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

MOMIS: servizi di wrapping per sorgenti relazionali JDBC

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Stefano Coriani

Correlatore
ing. Francesco Guerra

Anno Accademico 2000 - 2001

Parole chiave:

Integrazione dati
Wrapper
ODL$_{I^3}$
Sorgenti relazionali JDBC
Metadati

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi per l'aiuto fornito durante la realizzazione della presente tesi.
Rivolgo un sincero ringraziamento all'Ing. Francesco Guerra per la costante disponibilità dimostrata e per i preziosi consigli.

Vorrei ringraziare anche la mia famiglia e i miei amici che mi hanno sostenuto durante questo lungo :-) percorso.

Indice

Elenco delle figure

Introduzione

Nell'ultimo decennio abbiamo assistito ad una diffusione su larga scala delle tecnologie legate all'informatica, in particolare a quelle legate alla trasmissione dell'informazione.Tutto ciò ha reso disponibili una grande quantità di dati accessibili, basti pensare al fenomeno *internet*. L'abbondanza di informazioni ha però reso problematico, da parte dell'utente, il discernere e l'isolare i dati significativi, fenomeno dell'*information overload* (sovraccarico di informazioni). A complicare ulteriormente lo scenario, oltre alla quantità di sorgenti dati, c'è anche la varietà di queste sorgenti; i dati cercati , infatti, si trovano spesso in più documenti collocati su diverse sorgenti frequentemente eterogenee. L'eterogeneità si manifesta sotto diversi aspetti: differenti piattaforme hardware e software sulle quali sono implementate le sorgenti (DBMS, linguaggi di interrogazione, ...), differenti modelli di dati (relazionale, ad oggetti, ...), differenti schemi di rappresentazione della struttura logica.
Per ottenere il dato cercato, quindi, l'utente dovrebbe innanzitutto conoscere le piattaforme su cui può essere memorizzato, la struttura logica di tali sorgenti, i meccanismi di interrogazione da utilizzare ed infine essere in grado di interpretare i risultati ottenuti.
Da qui la necessità di realizzare strumenti in grado di omogeneizzare, in modo automatico o semi-automatico, l'accesso alle risorse informative. Nasce quindi la problematica dell'integrazione delle sorgenti dati.

L'importanza e l'interesse intorno all'argomento non solo dal punto di vista teorico, ma anche applicativo, sono dimostrate dallo sviluppo, a livello commerciale, di sistemi quali *Sistemi di Workflow, Datawarehouse, Dataminer, ecc* che mirano ad ottenere proprio questo risultato.

Questa tesi si inserisce all'interno di un progetto più ampio denominato **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [1, 2, 3, 4, 5], una ricerca condotta dal Dipartimento di Scienze dell'Ingegneria dell'Università di Modena e Reggio Emilia e dal dipartimento dell'informazione dell'Università di Milano,sviluppato allo scopo di realizzare un tool semiautomatico per l'integrazione di sorgenti dati eterogenee e distribuite, siano esse strutturate o semistrutturate.
La particolarità del progetto è il tipo di approccio utilizzato per l'integrazione, cioè un approccio di tipo semantico.Questo consente un'integrazione basata non

solo sulla struttura delle sorgenti, ma anche sul significato che il progettista assegna ai singoli campi delle sorgenti stesse.
MOMIS adotta un'architettura a tre livelli:

1. livello *dati*
2. livello *mediatore*
3. livello *utente*

Il mediatore è la parte fondamentale del sistema, cioè il tool semiautomatico che realizza il vero e proprio processo di integrazione.Questi è formato da due macro blocchi:

1. *Global Schema Builder*
2. *Query Manager*

Global Schema Builder è il modulo che si occupa dell'integrazione delle sorgenti, il processo avviene grazie all'utilizzo del tool grafico SI_DESIGNER, che genera un unico schema globale.
Query Manager è il gestore delle interrogazioni, che, partendo da una singola query formulata dall'utente sullo schema globale, compone ed invia le query alle interfacce con le sorgenti.
L'interfacciamento con le sorgenti è garantito dai *Wrapper* che, presidiando le singole sorgenti, hanno il compito di fornire diversi servizi:

1. fornire una descrizione, nel linguaggio comune ODLi3, della sorgente alla quale e' connesso;
2. permettere l'esecuzione di query locali e fornire i risultati al mediatore;
3. esportare i significati, assegnati dal progettista della sorgente, relativi ai singoli campi;
4. esportare le relazioni intra-schema relative alla struttura;

Nella presente tesi ci si occuperá della parte relativa ai wrapper, in particolare l'obiettivo è realizzare un wrapper intelligente per sorgenti relazionali JDBC, cioè in grado di estrarre le relazioni intra-schema e permettere l'annotazione di interfacce ed attributi della sorgente relazionale che si vuole integrare.

La tesi è organizzata nel seguente modo:

Nel **Capitolo 1** verrà introdotta una metodologia di integrazione basata sull'utilizzo di tecniche di intelligenza artificiale, si parlerà in particolare dell'architettura I^3(Intelligent Integration of Information) proposta dall'agenzia ARPA degli Stati Uniti. Si tratteranno inoltre le problematiche relative alla progettazione di un mediatore e, nella parte finale, del progetto MOMIS.

Il **Capitolo 2** riporta una panoramica degli strumenti utilizzati nella realizzazione del software, in particolare CORBA, JDBC, WordNet e metodologie di estrazione di relazioni intra-schema.

Nel **Capitolo 3** viene illustrato il processo di integrazione di MOMIS.

Il **Capitolo 4** tratta la progettazione e la realizzazione del wrapper per sorgenti dati JDBC.

Nel **Capitolo 5** viene presentata l'interfaccia intelligente del wrapper che consente di introdurre comportamenti intelligenti sin da questo livello.

