-rule dipendono dal tipo di fusione attuata per generare l'attributo globale a cui si riferiscono. È pertanto evidente come la generazione delle mapping table debba essere eseguita contemporaneamente alla fusione degli attributi.

3.2.1 Mapping degli attributi

Vediamo quali sono i tipi di mapping-rule e la corrispondente rappresentazione nella mapping table. Per gli esempi citati si faccia riferimento alla Figura 3.5.

Corrispondenza semplice

Questa mapping rule indica che l'attributo globale a_g per la classe locale c_l assume i valori di un solo attributo locale a_l. Nella mapping table, la casella $MT[c_l][a_g]$ assumerà il nome dell'attributo a_l.
Questo tipo di mapping si applica quando:

- a_g rappresenta solamente le informazioni dell'attributo a_l per la classe c_l e non assume alcun valore per tutte le altre classi. Esempio: l'attributo locale `employee_nr` della classe `CS.Division` non è coinvolto in nessuna relazione del Thesaurus Comune per cui deve essere mappato in un attributo

globale specifico, che sarà caratterizzato dallo stesso dominio e assumerà lo stesso nome;

- l'analisi di relazioni di sinonimia porta alla fusione di più attributi, appartenenti a classi locali diverse, in un unico attributo. É il caso degli attributi `faculty_name` e `faculty`, delle classi `TP.University_Student` e `UNI.School_Member`, che sono fusi nell'attributo globale `faculty`.

- l'analisi delle relazioni di specializzazione porta alla fusione di più attributi, appartenenti a classi locali diverse, in un unico attributo. Ad esempio, gli attributi `dept_code` e `belongs_to`, rispettivamente delle classi `UNI.Research_Staff` e `CS.Division`, sono fusi nell'attributo globale `works`.

Corrispondenza in *and*

Questa corrispondenza viene usata in concomitanza alla fusione di più attributi appartenenti ad una medesima classe locale c_l e i valori che deve assumere l'attributo globale a_g sono il concatenamento dei valori contenuti negli attributi fusi. Nella mapping-table, la corrispondente casella $MT[c_l][a_g]$ assume la seguente espressione:

nome_att1 `and`
nome_att2 `and`
`...`

dove *nome_att1*, *nome_att2*, ... sono i nomi degli attributi da cui sono estratte le informazioni da concatenare.
Ad esempio, gli attributi `first_name` e `last_name`, appartenenti alla medesima classe locale `UNI.Research_Staff`, sono fusi nell'attributo `name`.

Corrispondenza in *union*

É una corrispondenza analoga a quella in and. La differenza è che, in questo caso, l'attributo globale a_g può assumere il valore di *uno solo* tra gli attributi fusi appartenenti alla stessa classe c_l. La scelta del valore che l'attributo globale assume avviene quando il Query Manager deve eseguire l'integrazione delle risposte ottenute dalle sorgenti, ed è determinata dal valore di un altro attributo locale, denominato *tag*, appartenente sempre alla classe c_l.
Nell'esempio di riferimento, per avere una integrazione corretta, l'attributo globale `email` deve assumere il valore di `home_email` o `phd_email` a seconda del valore assunto dall'attributo locale `rank` (attributo *tag*), appartenente anch'esso

alla classe locale `CS.Student`: in particolare, se `rank` = `'course'` allora `email` deve assumere il valore di `home_email` altrimenti se `rank` = `'phd'` allora `email` deve assumere il valore di `phd_email`.
Come si può constatare nella mapping table di esempio, in questa caso la casella $MT[c_l][a_g]$ conterrà la seguente espressione:

`case` *nome_att_tag* `of`
cost1 `:` *nome_att1*
cost2 `:` *nome_att2*
`...`

dove *cost1*, *cost2*, ... sono le costanti che permettono al Query Manager di selezionare il valore opportuno da assegnare all'attributo globale e *nome_att1*, *nome_att2*, ... sono i nomi degli attributi locali coinvolti nel mapping.

Valori di default

A volte succede che alcune informazioni sono presenti negli schemi delle sorgenti sottoforma di metadato e il sistema non le considera. MOMIS permette di risolvere il problema dando al progettista la possibilità di assegnare opportuni valori costanti agli attributi globali in corrispondenza di determinate classi: ciò viene fatto attraverso i valori di default e la casella $MT[c_l][a_g]$ della mapping-table assumerà l'espressione di tali valori costanti.
Nell'esempio di riferimento, l'attributo globale `rank` assume il valore costante `'Professor'` nel caso in cui si acceda alla classe `UNI.Research_Staff` e il valore `'Student'` se si accede alla classe `UNI.School_Member` o TP.University_Student.

Nessuna corrispondenza

Questo é il caso in cui il Query Manager deve accedere alla classe c_l e l'attributo globale a_g non assume alcun valore.
Per esempio, l'attributo globale `faculty` non assume alcun valore nella classe locale `UNI.Research_Staff`: ciò viene indicato nella mapping table con la costante $null$.

3.2.2 Revisione degli attributi

A volte l'unione e la successiva fusione di attributi non sono sufficienti per rappresentare, nello schema globale, alcune informazioni ricavabili dalla struttura dei singoli schemi locali. In particolare, potrebbe risultare impossibile isolare un concetto tra le diverse entità contenute all'interno di una stessa classe globale.

University_Person	name	rank	works	faculty	email	...
Research_Staff	first_name *and* last_name	'Professor'	dept_code	null	email	...
School_Member	first_name *and* last_name	'Student'	null	faculty	null	...
CS_Person	name	null	null	'Computer_Science'	null	...
Professor	name	rank	belongs_to	'Computer_Science'	null	...
Student	name	rank	null	'Computer_Science'	*case* rank *of* 'course':home_mail 'phd':phd_mail	...
University_Student	name	'Student'	null	faculty_name	null	...

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

Figura 3.5: Alcune mapping-table create con l'esempio di riferimento: le classi globali `University_Person` e `Workplace`

Infatti, per come è costruito lo schema, ogni classe globale ottenuta è l'unione intensionale ed estensionale di alcune classi locali, invisibili individualmente a chi usufruisce dello schema integrato.
L'aggiunta di uno o più attributi globali da parte del progettista risolve il problema e arricchisce ulteriormente il contenuto informativo dello schema integrato. Spesso gli attributi locali aggiunti vengono mappati con valori di default poiché le informazioni contenute negli schemi locali sottoforma di metadato non possono essere ricavate automaticamente dal sistema.

Esempio 16 Supponiamo che, nell'esempio di riferimento, la sorgente `University` rappresenti il database dell'università di Modena e che `Computer_Science` rappresenti quello dell'università di Reggio Emilia. La conoscenza della sede universitaria a cui è iscritto uno studente è nota a livello dei singoli sorgenti, perché non compare tra le informazioni del database. Allora per poter distinguere le persone che lavorano presso una università piuttosto che nell'altra, si può rappresentare la sede universitaria aggiungendo un'attributo `uni_location` alla classe globale `University_Person`. I mapping dell'attributo con le classi locali saranno valori di default: ad esempio, il valore `'Modena'` in corrispondenza delle classi locali appartenenti alla sorgente `University` e il valore `'Reggio Emilia'` in corrispondenza delle classi della sorgente `Computer_Science`.

Capitolo 4

SI-Designer: il tool per l'integrazione di sorgenti distribuite in MOMIS

SI-Designer è l'interfaccia grafica di MOMIS che permette al progettista di eseguire integrazione di schemi ODL_{I^3} provenienti da basi di dati eterogenee. Le operazioni che il tool deve eseguire sono quelle già trattate nel capitolo 2.

In questo capitolo verrà illustrato come è stata ridisegnata l'architettura del tool rispetto al prototipo precedente. Nella prima parte viene descritto il prototipo di SI-Designer preesistente, specificandone le caratteristiche salienti e i motivi che hanno portato ad una sua riprogettazione. La seconda parte invece descrive l'architettura del nuovo prototipo di SI-Designer, con un approfondimento sui moduli preposti alla realizzazione delle fasi di acquisizione delle sorgenti e creazione delle classi globali.

4.1 Analisi critica del primo prototipo

La prima versione di SI-Designer è stata sviluppata presso questa università dall'Ing. Alberto Zanoli come tesi di laurea [32]. La Figura 4.1 mostra l'architettura del primo prototipo del Global Schema Builder, la parte del mediatore di MOMIS che si preoccupa della costruzione dello schema globale. Si notano tre componenti:

- *SI-Designer*: il modulo in questione, che realizza l'interfaccia fra il sistema e il progettista, coordinando l'esecuzione dei diversi software che partecipano all'integrazione:

  - ODB-Tools per l'estrazione e la validazione delle relazioni terminologiche,
  - WordNet per l'estrazione automatica di relazioni lessicali tra i nomi di classi ed attributi,
  - ARTEMIS per il calcolo delle affinità;

- *SIM_1A e SIM_1B*: descritti in [3] e analizzati nella sezione 4.1.2, partecipano alla generazione del Thesaurus Comune estraendo le relazioni intra-schema da ogni singola sorgente (SIM_1A) e, con l'aiuto di ODB-Tools , validano tutte le relazioni terminologiche e ne inferiscono di nuove (SIM_1B);
- *Parser ODL_{I^3}*: descritto nella sezione 4.1.3, effettua l'analisi degli schemi sorgenti, verificandone la consistenza, ed archivia tutte le informazioni sugli schemi sorgenti e sullo schema integrato.

Vediamo nel dettaglio quali sono le caratteristiche implementative di ciascun componente.