Sono inoltre presenti quattro appendici. In particolare in Appendice A viene riportato un glossario dei termini usati in ambito I^3, in Appendice B viene mostrata la sintassi di ODL_{I^3}, in Appendice C sono presenti le interfaccie IDL delle connessioni CORBA utilizzate e in Appendice D le descrizioni ODL_{I^3}delle sorgenti utilizzate nell'esempio di integrazione nel capitolo 3.

Capitolo 1

Un Sistema Intelligente per l'Integrazione delle Informazioni

La disponibilità di sorgenti di dati, soprattutto sulla rete, ha reso disponibili una grande quantitá di informazioni. Allo stesso tempo , però, l'accesso alle informazioni si è notevolmente complicato, questo è dovuto soprattutto alla grande eterogeneità dei dati disponibili, sia per quanto riguarda la natura (testi, immagini, etc.), sia il modo in cui vengono descritti. Gli standard esistenti (TPC/IP, ODBC, OLE, CORBA, SQL, etc.) risolvono parzialmente i problemi relativi alle diversità hardware e software, dei protocolli di rete e di comunicazione tra i moduli; rimangono però irrisolti quelli relativi alla modellazione delle informazioni. Difatti i modelli e gli schemi dei dati sono differenti e questo crea una eterogeneità semantica (o logica) non risolvibile da questi standard.

Gli approcci all'integrazione, descritti in letteratura presentano diverse metodologie come ad esempio il *repository independence*, e cioé un approccio che prevede di isolare al di sotto di una vista integrata le applicazioni ed i dati integrati dalle sorgenti, consentendo la massima autonomia e nascondendo al tempo stesso le differenze esistenti; i *datawarehouse* che realizzano presso l'utente finale delle viste, ovvero delle porzioni di sorgenti, replicando fisicamente i dati ed affidandosi ad algoritmi di 'allineamento' per assicurare la loro consistenza in caso di modifiche nelle sorgenti originali.

Verrà ora presentata la proposta dell'ARPA (Advanced Research Projects Agency) [6] per la realizzazione di un'architettura che, allo stesso tempo, consenta sia l'autonomia delle sorgenti, sia la flessibilitá dell'architettura stessa, senza dover ricorrere alla duplicazione fisica dei dati.

1.1 L'Integrazione Intelligente delle Informazioni

L'integrazione delle Informazioni (I^2) si distingue da quella dei dati e dei database in quanto non cerca di collegare semplicemente alcune sorgenti ma, piuttosto, di fornire risultati opportunamente selezionati da esse mediante l'uso di tecniche di Intelligenza Artificiale.Per ottenere i risultati cercati sono quindi necessarie *conoscenza* ed *intelligenza* finalizzate alla scelta delle sorgenti e dei dati, alla loro fusione e alla conseguente sintesi.Si parla allora di Integrazione Intelligente di Informazioni (*Intelligent Integration of Information*, I^3).

1.1.1 Il Programma I^3

Il programma I^3 è un progetto realizzato dall'ARPA, Agenzia facente capo al Dipartimento della Difesa degli Stati Uniti, che mira ad indicare un'architettura di riferimento che realizzi l'integrazione di sorgenti eterogenee in modo automatico. L'utilizzo di tecniche di intelligenza artificiale accresce il valore informativo dei dati trattati, permette, infatti, di dedurre informazioni direttamente dagli schemi delle sorgenti.

I^3prevede l'introduzione di architetture sviluppabili seguendo uno standard che definisca i servizi da inserire in qualsiasi integratore, ed abbassi i costi di sviluppo e mantenimento. Un approccio di questo tipo risolverebbe i problemi di realizzazione, manutenzione ed adattabilità, di eventuali supersistemi, creati ad arte per integrare una notevole quantità di sorgenti non correlate semanticamente. Riuscendo a riutilizzare la tecnologia già sviluppata, la costruzione di nuovi sistemi risulterebbe più veloce e meno difficoltosa, con conseguente abbassamento dei costi. Per poter sfruttare un'elevata riusabilità bisogna disporre di interfacce ed architetture standard. Il paradigma suggerito per la suddivisione dei servizi e delle risorse nei diversi moduli si articola su due dimensioni:

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

I domini nei vari livelli 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 finalià diverse.Ad esempio, si supponga di dover integrare informazioni su trasporti marittimi ed aerei: ciò permette all'utente di avere un'idea completa su quale mezzo di trasporto sia più conveniente per le sue esigenze. Aggiungendo a questo altri domini, quali le distanze chilometriche e le velocità dei mezzi, si possono, ad

esempio, facilitare le scelte di un rappresentante che deve decidere come e quando iniziare un determinato viaggio.

Il livello intermedio è quello su cui è più importante concentrare l'attenzione per quello che concerne l'uso di tecniche di intelligenza artificiale; esso infatti costituisce il tramite tra i dati posti nelle sorgenti e le applicazioni sviluppate per gli utenti. Questo livello deve fornire diversi servizi quali:

- servizi dinamici per la selezione delle sorgenti;
- gestione degli accessi;
- gestione delle *interrogazioni*;
- acquisizione e combinazione dei dati;
- analisi e sintesi delle informazioni ottenute.

Nell'Appendice A è presente un glossario di termini comunemente usato in ambito I^3.

1.2 Architettura di riferimento per sistemi I^3

L'obiettivo del programma I^3é quello di ridurre il tempo necessario per la realizzazione di un integratore di informazioni, fornendo una raccolta e una formalizzazione delle soluzioni prevalenti nel campo della ricerca. Le problematiche che stanno alla base del progetto I^3sono sostanzialmente due:

- la difficoltà che si ha nel ricercare delle informazioni all'interno della molteplicità delle sorgenti di informazione individuabili;
- la constatazione del fatto che le fonti di informazione e i sistemi informativi, pur essendo spesso semanticamente correlati tra di loro, non lo sono in una forma semplice né preordinata.

Proporremo un'architettura di riferimento che rappresenta alcuni dei servizi che un integratore di informazioni deve fornire e le possibili interconnessioni fra essi. Essa individua cinque famiglie di attività omogenee, illustrate in Figura 1.1 unitamente ai loro legami. La reciproca iterazione tra queste attività consente di eseguire le operazioni di comunicazione, traduzione ed integrazione dei dati nelle sorgenti.
Osservando i servizi individuati nell'architettura proposta è possibile notare che

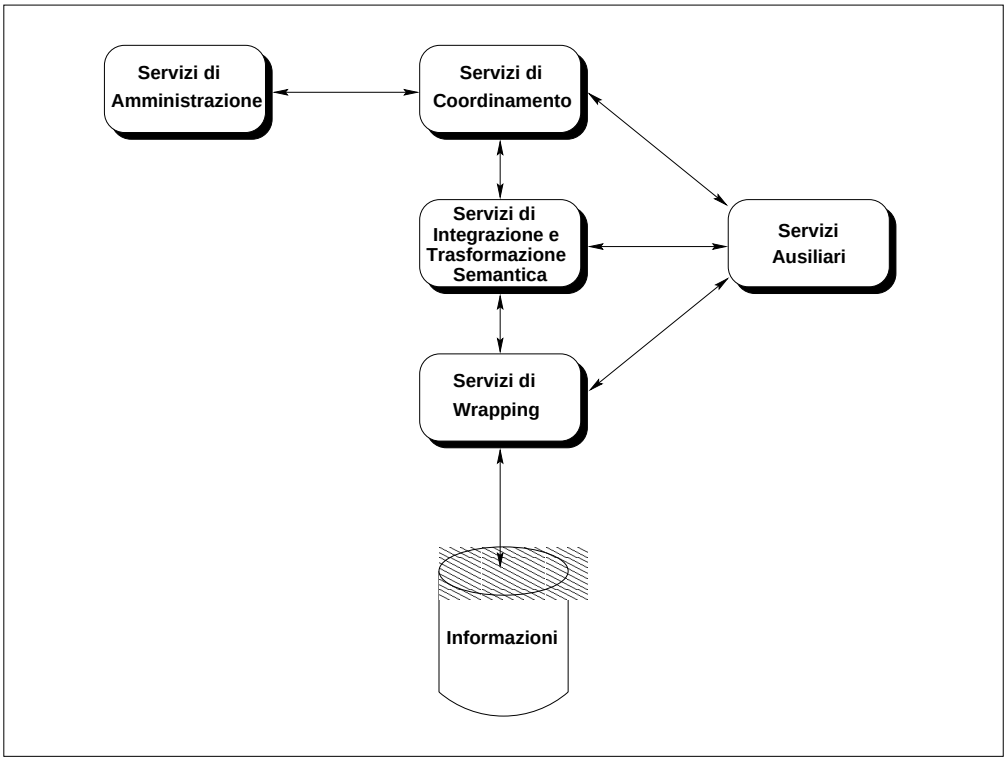

Figura 1.1: Diagramma dei servizi I^3

essi sono organizzati, sostanzialmente, su due assi:uno orizzontale e l'altro verticale.
Percorrendo l'asse verticale si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i servizi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti, tali informazioni vengono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazione Semantica, per poi essere passate ai servizi di Coordinamento che ne avevano fatto richiesta.
L'osservazione dell'asse orizzontale, invece, mette in risalto il rapporto tra i servizi di Coordinamento e quelli di Amministrazione ai quali spetta , infatti, il compito di mantenere informazioni sulle capacità delle varie sorgenti , ovvero che tipo di dati possono fornire ed in quale modo devono essere interrogate.Sono presenti inoltre i servizi Ausiliari, responsabili dei servizi di arricchimento semantico delle sorgenti. Vengono ora descritte nel dettaglio le specifiche funzionalità di ogni servizio e le problematiche che devono essere affrontate.

1.2.1 Servizi di Coordinamento

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

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

2. **Matchmaking**: il sistema viene configurato manualmente da un operatore al momento dell'inizializzazione. Successivamente tutte le richieste vengono trattate allo stesso modo. Sono definiti gli anelli di collegamento tra tutti i moduli del sistema, e nessuna ottimizzazione è fatta a tempo di esecuzione.

1.2.2 Servizi di Amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti *utili*, per determinare le loro capacità, e per creare ed interpretare TEMPLATE. I Tem-

plate sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task. Sono quindi utilizzati dai sistemi meno "intelligenti", e consentono all'operatore di predefinire le azioni da eseguire a seguito di una determinata richiesta, limitando al minimo le possibilità di decisione del sistema.
In alternativa all'uso dei Template, possono essere utilizzate le **Yellow Pages**: si tratta di servizi di directory che memorizzano le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata). Consultando queste Yellow Pages, il mediatore è in grado di spedire alla giusta sorgente la richiesta di informazioni, ed eventualmente di rimpiazzare questa sorgente con una equivalente nel caso non fosse disponibile. Fanno parte di questa categoria di servizi il Browsing: si permette all'utente di "navigare" attraverso le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste. Il servizio si basa sulla premessa che queste descrizioni siano fornite esplicitamente tramite un linguaggio dichiarativo leggibile e comprensibile all'utente. Inoltre tale servizio potrebbe contenere dei servizi di trasformazione del vocabolario e dell'ontologia, come pure di integrazione semantica. Da citare sono pure i servizi di Iterative Query Formulation: si tratta di sistemi che aiutano l'utente a rilassare o a meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