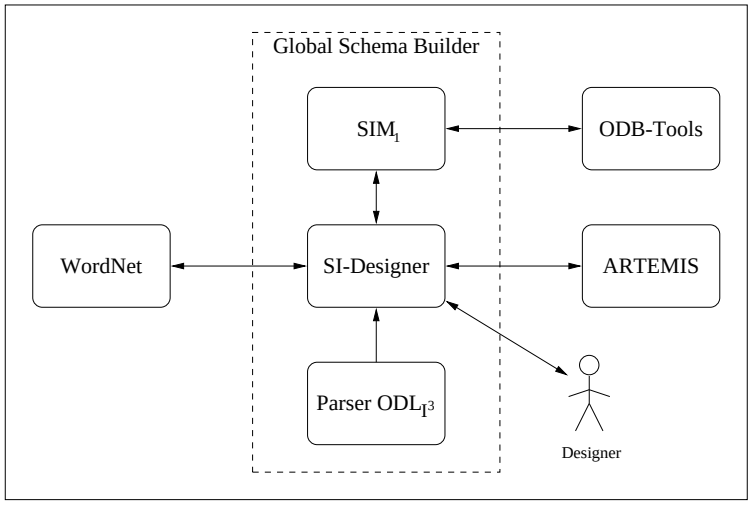


Figura 4.1: L'architettura del Global Schema Builder nel prototipo preesistente

4.1.1 SI-Designer

L'interfaccia grafica implementata presenta una sequenza di pannelli (uno per ogni fase): partendo dal primo il progettista passa da un pannello all'altro fino a completare l'integrazione delle sorgenti. La sequenza dei pannelli è predefinita ed è unidirezionale ovvero non è possibile tornare ad una fase precedente ma passare solamente a quella successiva. Quando il modulo è stato progettato, non era

ancora stata definita l'architettura dei wrapper, cosicché SI-Designer inizia l'acquisizione degli schemi (attraverso il parser ODL_{I^3}, sezione 4.1.3) leggendo file che contengono le descrizioni ODL_{I^3} delle sorgenti da integrare. Poi si passa alle fasi successive dell'integrazione, vale a dire:

- estrazione automatica delle relazioni (instensionali ed estensionali),
- integrazione/revisione delle relazioni estratte,
- validazione del Thesaurus Comune,
- inferenza di nuove relazioni,
- calcolo delle affinità,
- generazione dei cluster,
- creazione delle mapping table.

Occorre sottolineare tuttavia che SI-Designer, nell'estrazione delle relazioni, esegue una interazione molto rudimentale con i moduli SIM e WordNet, ottenendo quindi un Thesaurus non particolarmente ricco. Inoltre, della fase di creazione delle classi globali è stata implementata solo la complessa interfaccia grafica e non il codice che la esegue.

Il tool grafico è stato realizzato in linguaggio Java e i motivi che hanno portato a scegliere tale linguaggio implementativo si possono così riassumere:

1. *Programmazione ad oggetti.* Gli schemi locali da integrare sono descritti nel linguaggio ad oggetti ODL_{I^3} per cui è più semplice trattare le relative informazioni con un linguaggio ad oggetti come Java piuttosto che con un linguaggio strutturato[1].
2. *Portabilità di Java.* Esiste l'interprete Java per moltissime piattaforme hw/sw.
3. *Minor lavoro di programmazione.* L'interprete Java ha la caratteristica di deallocare automaticamente la memoria per le istanze degli oggetti che non sono utilizzate (tale meccanismo è chiamato *garbage collection*): ciò rende più semplice la scrittura del codice e permette al programmatore di concentrarsi maggiormente sulla codifica dell'algoritmo da implementare[2].

[1] L'ANSI C, ad esempio, avrebbe comportato una certa mole di lavoro nella creazione di strutture dati atte a contenere oggetti o strutture più complesse.

[2] In questo senso, l'uso dell'ANSI C++, nonostante sia anch'esso un linguaggio ad oggetti, avrebbe comportato un lavoro aggiuntivo per gestire la deallocazione delle istanze di oggetti che non sono più utilizzate.

4. *Larga diffusione di Java per applicazioni che utilizzano Internet.* L'obiettivo di MOMIS[3] è sviluppare un sistema a *mediatore* che integri diverse sorgenti di informazione eterogenee e distribuite, sia strutturate che semistrutturate: Internet è piena di sorgenti di questo tipo e la necessità di accedere in modo integrato a tali informazioni è pressante, vista la continua espansione della rete.

Osservazioni. L'uso di un linguaggio ad oggetti come Java permette di creare facilmente applicazioni modulari semplificando eventuali modifiche al codice già creato: in un progetto complesso come il sistema MOMIS, del quale non è ancora terminata la fase di ricerca, ciò è senz'altro un pregio. Per quanto riguarda il funzionamento dell'interfaccia, il fatto che il progettista non possa tornare indietro ad una fase di integrazione precedente può costituire un problema, poiché se il progettista vuole rimediare ad un errore commesso, deve ripetere l'intera sequenza di fasi fino a quel punto. Occorre inoltre migliorare lo sfruttamento dei moduli WordNet e SIM per la costruzione del Thesaurus Comune e implementare la fase di creazione delle mapping table.

4.1.2 I moduli SIM_1A e SIM_1B

Entrambi i moduli sono stati realizzati in linguaggio ANSI C.

SIM_1A

Analizza gli schemi locali ed estrae le relazioni terminologiche in tre fasi successive:

1. *acquisizione degli schemi*: controlla la sintassi degli schemi ODL_{I^3} e popola, attraverso un proprio parser interno (differente quindi dal parser descritto nella sezione 4.1.3), una struttura dati ad oggetti che memorizza gli schemi locali da analizzare;

2. *controllo di coerenza*: rileva eventuali incongruenze nella descrizione degli schemi locali come, ad esempio, la presenza di foreign key che non hanno attributi corrispondenti nella classe referenziata;

3. *analisi*: estrae le relazioni intra-schema analizzando le sorgenti.

L'analisi degli schemi avviene attraverso due procedure distinte, una per tipo di sorgente: l'analisi delle classi relative agli schemi relazionali viene fatta

[3]É parzialmente finanziato dall'*Italian MURST ex-40% INTERDATA project - Metodologie e Tecnologie per la Gestione di Dati e Processi su Reti Internet e Intranet*.

direttamente dal modulo SIM$_1$A mentre per l'analisi degli schemi ad oggetti il modulo si appoggia ad un tool esterno, OCDL-Designer (degli ODB-Tools). Il tool esterno OCDL-Designer è in grado di analizzare solamente schemi ad oggetti rappresentati in linguaggio OCDL (Object Constraint Description Logics): dunque è stato necessario realizzare anche un traduttore dal linguaggio ODL$_{I^3}$ al linguaggio OCDL.

Osservazioni. Il modulo è altamente coeso perché riceve in input una qualsiasi descrizione ODL$_{I^3}$ di schemi e fornisce in output l'insieme delle relazioni intra-schema estratte, espresse anch'esse in linguaggio ODL$_{I^3}$. Ci sono tuttavia alcune carenze importanti che limitano fortemente l'utilizzo del modulo in applicazioni reali:

1. se negli schemi di due sorgenti distinte compaiono classi che hanno attributi con lo stesso nome, SIM$_1$A segnala errore: ciò è evidentemente sbagliato ed è limitativo perché la probabilità che si presenti un'eventualità del genere è molto alta;

2. SIM$_1$A è stato sviluppato nel 1996 e nel frattempo la sintassi del linguaggio ODL$_{I^3}$ è cambiata;

3. non vengono riconosciute le sorgenti semistrutturate;

4. il legame tra due classi "relazionali" con foreign key viene espresso attraverso le sole relazioni terminologiche di tipo RT: dunque non riconosce le gerarchie negli schemi di sorgenti relazionali;

5. è un modulo realizzato in C: quindi necessita di ricompilazione per essere utilizzato su piattaforme diverse e non è così portabile come il linguaggio Java, utilizzato per implementare SI-Designer.

SIM$_1$B

Esegue due delle operazioni utili all'integrazione di schemi:

- *validazione del Thesaurus Comune*: ricevuti in input gli schemi delle sorgenti e il Thesaurus Comune, attraverso l'interazione con ODB-Tools esegue la validazione delle relazioni, restituendo in output le medesime relazioni, distinguendo quelle relazioni validate da delle che non lo sono;

- *inferenza di nuove relazioni*: dagli schemi e dalle relazioni del Thesaurus Comune ricevuti in input, restituisce l'insieme di relazioni inferite tramite ODB-Tools .

I due moduli, anche se separati, utilizzano le medesime librerie, poiché entrambi interagiscono con ODB-Tools .

Osservazioni. Le stesse del modulo SIM_1A.

4.1.3 Il parser ODL$_{I^3}$

L'obiettivo principale di un parser è acquisire delle informazioni da un testo. Per fare questo il parser deve controllare che il testo soddisfi determinate regole sintattiche e grammaticali, eseguendo anche alcuni controlli semantici sulla consistenza delle informazioni acquisite. L'output del parser è una struttura dati riempita in base alle informazioni estratte dal testo.
La struttura dati, utilizzata dai componenti del mediatore, deve avere le seguenti caratteristiche:

- essere rappresentativa di tutta la conoscenza che il linguaggio ODL$_{I^3}$ può esprimere;
- essere facilmente utilizzabile nei componenti che devono attingere alle informazioni memorizzate.

Realizzato in linguaggio Java, questo parser viene utilizzato in modo particolare da SI-Designer per acquisire le descrizioni ODL$_{I^3}$ delle sorgenti e dal Query Manager nell'acquisizione degli schemi globali e locali.

Facendo riferimento alla Figura 4.2 e all'appendice B, di seguito verranno descritte le principali caratteristiche del linguaggio ODL$_{I^3}$ e come le informazioni contenute in una descrizione ODL$_{I^3}$ sono implementate nella complessa struttura dati (si veda [32] per una trattazione più dettagliata).

I tipi

L'ODL$_{I^3}$ distingue i tipi in due famiglie: i *tipo-valore* e i *tipo-classe*: le classi che appartengono ad entrambe le famiglie sono figlie della classe **Type**.

I tipo-valore Sono i tipi propri delle variabili e degli attributi semplici, che hanno un unico valore. Di questa famiglia fanno parte:

- i tipi atomici base (**SimpleType**), come integer, float, char e boolean, e i tipi collezione (**TemplateType**), quali set, list, bag, array, nonché i tipi nuovi (**DefinedType**);
- i tipi che rappresentano le strutture dati complesse (**StructType**, **UnionType** e **EnumType**).