1.2.3 Servizi di Integrazione e Trasformazione Semantica

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

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

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

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

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

1.2.4 Servizi di Wrapping

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

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

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

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

1.2.5 Servizi Ausiliari

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

1.3 Il mediatore

Secondo la definizione proposta da Wiederhold in [7] "un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore ... Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti."
Compiti di un mediatore sono quindi:

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

Tra gli obiettivi principali del progetto MOMIS c'è quello di realizzare un Mediatore; infatti l'approccio architetturale scelto sviluppa il sistema su tre livelli:

1. utente: attraverso un'interfaccia grafica l'utente pone delle query su uno schema globale e riceve un'unica risposta, come se stesse interrogando un'unica sorgente di informazioni;
2. mediatore: il mediatore gestisce l'interrogazione dell'utente, combinando, integrando ed eventualmente arricchendo i dati ricevuti dai wrapper, ma usando un modello (e quindi un linguaggio di interrogazione) comune a tutte le fonti;
3. wrapper: ogni wrapper gestisce una singola sorgente, ed ha una duplice funzione: da un lato converte le richieste del mediatore in una forma comprensibile dalla sorgente, dall'altro traduce informazioni estratte dalla sorgente nel modello usato dal mediatore.

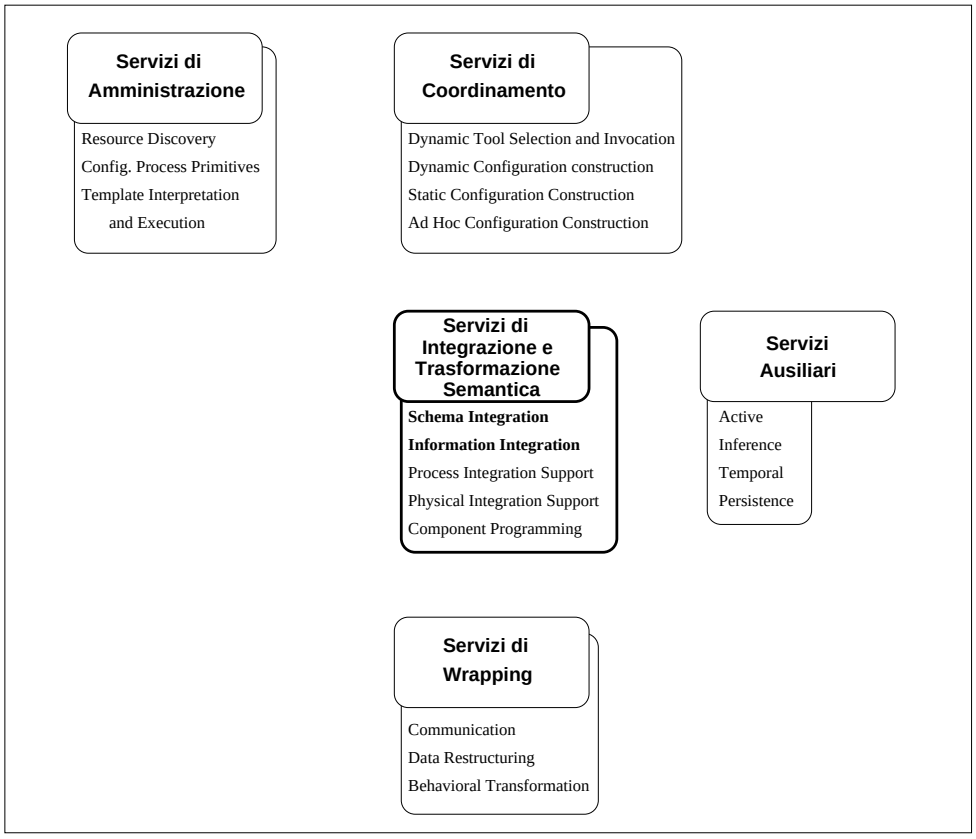

Figura 1.2: Servizi I^3 presenti nel mediatore

Facendo riferimento ai servizi descritti in precedenza, l'architettura del mediatore che si è progettato è riportata in Figura 1.2.
In questa tesi si è trattata la progettazione di un modulo che realizzi i servizi di Wrapping, in particolare per le sorgenti relazionali. Il software realizzato , quindi, deve servire al mediatore come interfaccia con le singole sorgenti, fornendo quattro funzionalità specifiche:

1. restituire la descrizione, in linguaggio ODL_{I^3}, della sorgente presidiata;

2. permettere l'esecuzione in locale di query generate dal query manager;

3. fornire al mediatore informazioni relative alla struttura della sorgente (Relazioni intra-schema);

4. esportare le informazioni lessicali della suddetta sorgente con l'ausilio del DataBase lessicale WordNet.

L'impostazione architetturale presentata dimostra che il mediatore in progettazione vuole distaccarsi dall'approccio *strutturale*, cioè sintattico, tuttora dominante tra i sistemi presenti sul mercato. L'approccio strutturale, adottato da sistemi quali TSIMMIS [8, 9, 10, 11], è caratterizzato dal fatto di usare un self-describing model per rappresentare gli oggetti da integrare, limitando l'uso delle informazioni semantiche alle regole predefinite dall'operatore. In pratica, il sistema non conosce a priori la semantica di un oggetto che va a recuperare da una sorgente (e dunque di questa non possiede alcuno schema descrittivo) bensì è l'oggetto stesso che, attraverso delle etichette, si autodescrive, specificando tutte le volte, per ogni suo singolo campo, il significato che ad esso è associato. Questo approccio porta quindi ad un insieme di vantaggi, tra i quali possiamo identificare:

- la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo: il mediatore non si basa infatti su una descrizione predefinita degli schemi delle sorgenti, bensì sulla descrizione che ogni singolo oggetto fa di sè. Oggetti simili provenienti dalla stessa sorgente possono quindi avere strutture differenti, cosa invece non ammessa in un ambiente tradizionale object-oriented;
- per trattare in modo omogeneo dati che descrivono lo stesso concetto, o che hanno concetti in comune, ci si basa sulla definizione manuale di rule, che permettono di identificare i termini (e dunque i concetti) che devono essere condivisi da più oggetti.