I tipo-classe L'altra famiglia in cui sono suddivisi i tipi dell'ODL$_{I^3}$ è quella dei *tipo-classe*: questi sono propri degli attributi complessi, delle viste e delle istanze di classi (che in ODL$_{I^3}$ sono chiamate *interface*), con OID, interfaccia e comportamento. A questa famiglia appartiene l'unica classe **Interface**. In ODL$_{I^3}$ è prevista l'ereditarietà multipla ed è per questo che nella classe **Interface** la proprietà `inheritance` è un *set* e non un valore semplice.
Ogni classe ODL$_{I^3}$ può avere diverse proprietà: una sorgente di appartenenza (**Source**), una lista delle chiavi che identificano univocamente gli oggetti della classe (**KeyList**), delle foreign key (**ForeignKey**) (se la interface appartiene ad una sorgente relazionale). In ODL$_{I^3}$ sono previste inoltre più implementazioni (**IntBody**) per una stessa interface: con *implementazione* si intende un insieme di attributi e metodi.

Le implementazioni di interface.

Ogni implementazione di una interface può essere caratterizzata da diverse proprietà: attributi locali (**SimpleAttribute**), definiti nelle sorgenti da integrare; attributi globali (**GlobalAttribute**), che caratterizzano le classi dello schema globale; relazioni con altre interface (**Relationship**); oppure ancora metodi locali (**Operation**).
In ODL$_{I^3}$ sono definite poi delle *mapping-rule* (vedere capitolo 3) che indicano quali valori assume l'attributo di una classe globale quando il Query Manager deve accedere ad una classe locale: tali regole (**MappingRule**) sono associate agli attributi globali (**GlobalAttribute**). Ci sono diversi tipi di corrispondenza tra gli attributi locali e gli attributi globali:

- corrispondenza semplice (**AttributeOnly**);
- corrispondenza in *and* (**AndList**);
- corrispondenza in *union* (**UnionList**);
- valore di default costante (**DefaultValue**).

Le relazioni terminologiche

In ODL$_{I^3}$, le relazioni terminologiche (**ThesRelation**) possono rappresentare: un legame tra due interface (**InterfaceRel**), tra due attributi (**AttributeRel**) oppure una interface e un attributo (**IntAttrRel**).

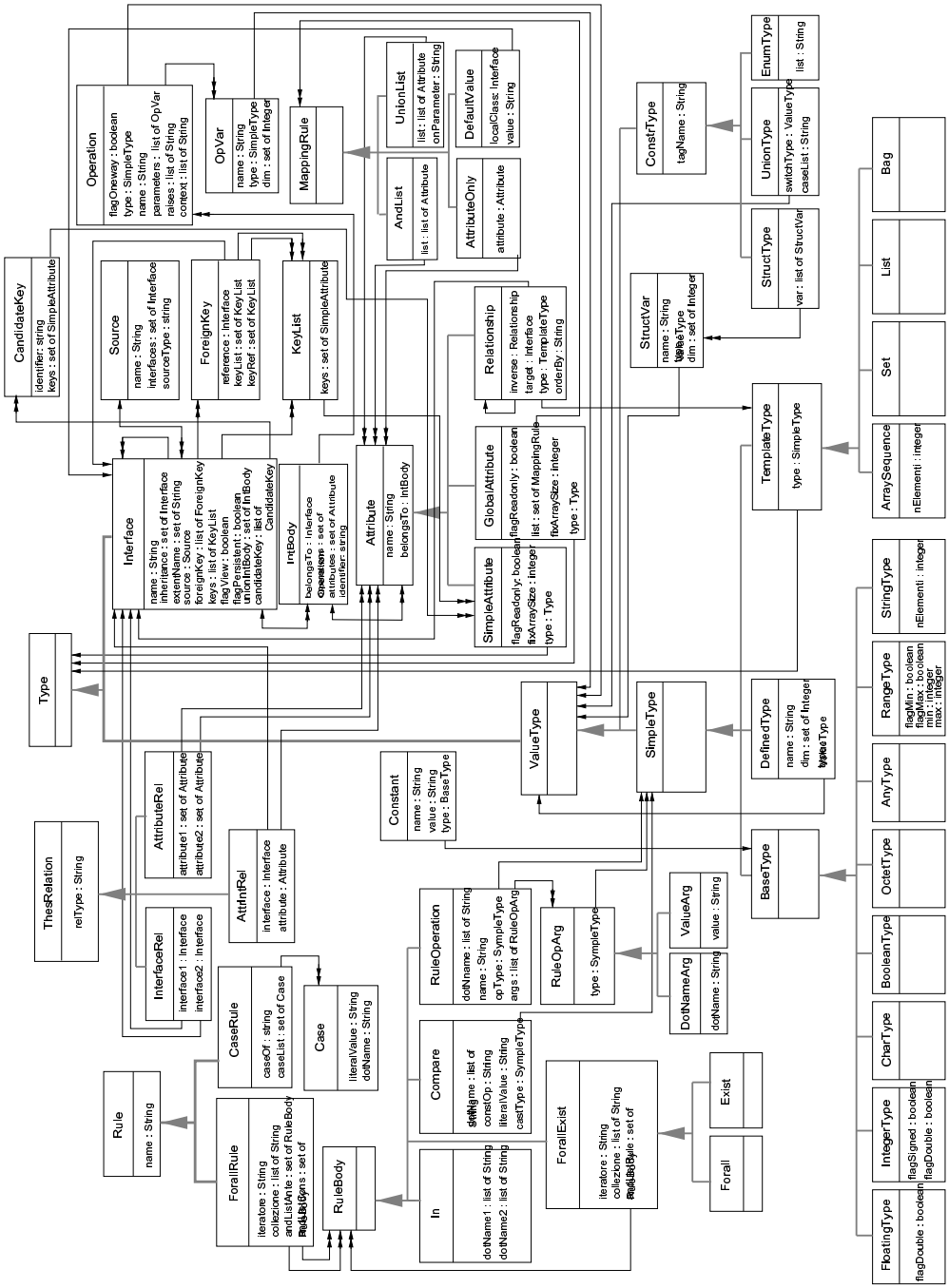


Figura 4.2: Il modello ad oggetti della struttura dati ODL$_{I^3}$

Le regole di integrità

L'ODL$_{I^3}$ prevede la possibilità di definire regole *if-then* del tipo:

rule *nomerule*
forall *iteratore* **in** *collezione* **:** *antecedente* **then** *conseguente*

oppure:

rule *nomerule*
[{ **case of** *identifier* **:** *caselist* }]

Nella struttura dati le regole sono rappresentate dalle classi **ForAllRule** e **CaseRule** che appartengono ad una gerarchia in cui la classe padre è **Rule**.

Osservazioni. Il parser ODL$_{I^3}$ deve essere aggiornato, sia nella struttura dati che nel riconoscitore sintattico, poiché dal momento in cui è stato progettato sono state apportate delle modifiche alla sintassi dell'ODL$_{I^3}$. In particolare, è stata introdotta la possibilità di definire delle chiavi candidate:

⟨c_key_spec⟩ : : = **candidate_key** ⟨identifier⟩
(⟨key_list⟩**)**

A tale modifica si aggiunge anche quella relativa alla sintassi per definire le varie implementazioni che una stessa interface può assumere rendendo obbligatorio associare un identificatore ad ogni implementazione:

⟨interface_dcl⟩ : : = ⟨interface_header⟩
{[⟨ interface_body⟩]};
[**union** ⟨identifier⟩ { ⟨interface_body⟩ }**;**]

4.1.4 Considerazioni finali

Da quanto esposto, si può concludere che:

- SI-Designer deve essere aggiornato per due motivi:

  1. lo scopo di MOMIS è la creazione di un sistema integratore di sorgenti, soprattutto quelle disponibili in rete: occorre quindi implementare l'accesso a tali sorgenti attraverso i wrapper remoti (localizzati sulla stessa macchina in cui si trova la sorgente per la quale sono stati creati);

2. migliorare l'uso dei tool esterni quali ODB-Tools e WordNet, in modo da ottenere un Thesaurus Comune più ricco, a vantaggio di una migliore integrazione delle informazioni.

- I moduli SIM_1A e SIM_1B hanno un utilizzo reale molto limitato. Per renderli utilizzabili ci sono due possibilità: modificare il codice C del prototipo oppure riscrivere il codice in linguaggio Java. La seconda è senz'altro la strada migliore perché:
  - SI-Designer utilizza un parser scritto in Java mentre i moduli SIM utilizzano un loro parser interno scritto in C: ogni modifica futura al linguaggio ODL_{I^3} (che non è standard) comporterebbe l'apporto di modifiche ad entrambi i parser, scritti in linguaggi diversi, mentre con un unico parser si risparmierebbe metà del lavoro (conviene utilizzare il parser scritto in Java poiché è più completo ed è scritto in un linguaggio ad oggetti).
  - Ulteriori evoluzioni future dei moduli SIM comporterebbero la scrittura di codice in un linguaggio che non è ad oggetti: essendo l'ODL_{I^3} ad oggetti risulterebbe più semplice utilizzare Java.
  - Le carenze che presentano i due moduli sono importanti e le modifiche da apportare al codice per eliminarle comporterebbero comunque una mole di lavoro non indifferente.
  - Il linguaggio Java permette la creazione di applicazioni altamente portabili.
- Il parser ODL_{I^3} necessita di un aggiornamento per allinearlo alla nuova sintassi di ODL_{I^3}.

4.2 Progettazione e realizzazione del software

Per implementare l'interazione di SI-Designer con i wrapper è stata adottata l'architettura CORBA: in questo modo i wrapper sono dei *servant-object* (vedi appendice C) che rispondono ai messaggi inviati dal client SI-Designer. La stessa architettura è stata poi adottata anche per utilizzare i tool esterni: WordNet e ODB-Tools sono stati *incapsulati* i oggetti CORBA, rendendo uniforme l'interazione di SI-Designer con il mondo esterno. Rispetto al precedente prototipo di SI-Designer, il modulo ARTEMIS (scritto in Java) è stato inglobato all'interno di SI-Designer: l'utilizzo di ARTEMIS come modulo esterno infatti avrebbe comportato il trasferimento dell'intera struttura dati dello schema globale e degli

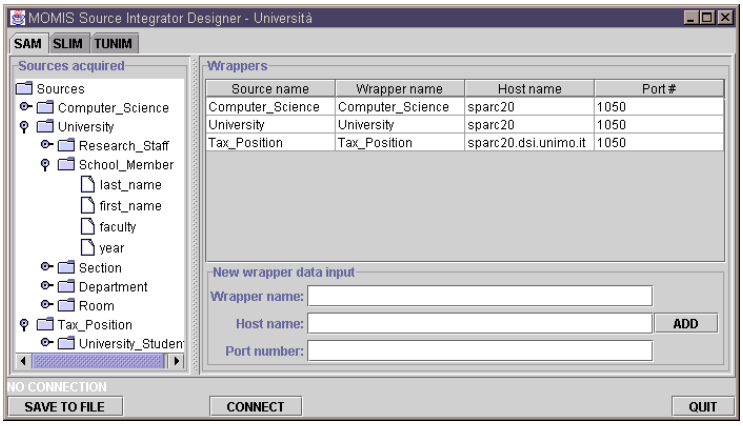


Figura 4.3: La schermata iniziale di SI-Designer: il modulo SAM

schemi locali da SI-Designer ad ARTEMIS stesso, con conseguente dilatazione dei tempi necessari alla creazione dello schema integrato.

Contemporaneamente è stata riprogettata l'interfaccia grafica del tool rendendola più modulare. Essa può essere pensata come un grande *contenitore* in cui sono inseriti vari *moduli* indipendenti, uno per ogni fase dell'integrazione: il contenitore, implementato in una classe Java, gestisce l'attivazione dei moduli (quindi delle fasi) su richiesta del progettista e ogni modulo, implementato ciascuno in una classe Java diversa, fornisce le funzionalità necessarie al completamento della fase relativa. Tutti i moduli condividono la stessa memoria in cui si trovano le informazioni sugli schemi da integrare e sullo schema globale che si sta creando. In Figura 4.3 si vede il pannello relativo alla fase preliminare dell'integrazione: l'acquisizione degli schemi ODL_{I^3}.
Per quanto riguarda i due moduli SIM_1A e SIM_1B si è deciso di *fonderli* in un unico modulo che implementa le stesse funzionalità: estrazione di relazioni intra-schema, validazione ed inferenza di nuove relazioni. È stato così creato il modulo SIM.
Infine, il parser ODL_{I^3} è stato aggiornato implementando la nuova sintassi del linguaggio di descrizione degli schemi.

Con riferimento alla Figura 4.5 nelle sezioni successive verranno descritti l'architettura di SI-Designer e i moduli che implementano le fasi di integrazione, descrivendo le principali classi Java implementate.

4.2.1 L'architettura di SI-Designer

In questa sezione verranno dapprima descritte le classi Java che memorizzano le informazioni sulle classi globali e sulle relative mapping-table; poi si passerà

alla descrizione di come SI-Designer gestisce le informazioni sullo schema globale; infine, verrà descritto come è stata implementata la struttura della interfaccia grafica.

Le strutture dati delle classi globali e delle mapping-table

Le informazioni sulle classi globali e le mapping-table verranno utilizzate dal Query Manager, che pertanto deve poter disporre di una struttura dati efficiente. Questa struttura è illustrata in figura 4.4. Ogni oggetto della classe **MappingTable**

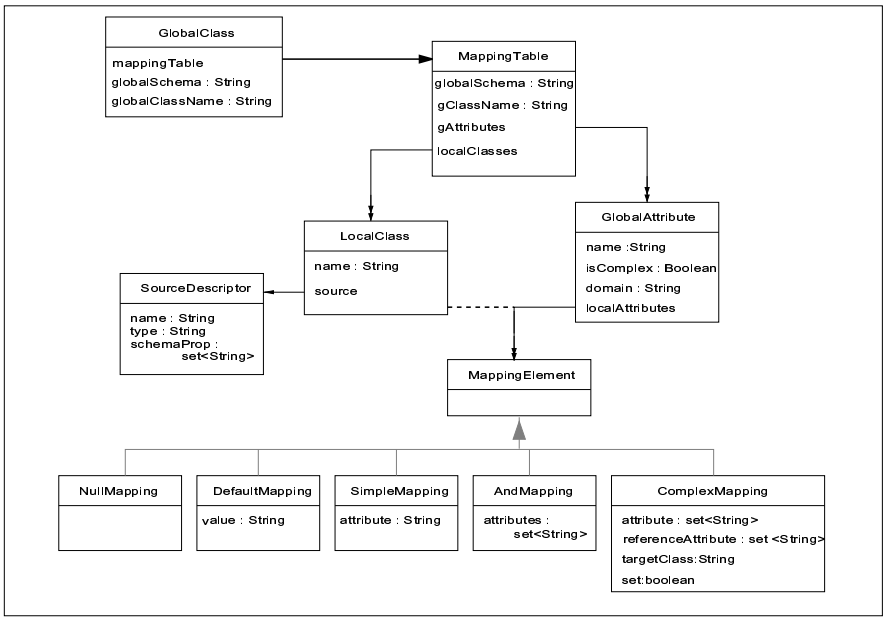


Figura 4.4: Modello a oggetti delle classi globali e delle mapping-table

contiene le informazioni su tutte le regole di mapping di una classe globale, nei seguenti attributi:

- *globalSchema*: stringa che contiene il nome dello schema globale;
- *gClassName*: nome della classe globale che l'oggetto rappresenta;
- *gAttributes*: elenco di oggetti **GlobalAttribute** (nella rappresentazione tabellare sono i titoli delle colonne);
- *localClasses*: elenco di oggetti **LocalClass** (nella rappresentazione tabellare sono i titoli delle righe).

Gli oggetti **LocalClass** contengono i dati sulle classi locali utili al Query Manager, e cioè il nome e la sorgente, a sua volta descritta dalla classe **SourceDescriptor**, contenente nome e tipo della sorgente, nonché il set di tutti gli attributi di tutte le classi dello schema locale. Gli oggetti **GlobalAttribute** contengono i dati sugli attributi globali, quali nome e dominio: l'attributo booleano *isComplex* dice se il dominio di questo attributo è un tipo-valore o un tipo-classe (in quest'ultimo caso *domain* contiene il nome di una classe globale).
I dati sui mapping sono contenuti negli oggetti **MappingElement**, raggiungibili attraverso l'attributo *localAttributes* che ne contiene l'insieme relativo all'attributo globale di cui fa parte. I diversi mapping di un attributo globale si distinguono in base a un valore chiave dato dal nome della classe locale cui sono indirizzati. Per questo motivo, pur non essendoci connessione fra **LocalClass** e **MappingElement**, nello schema compare una linea tratteggiata. Ogni oggetto **MappingElement** può essere:

- **NullMapping** se non c'è mapping fra attributo globale e classe locale;
- **DefaultMapping** se l'attributo globale assume, nella classe locale, un valore di default, contenuto in *value*;
- **SimpleMapping** se il mapping è semplice: in tal caso è sufficiente conoscere il nome dell'attributo locale;
- **AndMapping** se il mapping riguarda più attributi della classe locale in questione: i loro nomi sono contenuti nel set di stringhe *attributes*;
- **ComplexMapping** se il mapping avviene con un attributo locale che realizza aggregazione: notare che se la sorgente è relazionale, l'aggregazione si manifesta mediante una foreign key, che può essere composta da una lista di attributi locali.
  - *set* è un attributo booleano che indica se l'aggregazione è data da un unico attributo locale (è il caso di una sorgente a oggetti o di una foreign key semplice) o da una lista di attributi (quando la foreign key è composta);
  - *attribute* contiene l'insieme degli attributi locali (al limite uno solo) coinvolti nell'aggregazione;
  - *referenceAttribute* contiene lo stesso insieme di attributi, così come si chiamano nella classe mappata. Nel caso di sorgente a oggetti questo campo è vuoto;
  - *targetClass* è il nome della classe locale target nella gerarchia di aggregazione.

Gestione delle informazioni sullo schema globale

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

- un oggetto **Schema** che è composto da tutti gli oggetti istanza di classi Java appartenenti alla struttura dati utilizzata dal parser ODL_{I^3}e gli oggetti istanza istanza delle classi che rappresentano le informazioni sullo schema globale (descritte nel paragrafo precedente);

- un insieme di altre proprietà che servono a SI-Designer per mantenere traccia delle operazioni di integrazione.

Vediamo ora la descrizione dei metodi principali implementati nelle classi **GlobalSchema** e **GlobalSchemaProxy**, ma prima facciamo una precisazione. Un oggetto **GlobalSchemaProxy** legge e aggiorna lo stato dello schema globale in un oggetto CORBA **GlobalSchema** attraverso l'invocazione di opportuni metodi (*getStatus()* e *setStatus()*). Considerando che i parametri passati ai metodi invocati di un oggetto CORBA (l'oggetto **GlobalSchema** in questo caso) possono essere reference ad oggetti CORBA oppure tipi definiti utilizzando il linguaggio IDL, si pone il problema della *trasmissione* dell'oggetto Java **GSStatus**, stato dello schema globale: la soluzione è adottata è consistita nella serializzazione [33] preliminare dell'oggetto in questione ,riconducendo la trasmissione di un oggetto Javaa alla trasmissione di una stringa.

Classe CORBA GlobalSchema

METODI

- *getStatus()*: restituisce la serializzazione di un oggetto **GSStatus**. Questo metodo viene usato dal **GlobalSchemaProxy** per aggiornarsi con lo stato dello schema globale dell'oggetto remoto.
- *setStatus()*: ha come unico parametro una stringa che deve essere la serializzazione di un oggetto **GSStatus**. Analogo a *getStatus()*, imposta lo stato dello schema globale a quello passato come parametro, sovrascrivendo quello precedente. Questo metodo viene usato dal **GlobalSchemaProxy** per aggiornare lo stato dello schema globale nell'oggetto CORBA, ogni volta che il progettista lo ritiene opportuno.
- *getName()*: restituisce il nome dello schema globale. Il nome è memorizzato in una proprietà di **GSStatus**, memorizzato in una proprietà privata.
- *setName()*: imposta il nome dello schema globale a quello passato come parametro.