Altri progetti, e tra questi quello proposto, seguono invece un approccio definito *semantico*, che è caratterizzato dai seguenti punti:

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

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

1.3.1 Problematiche da affrontare

Pur avendo a disposizione gli schemi concettuali delle varie sorgenti, non è certamente un compito banale individuare i concetti comuni ad esse, le relazioni che possono legarli, né tantomeno è banale realizzare una loro coerente integrazione. Trascurando per il momento le differenze dei sistemi fisici, (alle quali dovrebbero provvedere i moduli wrapper) i problemi che si sono dovuti risolvere, o con i quali occorre giungere a compromessi, sono (a livello di mediazione, ovvero di integrazione delle informazioni) essenzialmente di due tipi:

1. problemi ontologici;
2. problemi semantici.

1.3.2 Problemi ontologici

Come riportato nel glossario A, per ontologia si intende, in questo ambito, "l'insieme dei termini e delle relazioni usate in un dominio, che denotano concetti ed oggetti". Con ontologia quindi ci si riferisce a quell'insieme di termini che, in un particolare dominio applicativo, denotano in modo univoco una particolare conoscenza e fra i quali non esiste ambiguità poiché sono condivisi dall'intera comunità di utenti del dominio applicativo stesso. Un'ontologia può essere più o meno generale a seconda del livello cui ci si riferisce: se ci si riferisce ad una precisa applicazione l'ontologia sarà limitata dipendendo dal dominio e dall'obiettivo della stessa. Le ontologie di livello superiore sono più vaste riferendosi solo al dominio ma essendo indipendenti dall'obiettivo, quelle di livello ancora superiore sono indipendenti anche dal dominio e definiscono concetti molto generali. Per focalizzare il concetto viene riportata una semplice classificazione delle ontologie (mutuata da Guarino [12, 13]). I livelli di ontologia sono essenzialmente tre:

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

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

Per quanto riguarda questo progetto, si è considerato di operare su sorgenti vincolate all'interno di un certo dominio, supponendo quindi che le fonti informative condividano almeno i concetti fondamentali (ed i termini con cui identificarli).

1.3.3 Problemi semantici

L'ipotesi che i progettisti di sorgenti diverse abbiano una visione similare del problema da modellare, ovvero utilizzino un insieme di concetti comuni, non garantisce che sistemi diversi usino esattamente gli stessi vocaboli per rappresentare gli stessi concetti, anzi è una condizione piuttosto improbabile; stessa considerazione vale anche per le strutture dati. Come riportato in [14] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma questa non è l'unica. Le differenze nei sistemi di DBMS possono portare all'uso di differenti modelli per la rappresentazione della porzione di mondo in questione: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate attraverso un modello relazionale, o ad oggetti.
L'obiettivo dell'integratore, quindi, è identificare i concetti comuni all'interno di queste sorgenti e risolvere le differenze semantiche che possono essere presenti tra di loro. Le differenze semantiche possono essere classificate in tre gruppi principali:

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

2. **eterogeneità tra le strutture delle classi**: comprendono le differenze nei criteri di specializzazione, nelle strutture per realizzare un'aggregazione, ed anche le *discrepanze schematiche*, quando cioè valori di attributi sono invece parte dei metadati in un altro schema.

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.

Le differenze semantiche però possono essere utilizzate per arricchire il nostro sistema, infatti, studiando attentamente queste differenze, si può giungere al

cosiddetto *arricchimento semantico*, cioè la possibilità di aggiungere esplicitamente ai dati tutte quelle informazioni che erano originariamente presenti solo come metadati negli schemi, dunque in un formato non interrogabile.

1.4 Il sistema MOMIS

Il sistema **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources), nasce all'interno del progetto **MURST 40%** ***INTERDATA*** (97/98) per rispondere all'esigenza non solo di ambienti software "$intelligenti$" in grado di fornire accesso a grosse moli di dati, ma anche capaci mi aumentare la *qualità* delle informazioni ottenibili. L'obiettivo del progetto è quello di realizzare uno strumento semi-automatico che consenta una reale integrazione di sorgenti distribuite, eterogenee, strutturate e semistrutturate. L'approccio di integrazione adottato é di tipo *semantico* e *virtuale* [15]. Per "$semantico$" si intente quanto illustrato nella sezione precedente, mentre con virtuale si intende che la vista integrata delle sorgenti non viene *materializzata* in locale, ma sarà il sistema che, basandosi sull'individuazione delle sorgenti da interrogare, provvederà alla generazione delle subquery da eseguire direttamente sulle sorgenti.Si opera quindi su di uno schema globale(vista virtuale integrata) che rappresenta le varie sorgenti collegate.

L'adozione di questo approccio è stata dettata da varie motivazioni:

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

1.4.1 Il Modello dei dati

All'interno del sistema è stato adottato un modello comune dei dati (ODM$_{I^3}$).
La base di partenza per la definizione di questo modello è rappresentata dalle raccomandazioni relative alla proposta di standardizzazione per linguaggi di mediazione, risultato del lavoro svolto in ambito I^3. Tali raccomandazioni sottolineano

la necessità per un mediatore di poter gestire sorgenti con modelli complessi, come quello ad oggetti, e sorgenti molto più semplici come file di strutture: l'impiego di un formalismo il più possibile completo, e quindi in grado di rappresentare in modo adeguato tutte le possibili situazioni, risulta una possibile soluzione.

Per quanto riguarda il linguaggio di definizione degli schemi si è cercato di cogliere le indicazioni emerse in ambito I^3discostandosi, nel contempo, il meno possibile dal linguaggio ODL proposto dal gruppo di standardizzazione ODMG-93. Si è così definito il linguaggio ODL_{I^3} come estensione del linguaggio standard ODL in modo da supportare le necessità del nostro mediatore.

Le principali caratteristiche del linguaggio ODL_{I^3}sono:

- possibilità di rappresentare sorgenti strutturate (database relazionali, ad oggetti, e file system) e semistrutturate (XML). Ciò significa che tutte le fonti di informazione, indipendentemente dal modello originario, e lo schema globale verranno descritti mediante il modello comune, facendo quindi riferimento al concetto di classe ed aggregazione (sarà poi compito dei Wrapper provvedere alla traduzione in termini del modello originale);

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

- 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;

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

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

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

- traduzione automatica e trasparente all'utente delle descrizioni nella logica descrittiva **OLCD**, con conseguente possibilitá di utilizzare comportamenti intelligenti nei controlli di consistenza e nell'ottimizzazione semantica delle interrogazioni;

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

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

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

- dichiarazione di relazioni estensionali: il sistema MOMIS integra gli schemi locali secondo criteri sia intensionali che estensionali. I conflitti intensionali rappresentano quelle incompatibilità derivanti dall'avere porzioni di schemi sovrapposte, ossia gli stessi aspetti del dominio applicativo rappresentati usando strutture differenti. Ciò che occorre fare è quindi fornire una rappresentazione unificata ed omogenea dei medesimi concetti descritti in sorgenti differenti.
  L'integrazione degli schemi non è però l'unico aspetto che occorre gestire per ottenere un'effettiva integrazione di sorgenti eterogenee: è necessario risolvere anche i conflitti derivanti dalla sovrapposizione delle estensioni, cioè dalla presenza, in sorgenti diverse, di informazione relativa alla stessa entità del "mondo reale". Per arrivare ad un'integrazione corretta può non essere sufficiente fare una semplice unione delle estensioni delle classi, in quanto dati relativi alla stessa entità potrebbero essere presenti in più classi. Questi inconvenienti possono essere risolti soltanto gestendo in modo adeguato le relazioni fra le estensioni. Si introducono, a tale scopo, gli *assiomi estensionali* [16], i quali descrivono le relazioni insiemistiche esistenti fra le estensioni delle sorgenti. Ogni assioma estensionale definito *vincola* le classi coinvolte ad avere anche un legame intensionale.

Utilizzando questo linguaggio (vedi Appendice B), il wrapper compie la traduzione delle classi da integrare e la fornisce al mediatore: da sottolineare che le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema, e quindi interrogabili. Non è detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente, bensì ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa. Per quanto riguarda il linguaggio di interrogazione si è adottato OQL_{I^3} che adotta la sintassi OQL senza discostarsi dallo standard. Questo linguaggio richiede un maggiore sforzo dal punto di vista dello sviluppo di moduli per l'interpretazione e la gestione delle interrogazioni (implementando le funzionalità tipiche di un ODBMS), ma risulta altresì estremamente versatile ed espressivo fornendo la possibiltà di sfruttare le informazioni rappresentate nello

schema globale. Le maggiori difficoltà implementative sono quindi ampliamente giustificate da una maggiore versatilità e da una migliore facilità d'uso per l'utente finale.