Classe GlobalSchemaProxy

METODI

- *getStatus()*: restituisce la serializzazione di un oggetto **GSStatus**. Questo metodo viene invocato dal modulo principale di SI-Designer per ottenere lo stato dello schema globale e memorizzarlo in un file.
- *setStatus()*: ha come unico parametro una stringa che deve essere la serializzazione di un oggetto **GSStatus**. Analogo a *getStatus()*, imposta lo stato dello schema globale a quello passato come parametro, sovrascrivendo quello precedente. Il metodo è usato dalla classe principale dell'applicazione SI-Designer quando il progettista lancia in esecuzione il tool e vuole lavorare su uno schema globale il cui stato era stato salvato precedemente in un file.
- *getName()*: restituisce il nome dello schema globale.
- *setName()*: imposta il nome dello schema globale a quello passato come parametro.
- *getGlobalClasses()*: ritorna una collezione indicizzata di oggetti **GlobalClass**, che sono le classi globali attualmente presenti nello schema, dove le chiavi dell'indice sono i nomi delle classi globali. Il metodo viene invocato dal modulo TUNIM e dal Query Manager per ottenere l'insieme di classi globali.
- *getLocalSchemata()*: questo metodo ritorna un oggetto **Schema** che contiene tutte le informazioni degli schemi locali più quelle dello schema globale. Viene invocato dai vari moduli di SI-Designer.

- *setLocalSchemata()*: il metodo duale di *getLocalSchemata()*. Aggiorna lo stato del **GlobalSchemaProxy** con le informazioni contenute nell'oggetto **Schema** passato come parametro. Viene invocato dal modulo **SAM** di SI-Designer per aggiornare lo stato dopo che è stata acquisita una nuova sorgente.
- *getWrapper()*: come parametro riceve un oggetto **Source** del package **odli3** e ritorna un object-reference all'oggetto CORBA **Wrapper** che gestisce quella sorgente. Anche se nell'object model di Figura 4.2 non compaiono, in realtà nella classe **Source** sono definite tre proprietà per memorizzare le informazioni che permettono di accedere al wrapper che gestisce la sorgente, vale a dire: nome del wrapper, indirizzo della macchina su cui il wrapper *gira* e la porta di accesso al wrapper.
- *loadStatus()*: invocato dal modulo principale di SI-Designer per aggiornare lo stato del **GlobalSchemaProxy** con quello memorizzato nell'oggetto CORBA remoto **GlobalSchema** attualmente collegato.
- *saveStatus()*: invocato dal modulo principale di SI-Designer per aggiornare lo stato del **GlobalSchema** con quello memorizzato in **GlobalSchemaProxy**. L'invocazione avviene su richiesta del progettista, che schiaccia un pulsante nella interfaccia grafica.
- *connectGlobalSchema()*: imposta la connessione del **GlobalSchemaProxy** con un nuovo oggetto CORBA **GlobalSchema** il cui object reference viene passato come parametro. Prima di connettersi salva lo stato dello schema globale nel vecchio **GlobalSchema** (se ce n'era uno collegato) e aggiorna lo stato del **GlobalSchemaProxy** invocando il metodo *getStatus()* del nuovo **GlobalSchema**. Il metodo è invocato dal modulo principale di SI-Designer su richiesta del progettista che schiaccia un pulsante nella interfaccia grafica.
- *disconnectGlobalSchema()*: interrompe la connessione con il **GlobalSchema** attuale, aggiornando lo stato dell'oggetto CORBA se il valore del parametro passato vale **true**.

L'interfaccia grafica

L'interfaccia grafica di SI-Designer è costituita da un pannello principale in cui sono inseriti una sequenza di pannelli: uno per ogni fase dell'integrazione. Ogni pannello è selezionabile dal progettista attraverso la selezione di una *etichetta* che lo identifica.
In Figura 4.3 è visibile il pannello della fase preliminare all'integrazione: l'acquisizione degli schemi delle sorgenti. Si notano i comandi sempre accessibili in

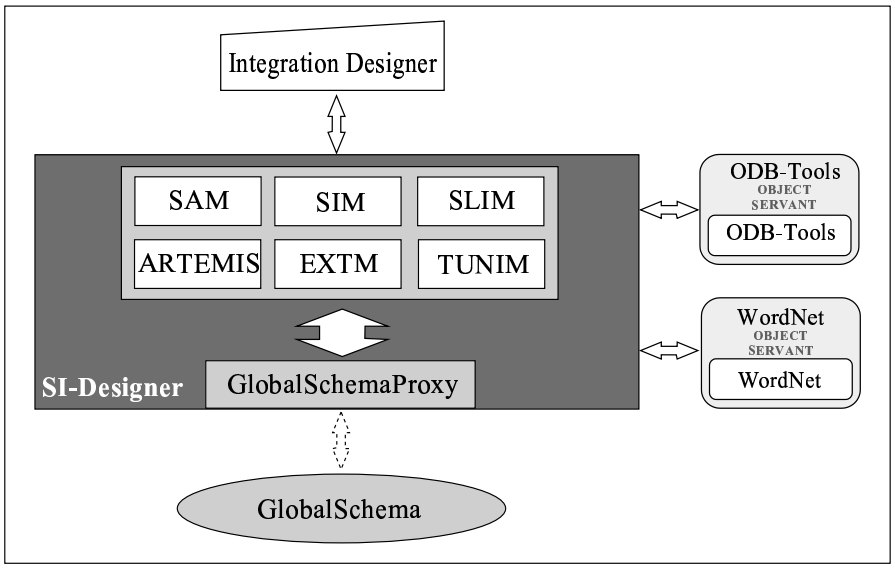


Figura 4.5: L'architettura di SI-Designer

qualsiasi fase dell'integrazione ci si trovi, i quali sono implementati nella classe Java principale di SI-Designer che: lancia in esecuzione il tool, crea il pannello principale e vi aggiunge i pannelli delle varie fasi di integrazione. In particolare, si notano in alto a sinistra le etichette dei vari pannelli delle fasi: cliccando su una etichetta il progettista può selezionare il modulo relativo alla fase che vuole eseguire. In basso invece sono identificabili i pulsanti:

- SAVE: pre salvare lo stato dello schema globale in un file;
- CONNECT: per creare un oggetto CORBA **GlobalSchema** e salvarvi lo stato dello schema globale attuale;
- QUIT: per terminare l'esecuzione di SI-Designer.

Dal punto di vista implementativo, la modularità di SI-Designer è stata ottenuta creando una classe Java che gestisce il pannello principale. I pannelli delle fasi di integrazione sono gestiti da dei moduli costituiti da classi Java, una per ogni modulo, figlie di una medesima classe Java astratta, **SIDPhase**.

Classe astratta SIDPhase

METODI

- *getProxy()*: ritorna l'oggetto **GlobalSchemaProxy** con cui **SIDPhase** è stato inizializzato. Il metodo viene invocato da tutti i moduli di

SI-Designer quando devono ottenere le informazioni sullo schema globale e sugli schemi locali.

- *saveStatus()*: invocato dal modulo principale di SI-Designer ogni volta che il progettista sceglie di passare da una fase all'altra. L'oggetto di cui viene invocato il metodo è quello relativo alla fase attualmente visualizzata e che il progettista vuole terminare per passare ad un'altra. Invocando questo metodo, SI-Designer si aspetta che il modulo della fase attuale termini le operazioni che stava eseguendo e salvi nel proxy tutte le modifiche che sono state fatte. Il metodo genera una eccezione Java se la fase attuale non può terminare: in questo caso, viene visualizzato un messaggio che indica l'impossibilità a passare ad altra fase.
- *update()*: il metodo viene invocato dal modulo principale di SI-Designer ogni volta che il progettista sceglie di passare da una fase all'altra. L'oggetto di cui viene invocato il metodo è quello relativo alla fase che il progettista vuole attivare. Invocando questo metodo, SI-Designer si aspetta che venga aggiornata l'interfaccia grafica con i dati attualmente presenti nel proxy. Il metodo genera una eccezione Java se la fase nuova non può essere inizializzata. Ciò può accadere perché non sono disponibili tutte le informazioni necessarie per una esecuzione.
- *SIMValidation()*: invocato dai moduli di SI-Designer ogni volta che si rende necessario eseguire una validazione di un insieme di relazioni del Thesaurus Comune. Il parametro passato è una collezione di oggetti **ThesRelation** che rappresentano le relazioni da validare. Il metodo non ritorna alcun valore ma imposta una proprietà degli oggetti passati che indica se quella relazione è validata o meno: ciò viene fatto invocando il metodo *setValidated()* nella classe **ThesRelation**, passando come parametro **true** o **false** a seconda dell'esito.

I moduli implementati in SI-Designer che realizzano le fasi dell'integrazione degli schemi sono i seguenti:

- *SAM, Sources Acquisition Module*: è il modulo di acquisizione delle sorgenti e acquisisce, tramite il parser ODL_{I^3} gli schemi delle sorgenti da integrare. Nella Figura 4.3 è presente la prima schermata di SI-Designer con l'interfaccia grafica di SAM. Una descrizione approfondita di questo modulo verrà presentata nella sezione 4.3;

- *SIM, Sources Integrator Module*: si occupa della generazione del Thesaurus Comune. In particolare svolge i seguenti compiti:

  - estrae le relazioni intra-schema analizzando gli schemi di ogni singola sorgente,
  - valida, con l'ausilio di ODB-Tools , tutte le altre relazioni che verranno aggiunte al Thesaurus Comune (relazioni inter-schema e aggiunte direttamente dal progettista),
  - inferisce nuove relazioni attraverso l'uso di ODB-Tools ;

  A differenza degli altri moduli questo non ha una propria interfaccia grafica ma implementa solamente metodi che forniscono i servizi citati;

- *SLIM, Sources Lessical Integrator Module*: appoggiandosi al sistema lessicale WordNet, partecipa alla generazione del Thesaurus Comune occupandosi dell'estrazione di relazioni terminologiche inter-schema, sulla base dei significati dei nomi di classi e attributi che compaiono negli schemi delle sorgenti;

- *ARTEM*: calcola l'affinità tra le classi che compongono le diverse sorgenti da integrare. In base a queste affinità, attraverso un algoritmo di clustering, il tool crea dei gruppi di classi *affini*, i *cluster*, che andranno a costituire poi le classi dello schema globale;

- *TUNIM, TUNIng of mapping-table Module*: crea le classi globali a partire dai cluster e dal Thesaurus Comune creato. Una descrizione approfondita di questo modulo verrà presentata nella sezione 4.4.

4.3 SAM: Sources Acquisition Module

Il processo di integrazione degli schemi è preceduto dalla fase di acquisizione degli stessi. Questo modulo si occupa proprio di questo e il suo compito è popolare le strutture dati atte a memorizzare le informazioni sugli schemi locali: nella fattispecie le proprietà della classe **GlobalSchemaProxy**. Tale operazione viene fatta attraverso il parser ODL$_{I^3}$.

4.3.1 L'acquisizione di uno schema ODL$_{I^3}$

Facendo riferimento all'interfaccia grafica di questo modulo in Figura 4.3, di seguito verrà descritto come avviene l'acquisizione di uno schema ODL$_{I^3}$ da un wrapper.

Per prima cosa occorre indicare al tool le informazioni per accedere all'oggetto wrapper, vale a dire: il nome dell'oggetto wrapper, l'indirizzo internet della

macchina sulla quale *gira* il wrapper e la porta di accesso. Tutte queste informazioni possono essere inserite semplicemente digitandole negli appositi campi posti in basso a destra. Una volta fornite tutte le informazioni si avvia la procedura di acquisizione dello schema premendo il pulsante ADD posto in basso a destra.
Il modulo SAM esegue l'acquisizione secondo i seguenti passi:

1. *controllo dell'esistenza del wrapper indicato*: il modulo richiede all'ORB (vedere appendice C) un object-reference per il wrapper CORBA e, nel caso l'object reference venga fornito, esegue un test per controllare se il wrapper è attivo. Il test consiste nell'invocare un metodo del wrapper: in caso di fallimento viene visualizzato un messaggio di errore;

2. *acquisizione dello schema ODL$_{I^3}$*: viene eseguito il *parsing* dello schema ODL$_{I^3}$ fornito dal wrapper sottoforma di stringa. Nel caso in cui si siano verificati errori sintattici o semantici viene visualizzato un messaggio di errore, e l'acquisizione è come se non fosse iniziata;

3. *aggiunta della nuova acquisizione alle precedenti*: le informazioni sul nuovo schema vengono aggiunte a quelle degli eventuali schemi acquisiti in precedenza. Se il modulo si accorge che era stata acquisito lo schema di una sorgente avente lo stesso nome di quella descritta nello schema appena acquisito allora viene richiesto al progettista di cambiare nome alla sorgente prima di mettere assieme le informazioni degli schemi.

Dopo l'acquisizione di ogni schema, il modulo provvede all'aggiornamento dell'interfaccia grafica che comprende:

- l'elenco delle sorgenti, delle classi e degli attributi organizzati in un albero in cui gli attributi sono le foglie (sulla sinistra della Figura 4.3);

- la tabella dell'elenco delle sorgenti acquisite con le informazioni sul relativo wrapper che le gestisce.

4.3.2 L'oggetto Wrapper

Per completezza si riporta la descrizione dell'oggetto CORBA Wrapper, con il quale interagisce il modulo SAM.
L'oggetto deve fornire le seguenti funzionalità:

- fornire gli schemi ODL$_{I^3}$ della sorgente che gestisce,

- eseguire le query OQL$_{I^3}$ dal Query Manager

- ritornare al Query Manager i risultati delle query.

Vediamo quali sono i metodi implementati nell'interfaccia dell'oggetto.

Classe CORBA Wrapper

METODI

- *getType()*: restituisce una stringa che descrive il tipo di sorgente gestita dal wrapper. Viene invocata da SI-Designer per eseguire un test sulla attivazione o meno del wrapper in questione (il tipo della sorgente è in ogni caso indicato nella descrizione ODL_{I^3} dello schema);
- *getDescription()*: ritorna una stringa con lo schema ODL_{I^3} della sorgente. Il metodo è invocato da SI-Designer per ottenere lo schema da acquisire.
- *runQuery()*: ritorna il risultato della esecuzione di una query OQL_{I^3}, passata come parametro, sottoforma di un oggetto CORBA **MomisResultSet**. Il metodo è invocato dal Query Manager quando richiede l'esecuzione di una query.
- *getSourceName()*: ritorna il nome della sorgente gestita dal wrapper.

4.4 TUNIM

Al calcolo dei cluste, eseguito attraverso il modulo ARTEM, segue la fase di generazione delle classi globali: e proprio di questo si occupa il modulo TUNIM.

Il procedimento seguito dal modulo per creare le classi globali ricalca quello descritto nel capitolo 3:

- unione degli attributi;
- fusione degli attributi simili.

Vediamo più in dettaglio qual'è il procedimento seguito dal modulo per eseguire le due fasi.

4.4.1 Unione degli attributi

Dato un cluster, per ogni attributo locale a_l, della classe locale c_l, presente nel cluster Cl, il modulo crea un attributo globale a_g che viene aggiunto alla mapping-table: il nome dell'attributo è lo stesso dell'attributo locale a meno che non esista già un altro attributo globale con lo stesso nome, nel qual caso il modulo aggiunge un numero progressivo o un carattere aggiuntivo che distingua il nome da tutti gli altri. (Il motivo per cui non viene chiesto al progettista di inserire il nome

è che così facendo la procedura non sarebbe più automatica e probabilmente ci sarebbero continuamente dei conflitti di quel tipo.) Nella mapping-table, in corrispondenza della casella MT[c_l,a_g] verrà aggiunta una mapping-rule del tipo a **corrispondenza semplice**, poiché l'attributo globale aggiunto, per la classe locale c_l, assume il solo valore dell'attributo a_l: tutti gli altri mapping sono di tipo **null**.

4.4.2 Fusione degli attributi simili

La fusione degli attributi deve essere fatta attentamente per evitare di ottenere dei mapping impropri.
Riorganizzando le relazioni del Thesaurus Comune in un grafo e analizzandone la struttura, il modulo è in grado di individuare delle fusioni *sicure* e delle fusioni *possibili*: le prime vengono attuate dal modulo in modo automatico senza interazione con il progettista a meno che non vi siano conflitti tra i domini; le seconde invece richiedono l'interazione col progettista, il quale dovrà confermarle oppure scegliere un tipo di fusione da diverse alternative.
Gli attributi globali che vengono considerati per la ricerca delle fusioni sono quelli che sono stati mappati su un solo attributo locale di una classe locale e nessun altro mapping. Tentare di fondere assieme automaticamente attributi globali che sono mappati su più attributi locali sarebbe abbastanza complicato perché occorrerebbe tenere conto dei vari tipi di mapping impostati. Questi casi non vengono considerati da TUNIM, tuttavia il progettista può intervenire manualmente per eliminare, aggiungere o modificare attributi globali e mapping-table.

Le fusioni sicure I tipi di fusione *sicura* sono quelle illustrate in Figura 4.6:

- *attributi di classi locali diverse legati da relazioni di specializzazione*: ad esempio, i due attributi `A.att1` e `B.att2` appartengono a classi diverse e sono legati da relazioni di specializzazione con l'attributo `C.att3`. In base ai criteri enunciati nel capitolo 3, i tre attributi possono essere fusi in uno solo. Dalla mapping-table potranno così essere tolte due colonne (i due attributi `A.att1` e `B.att2`);

- *attributi appartenenti a classi locali diverse e legati da sole relazioni di sinonimia*: è il caso degli attributi `D.att3` e `C.att2` che soddisfano i criteri enunciati nel capitolo 3, dunque possono essere fusi in modo automatico, per cui uno dei due attributi viene eliminato dalla mapping-table.

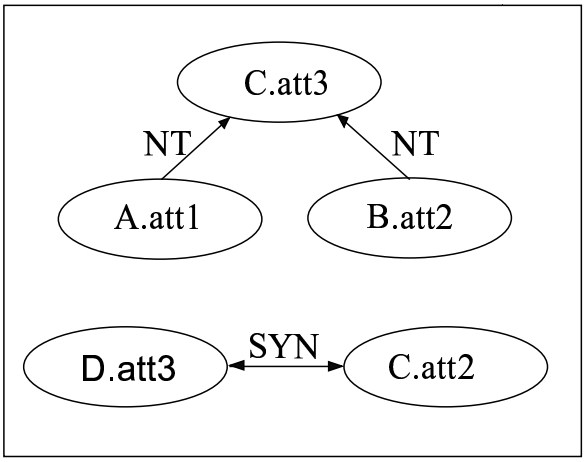


Figura 4.6: Le fusioni sicure

Le fusioni possibili I casi che si possono presentare sono più numerosi rispetto a quelli delle fusioni sicure. Come riferimento si consideri il grafo di Figura 4.7:

- *attributi partecipanti a relazioni di diverso tipo*: per esempio, si supponga che i due attributi b e c appartengano alla medesima classe, diversa da quella di tutti gli altri attributi. L'analisi delle relazioni validate di specializzazione porta all'individuazione di una possibile fusione tra gli attributi b, c ed a: b e c verrebbero fusi in a. Tuttavia, l'attributo c è legato da una relazione di sinonimia con d. Dunque si presentano due strade: o si esegue la fusione con le relazioni di specializzazione oppure con quella di sinonimia tra l'attributo c e d.
- *attributi appartenenti ad una medesima classe locale e figli di uno stesso attributo padre*: ad esempio, nell'ipotesi che i due attributi g ed h ap-

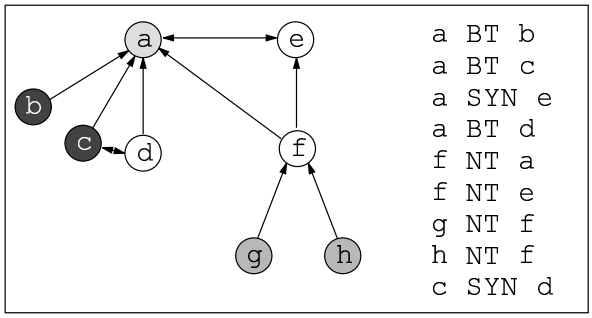


Figura 4.7: Le fusioni possibili

partengano alla stessa classe locale, si presentano due alternative: la fusione con corrispondenza in *and* oppure con corrispondenza in *union*. Non è possibile stabilire, solo sulla base delle relazioni del Thesaurus, quale dei due tipi di fusione sia il più appropriato.