1.4.2 L'architettura generale di MOMIS

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

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

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

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

L'ultimo modulo del Mediatore è rappresentato dall'**Extensional**

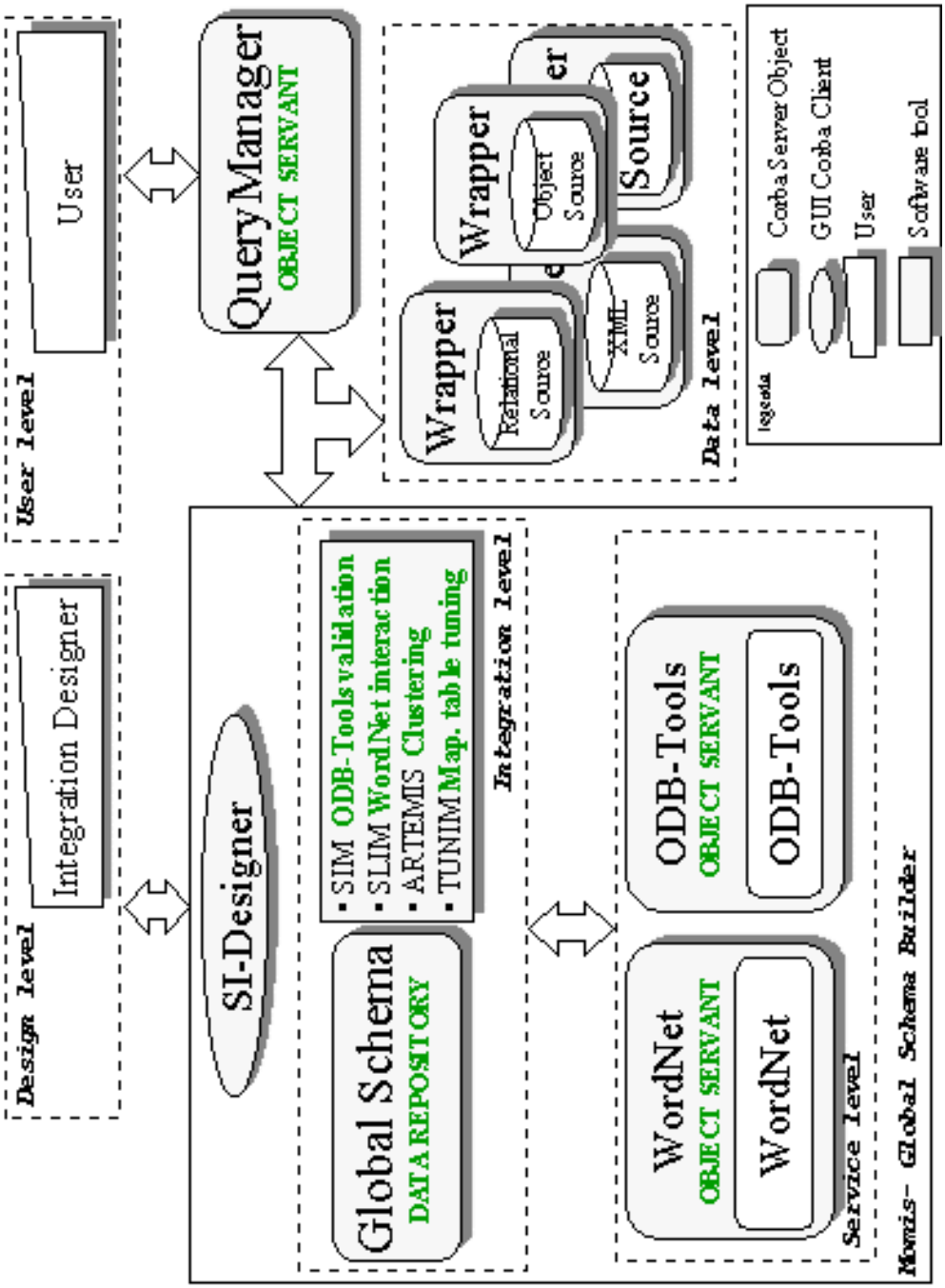

Figura 1.3: Architettura generale di MOMIS

Hierarchy Builder il quale si occupa della generazione della Conoscenza Estensionale (Gerarchie Estensionali e Base Extension) necessaria per ottimizzare le interrogazioni.
Il modulo software utilizzato per l'integrazione si chiama SI-Designer, un tool semiautomatico che realizza le funzionalità sopra descritte.

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

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

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

Livello Utente. L'utilizzatore del sistema dovrà potere interrogare lo schema globale. L'accesso ai dati si propone del tutto simile a come avvengono le usuali interrogazioni di un DataBase tradizionale: le sorgenti ed il modo in cui i dati vengono recuperati risultano all'utente del tutto trasparenti, in quanto é il sistema ad occuparsi di tutte le operazioni necessarie per reperire le informazioni e combinare le risposte in un'unica risposta corretta, completa e non ridondante.

Vediamo una rapida descrizione dei tool di ausilio al mediatore precedentemente citati:

- *ODB-Tools* è uno strumento software sviluppato presso il dipartimento di Ingegneria dell'Università di Modena e Reggio Emilia [17, 18]. Esso si occupa della validazione di schemi e dell'ottimizzazione semantica di interrogazioni rivolte a Basi di Dati orientate agli Oggetti (OODB). Facciamo riferimento alla figura 1.4 per una breve spiegazione. ODB-Designer si occupa della validazione di schemi: si può inserire la descrizione di uno schema di DataBase (in ODL) ed il sistema realizzerà automaticamente la sua

validazione e la sua riclassificazione; vale a dire che si occuperà di verificare che non esistano classi incoerenti (cioè che non possano essere popolate da nessun oggetto) e di determinare eventuali relazioni di specializzazione non esplicitate dallo schema stesso. ODB-Qoptimizer si occupa invece dell'ottimizzazione semantica delle interrogazioni: se si inserisce una query (in OQL) posta su di un determinato schema, questa viene automaticamente riformulata in una equivalente, ma più efficiente, sfruttando l'espansione semantica ed i vincoli di integrità.

- *WordNet* [19] (per ora ne viene data una breve descrizione ma sarà trattato più ampiamente in seguito) è un DataBase lessicale in lingua inglese capace di individuare relazioni semantiche fra termini; cioè, dato un insieme di termini, WordNet è in grado di identificare l'insieme di relazioni lessicali che li legano.

- *ARTEMIS* [20] riceve in ingresso il *thesaurus*, cioè l'insieme delle relazioni terminologiche (lessicali e strutturali) precedentemente generate, e sulla base di queste assegna ad ogni classe coinvolta nelle relazioni un coefficiente numerico indicante il suo grado di affinità. Questi coefficienti verranno utilizzati per raggruppare le classi locali in modo tale che ogni gruppo (*cluster*) comprenda solo classi con coefficienti di affinità simili.

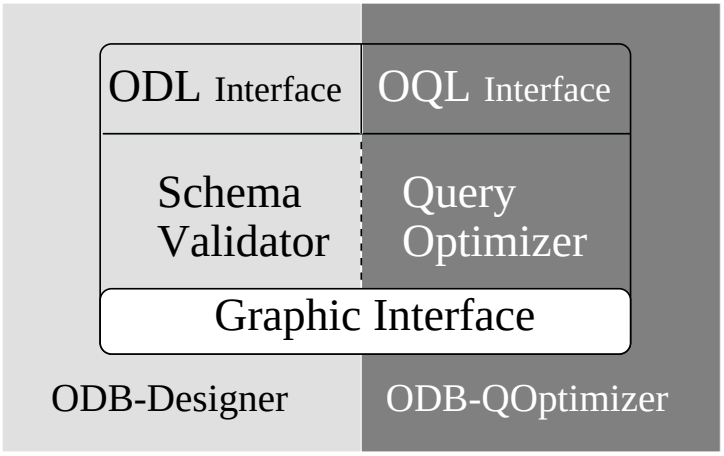

Figura 1.4: Architettura ODB-Tools

Capitolo 2

Gli strumenti utilizzati

L'integrazione di informazioni, così come intesa nell'architettura I^3, richiede l'utilizzo di tecniche di intelligenza artificiale ma non solo, si rendono necessari, altresì, strumenti per mettere in comunicazione i vari moduli software tra di loro e gli stessi con le sorgenti di dati. In questo capitolo si darà una descrizione approfondita degli strumenti utilizzati nella presente tesi; si parlerà dunque di tecnologia JDBC, architettura CORBA, WordNet e tecniche di estrazione ed inferenza di relazioni terminologiche.