- *attributi legati da relazioni non validate*: a questa categoria appartengono i casi presentati nella sezione 3.1.2, un esempio dei quali è illustrato in figura 3.4.

TUNIM quando si accorge che un attributo può essere fuso in modi diversi, propone al progettista alcune alternative chiedendogli di scegliere quella che ritiene più opportuna.

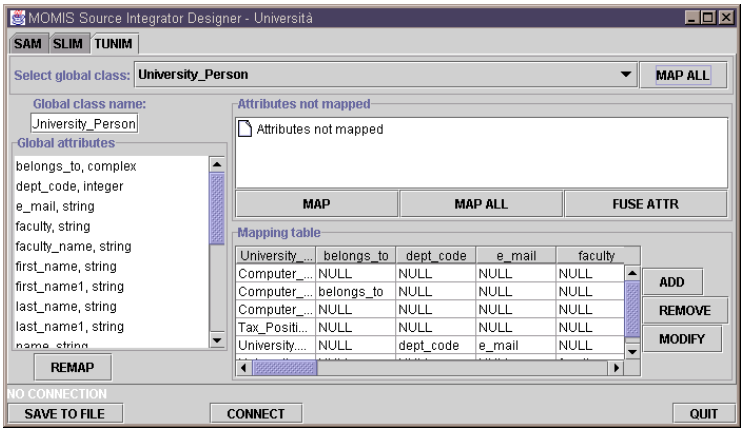


Figura 4.8: L'interfaccia grafica del modulo TUNIM

4.4.3 Funzionalità dell'interfaccia

In questa sezione viene descritta l'interfaccia del modulo, visibile in Figura 4.8.

Il pannello è suddiviso in tre parti:

- sulla sinistra troviamo gli attributi globali relativi alla classe globale attualmente selezionata;
- sulla destra, in alto, sono visualizzati gli attributi locali non mappati, vale a dire quegli attributi per i quali non è stata generata una mapping rule;
- sulla destra in basso si trova la mapping-table: ogni riga corrisponde ad una classe locale appartenente al cluster mentre ogni colonna corrisponde ad un attributo globale.

In cima, appena sotto il titolo della finestra, è posto il selettore della classe globale: scegliendo il nome della classe globale che si vuole creare, automaticamente il modulo aggiorna tutti gli altri elementi del pannello (elenco attributi globali, elenco attributi locali non mappati e mapping table).

Vediamo ora come si usa l'interfaccia per eseguire le principali operazioni inerenti la creazione o la modifica di una classe globale:

- *cambio nome della classe globale*: è sufficiente selezionare il nome attuale della classe dall'apposito selettore posto sul lato superiore della finestra e successivamente modificare il nome stesso posto nella casella in alto a sinistra, posto sotto la dicitura "Global class name";

- *unione degli attributi per tutte le classi globali*: premendo il pulsante "MAP ALL" posto sulla destra del selettore della classe globale, si avvia la procedura di creazione automatica della mapping-table iniziale, quella che si ottiene dalla unione di tutti gli attributi locali (spiegata nella sezione 4.4.1), per tutte le classi globali;

- *mapping di un attributo locale non mappato*: occorre selezionare l'attributo dall'elenco di attributi non mappati e successivamente premere il pulsante MAP posto al centro della finestra. In modo automatico, TUNIM crea il grafo di relazioni (vedere sezione 3.1) per quell'attributo e controlla se è possibile eseguire una fusione con altri attributi: individua se è possibile attuare una fusione sicura oppure propone al progettista una eventuale fusione possibile individuata.

- *fusione degli attributi fondibili*: questa funzionalità viene attuata da TUNIM attraverso la pressione del pulsante FUSE ATTR. Quello che il modulo fa, è considerare tutti gli attributi globali che sono stati mappati su un solo attributo locale di una classe locale, tentando di individuare delle possibili fusioni (anche sicure) in base all'analisi delle relazioni del Thesaurus. In questo modo, se alcune fusioni vengono eseguite, si riduce il numero degli attributi globali nella mapping-table.

- *eliminazione di un attributo globale*: è sufficiente selezionare la colonna nella mapping-table relativa all'attributo che si vuole eliminare e premere il pulsante REMOVE in basso a destra. Se si conferma l'eliminazione dell'attributo allora viene eliminata dalla mapping-table la colonna relativa così come tutti i mapping: gli attributi locali che erano mappati nell'attributo globale eliminato e non sono mappati in nessun altro attributo globale vengono aggiunti all'elenco degli attributi non mappati;

- *aggiunta di un attributo globale*: occorre premere il pulsante ADD in basso a destra. Il modulo chiede così al progettista di inserire il nome del nuovo attributo e di selezionarne il dominio che può essere: *string, integer, floating, array, boolean* oppure il nome di una classe globale nel caso in cui l'attributo sia complesso.

- *eliminazione di un mapping*: è sufficiente selezionare la casella della mapping-table contenente il mapping da eliminare e premere il pulsante REMOVE. Gli attributi locali che appartenevano al mapping eliminato, se non sono mappati in altri attributi globali, sono aggiunti all'insieme degli attributi non mappati.

I pulsanti che non sono ancora stati citati sono:

- REMAP: su conferma del progettista, elimina la mapping table attuale ed esegue l'union di tutti gli attributi;

- MAP ALL (il pulsante posto al centro): esegue il mapping di tutti gli attributi locali non ancora mappati tentando di eseguire le fusioni;

- MODIFY: se è selezionata una colonna della mapping-table allora viene richiesto al progettista di modificare il nome dell'attributo globale.

4.5 Gli strumenti hs/sw utilizzati

Il tool è stato sviluppato in linguaggio Java, versione 1.2.2, su una workstation Sun Ultra10 con sistema operativo Solaris e su personal computer equipaggiato con sistema operativo Windows 95.
Il software inoltre è stato testato su sistema operativo Linux, sempre in ambiente Java versione 1.2.2, senza dare problemi.

Conclusioni

La continua crescita del numero di sorgenti di informazioni ha creato la necessità di avere a disposizione strumenti che permettano di districarsi nel disordinato mare di informazioni quale è ad esempio Internet. Questa tesi ha presentato il progetto di un sistema che tenta di risolvere il problema: si tratta di MOMIS, un sistema intelligente, integratore di sorgenti di informazioni eterogenee e distribuite. Questo sistema si differenzia da altri progetti simili per l'automatizzazione della fase di integrazione degli schemi delle sorgenti: partendo da questi ultimi è in grado di ottenere in modo semi-automatico una vista globale sulle sorgenti stesse. È così possibile interrogare una moltitudine di sorgenti formulando delle query sullo schema della vista creata, lasciando al sistema il compito di formulare le interrogazioni per ogni singola sorgente nel linguaggio specifico, eseguendo anche una ottimizzazione dell'accesso. Il tool grafico presentato in questa tesi è l'interfaccia grafica che permette la costruzione dello schema globale in MOMIS e costituisce un valido aiuto per il progettista. Grazie alla semplicità dell'interfaccia grafica e alla flessibilità di utilizzo del tool il progettista può costruire facilmente lo schema globale: ciò è reso possibile anche dall'utilizzo di tecniche di Intelligenza Artificiale quali l'algoritmo di sussunzione e di un sistema lessicale, WordNet. Un grande sforzo è stato compiuto per rendere il software altamente modulare in modo da rendere semplici ulteriori modifiche future quali l'introduzione di nuove fasi nel procedimento di integrazione o di modifica delle esistenti. L'aver scelto una architettura standard, quale è CORBA, per l'interazione con il mondo esterno, costituisce inoltre una buona base di partenza per rendere il sistema MOMIS fruibile anche da parte di altri sistemi di gestione di informazioni. In questo senso si potrebbero riprogettare le strutture dati utilizzate all'interno degli oggetti CORBA, ridefinendole come oggetti CORBA tramite il linguaggio IDL.

Gli sviluppi futuri del sistema MOMIS. Il *collo di bottiglia* nella creazione dello schema globale è costituito dalla creazione del Thesaurus Comune perché il sistema deve capire il significato dei termini usati per rappresentare i dati immagazzinati nelle sorgenti. Un possibile miglioramento del software prodotto consiste allora in uno sfruttamento maggiore del sistema lessicale, eseguendo ancora ricerche sulle relazioni ottenute per i singoli termini, ma confrontando tra loro gli insiemi di relazioni ottenuti per ciascun termine in modo da eliminare tutte quelle relazioni che sicuramente non sono pertinenti al contesto in cui i medesimi termini sono usati. Ulteriori sviluppi possono essere poi apportati implementando

algoritmi più complessi per la ricerca e la eventuale fusione automatica di attributi.
Anche le funzionalità del software sono suscettibili di miglioramenti. Allo stato attuale infatti non è possibile eliminare una sorgente già integrata in uno schema globale: la si può solo aggiungere. Ne tantomeno è possibile integrare nello schema globale eventuali modifiche apportate agli schemi delle sorgenti già acquisite. Ciò costituisce senz'altro una limitazione nell'utilizzo del sistema in quanto le sorgenti di informazioni accessibili via Internet sono suscettibili di cambiamenti frequenti.
Per quanto riguarda il Query Manager infine occorre studiare e realizzare la parte del software che integra le informazioni ricevute dalle singole sorgenti in risposta alle interrogazioni.

Le difficoltà incontrate nello sviluppo del software hanno riguardato soprattutto la ridefinizione di proprietà e metodi di alcune delle classi Java che memorizzano le informazioni sugli schemi. Infatti, i due componenti principali del sistema MOMIS, Global schema Builder e Query Manager, sono stati sviluppati separatamente. Ciò ha comportato la creazione di differenti strutture dati per rappresentare le medesime informazioni relative allo schema globale. Si è dovuto quindi ricorrere ad una manipolazione delle classi per ottenere una unica struttura dati in cui memorizzare sia le informazioni relative agli schemi locali che quelle relative allo schema globale.

Il lavoro presentato in questa tesi è stato accettato al SEBD 2000, un importante convegno italiano sullo stato dell'arte della ricerca nel campo dei DataBase e al VLDB, la più importante manifestazione internazionale nel campo delle basi di dati.

La progettazione e realizzazione del tool presentato in questa tesi mi ha permesso di imparare la programmazione in linguaggio Java ma soprattutto di entrare nell'ottica della programmazione ad oggetti. Importante è stato anche l'aver partecipato alla realizzazione di un progetto complesso quale è il progetto MOMIS, lavorando in un team di persone.

Appendice A

Glossario I^3

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

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

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

A.1 Architettura

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

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

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

A.2 Servizi

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

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

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

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

A.3 Risorse

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

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

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

A.4 Ontologia

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

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

Appendice B

Il linguaggio ODL$_{I^3}$

Si propone la descrizione BNF del linguaggio ODL$_{I^3}$. Sono stati inclusi unicamente gli elementi sintattici che differiscono dalla grammatica originale ODL dello standard ODMG-93.

⟨interface_dcl⟩	: : =	⟨interface_header⟩ {[⟨ interface_body⟩]}; [**union** ⟨identifier⟩ { ⟨interface_body⟩ }**;**]
⟨interface_header⟩	: : =	**interface** ⟨identifier⟩[⟨inheritance_spec⟩] [⟨type_property_list⟩]
⟨inheritance_spec⟩	: : =	: ⟨scoped_name⟩ [,⟨inheritance_spec⟩]

Definizione di modello di schema locale: il wrapper deve potere indicare il tipo e il nome della sorgente per ogni modello.

⟨type_property_list⟩ : : = **(** [⟨source_spec⟩]
[⟨extent_spec⟩]
[⟨key_spec⟩] [⟨f_key_spec⟩] [⟨c_key_spec⟩] **)**
⟨source_spec⟩ : : = **source** ⟨source_type⟩
⟨source_name⟩
⟨source_type⟩ : : = **relational** | **nfrelational**
| **object** | **file**
| **semistructured**
⟨source_name⟩ : : = ⟨identifier⟩
⟨extent_spec⟩ : : = **extent** ⟨extent_list⟩
⟨extent_list⟩ : : = ⟨string⟩ | ⟨string⟩,⟨extent_list⟩
⟨key_spec⟩ : : = **key**[**s**] ⟨key_list⟩
⟨f_key_spec⟩ : : = **foreign_key** (⟨f_key_list⟩)
references ⟨key_list
⟩ [⟨f_key_spec⟩]
⟨c_key_spec⟩ : : = **candidate_key** ⟨identifier⟩
(⟨key_list⟩**)**

Regole di definizione del mapping fra attributi della classe globale dello schema del mediatore e i corrispondenti nelle sorgenti locali.

⟨attr_dcl⟩ : : = **[readonly] attribute**
[⟨domain_type⟩]
⟨attribute_name⟩ **[*]**
[⟨fixed_array_size⟩]
[⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ : : = **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ : : = ⟨rule⟩ | ⟨rule⟩**,**⟨rule_list⟩
⟨rule⟩ : : = ⟨local_attr_name⟩ |
‘⟨identifier⟩**’**
⟨and_expression⟩ |
⟨union_expression⟩
⟨and_expression⟩ : : = **(** ⟨local_attr_name⟩ **and**
⟨and_list⟩ **)**
⟨and_list⟩ : : = ⟨local_attr_name⟩
| ⟨local_attr_name⟩ **and**
⟨and_list⟩
⟨union_expression⟩ : : = **(** ⟨local_attr_name⟩ **union**
⟨union_list⟩ **on** ⟨identifier⟩ **)**
⟨union_list⟩ : : = ⟨local_attr_name⟩

| ⟨local_attr_name⟩ **union**
⟨union_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩**.**⟨class_name⟩**.**
⟨attribute_name⟩
...

Relazioni terminologiche utilizzate per definire il Common Thesaurus.

⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** |
⟨relationship_dcl⟩**;**
⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_name⟩
⟨relationship_type⟩
⟨local_name⟩
⟨local_name⟩ ::= ⟨source_name⟩**.**
⟨local_class_name⟩
[**.**⟨local_attr_name⟩]
⟨relationship_type⟩ ::= **SYN** | **BT** | **NT** | **RT**
...

Definizione dei vincoli di integrità **OLCD** dichiarazione delle regole (utilizzando le definizioni *if then*) valide per ogni istanza di dato; specificazione delle mapping rule (specificazione delle regole *or* e *and*).

⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_spec⟩
⟨rule_spec⟩ ::= ⟨rule_pre⟩ **then** ⟨rule_post⟩ |
{ ⟨case_dcl⟩ }
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨identifier⟩ **:**
⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨case_dcl⟩ ::= **case of** ⟨identifier⟩ **:** ⟨case_list⟩
⟨case_list⟩ ::= ⟨case_spec⟩ | ⟨case_spec⟩ ⟨case_list⟩
⟨case_spec⟩ ::= ⟨identifier⟩ **:** ⟨identifier⟩ **;**

⟨rule_body_list⟩ : : = **(** ⟨rule_body_list⟩ **)** |
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
(⟨rule_body_list⟩ **)**

⟨rule_body⟩ : : = ⟨dotted_name⟩
⟨rule_const_op⟩
⟨literal_value⟩ |
⟨dotted_name⟩
⟨rule_const_op⟩
⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in**
⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩

⟨rule_const_op⟩ : : = = | ≥ | ≤ | > | <

⟨rule_cast⟩ : : = **(**⟨simple_type_spec⟩**)**

⟨dotted_name⟩ : : = ⟨identifier⟩ | ⟨identifier⟩**.**
⟨dotted_name⟩

⟨forall⟩ : : = **for all** | **forall**

Appendice C

L'architettura CORBA

CORBA [34] (*Common Object Request Broker Architecture*) è un'architettura standard, distribuita e ad oggetti sviluppata dall'Object Management Group (OMG). Dal 1989 l'obiettivo del gruppo OMG è stato la progettazione di una architettura per un *software bus* aperto, chiamato *Object Request Broker* (ORB), sul quale oggetti diversi potessero interagire via rete, indipendentemente dal sistema operativo in cui sono stati implementati. Questo standard permette a più oggetti di invocarne altri senza conoscerne l'esatta locazione o in quale linguaggio sono stati implementati.
Il linguaggio utilizzato per definire un oggetto CORBA è l'IDL (*Interface Definition Language*) mentre gli ORB comunicano attraverso il protocollo standard IIOP (*Internet InterORB Protocol*), definito sempre dall'OMG.

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

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

Come funziona

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

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

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

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

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

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

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

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

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

Il Naming Service

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

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

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

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

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

La creazione di oggetti CORBA in Java

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

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

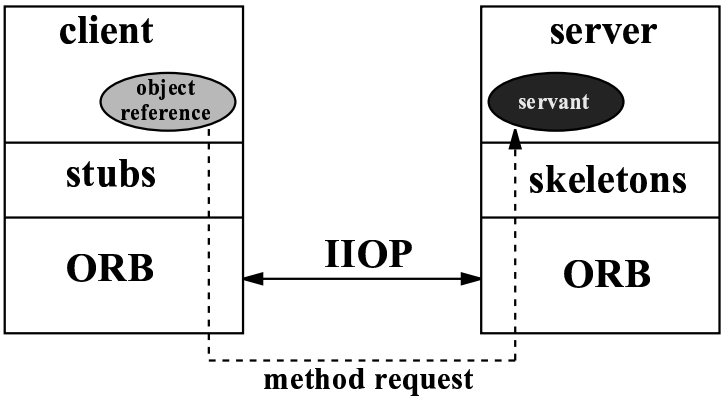


Figura C.1: La invocazione di un metodo di un oggetto CORBA remoto

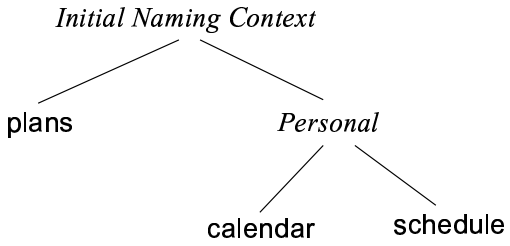


Figura C.2: Esempio di albero creato dal naming server

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

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

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

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

Appendice D

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

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

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

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

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

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

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

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

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

```
interface CS_Person
(  source object Computer_Science
```

```
interface Professor : CS_Person
 ( source object Computer_Science
```

```
  extent CS_Persons
  key name )
{ attribute string name; };
```

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

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

```
   extent Professors )
{ attribute string title;
  attribute Division belongs_to;
  attribute string rank; };
```

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

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

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

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

Bibliografia

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

[2] R. Hull and R. King et al. Arpa i^3 reference architecture, 1995. Available at http://www.isse.gmu.edu/I3_Arch/index.html.

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

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

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

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

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

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

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

[10] 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.

[11] Y.Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB Int. Conf.*, Bombay, India, September 1996.

[12] 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.

[13] 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, 1997.

[14] 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.

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

[16] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[17] B. Everitt. *Computer-Aided Database Design: the DATAID Project*. Heinemann Educational Books Ltd, Social Science Research Council, 1974.

[18] R. G. G. Cattell, editor. *The Object Database Standard: ODMG93*. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[19] R.G.G. Cattell and others., editors. *The Object Data Standard: ODMG 2.0*. Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[20] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. ODB-QOPTIMIZER: A tool for semantic query optimization in oodb. In *Int. Conference on Data Engineering - ICDE97*, 1997. http://sparc20.dsi.unimo.it.

[21] D. Beneventano, A. Corni, S. Lodi, and M. Vincini. ODB: validazione di schemi e ottimizzazione semantica on-line per basi di dati object oriented.

In *Quinto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD97, Verona*, pages 208–225, 1997.

[22] W. A. Woods and J. G. Schmolze. The KL-ONE family. In F. W. Lehman, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Pubblished as a Special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2-9.

[23] J. P. Ballerini, D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. A semantics driven query optimizer for OODBs. In A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, editors, *DL95 - Intern. Workshop on Description Logics*, volume 07.95 of *Dip. di Informatica e Sistemistica - Univ. di Roma "La Sapienza" - Rapp. Tecnico*, pages 59–62, Roma, June 1995.

[24] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[25] 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.

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

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

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

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

[30] Bates M. Subject access in online catalogs: A design model. *Journal of the American Society for Information Science*, 11:357–376, 1986.

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

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

[33] Object serialization. disponibile all'indirizzo http://java.sun.com/docs/books/tutorial/essential/io/serialization.html.

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