2.1 La Tecnologia JDBC

La tecnologia JDBC consiste in un insieme di API che permettono l'accesso ad ogni sorgente dati tabulare dal linguaggio di programmazione JAVA. Permette quindi non solo l'accesso ad una vasta gamma di database SQL ma anche ad altre sorgenti dati di tipo tabulare, come ad esempio i fogli elettronici. Le API JDBC possono inoltre essere utilizzate per interagire con sorgenti dati multiple in ambienti eterogenei e distribuiti.
Il vantaggio di JDBC rispetto ad altri approcci è che l'accesso alle sorgenti è di tipo *virtuale* ed avviene grazie alla Java Virtual Machine. In altre parole, con le API JDBC, non è necessario scrivere un'applicazione per il database Oracle, una per IBM DB2, una per SQL Server, ecc... Ogni programma che utilizzi questa tecnologia avrà la capacità di comunicare con una qualunque sorgente dati, a patto che per essa sia stato implementato il relativo driver JDBC.
Semplificando, un driver JDBC rende possibili tre operazioni:

- stabilire una connessione con la sorgente dati;
- inviare Query;
- elaborare i risultati ottenuti.

LE API JDBC sono contenute nei package:

- java.sql;
- javax.sql.

La tecnologia JDBC è basata su X/Open SQL CLI, che è anche il fondamento di ODBC, fin dalla sua introduzione, nel gennaio del 1997, è stata ampiamente accettata ed utilizzata, grazie alla sua semplicità e flessibilità.

L'utilizzo di JDBC, in questa tesi, ha consentito di realizzare il collegamento tra il Wrapper e la sorgente dati relazionale da presidiare. Nel seguito del capitolo verrà descritto l'accesso ad una sorgente dati con l'utilizzo delle API JDBC.

2.1.1 Stabilire una connessione

Per iniziare il dialogo con una sorgente dati è necessario innanzitutto stabilire una connessione, creare cioè un canale di comunicazione tra il modulo software e la sorgente dati. L'oggetto *Connection* rappresenta la connessione al database, in ogni sessione di connessione vengono inviati statement SQL e ricevuti risultati.
Un'applicazione può avere più connessioni con un singolo database oppure collegarsi a più database. Il modo classico per stabilire una connessione con la sorgente dati è chiamare il metodo *DriverManager.getConnection* passandogli come parametro un URL. La classe *DriverManager* è dotata di una lista di driver registrati e, quando viene invocato il metodo *getConnection*, la interroga fino a che non trova un driver capace di realizzare il collegamento con il database rappresentato nell'URL.
Allo sviluppatore è lasciata la possibilità di "Saltare" il livello JDBC per lavorare direttamente sui metodi dei *Driver*, anche se è preferibile delegare le operazioni di connessione alla classe *DriverManager*. Si riporta a titolo di esempio il codice utilizzato per realizzare la connessione con la sorgente dati relazionale (DB2):

Class.forName(COM.ibm.db2.jdbc.app.DB2Driver);
per indicare il driver da utilizzare

connection = DriverManager.getConnection(jdbc:db2:univers);
per stabilire la connessione vera e propria

In fase di implementazione il driver specifico per DB2 si è rivelato inadeguato alle esigenze del progetto MOMIS; esso infatti rende difficoltoso l'accesso alla sorgente da parte di utenti diversi dal proprietario. Si è reso quindi necessario l'utilizzo di un'interfaccia con il driver chiamata RmiJdbc la quale, ponendosi tra l'applicazione ed il driver specifico, permette ad ogni utente di interrogare la

sorgente.
Il codice relativo alla connessione è stato così modificato:
Class.forName(RmiJdbc.RJDriver);

_connection = DriverManager.getConnection(jdbc:rmi://sparc20.ing.-unimo.it:1100/jdbc:db2:univers);
dove 1100 e' il numero della porta utilizzata.

La collocazione di RmiJdbc è mostrata in figura 2.1.

2.1.2 Esecuzione di statements SQL e manipolazione dei risultati

Per inviare statement SQL ad un database si utilizzano gli oggetti *Statement*, attualmente esistono tre tipi di oggetti statement, ognuno dei quali permette di eseguire statement SQL su una data connessione:

1. *Statement*: usati per eseguire semplici statement SQL senza parametri;
2. *PreparedStatement*: usati per eseguire statement SQL precompilati con o senza parametri d'ingresso;
3. *CallableStatement* usati per invocare una store procedure.

Per crearli si utilizza il metodo *createStatement* della classe Connection, come mostrato nell'esempio seguente:

Connection con = DriverManager.getConnection(URL);
Statement stmt = con.createStatement();

Gli statement SQL verranno inviati al database come argomenti di uno dei metodi *execute* di un oggetto Statement, ad esempio:

ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table2");

La variabile rs si riferisce al ResultSet, cioè alla risposta del database, ovviamente una eventuale modifica ad rs non sortirà nessun effetto sulla base di dati.
Ci sono tre tipi di metodi differenti per eseguire statement SQL:

1. *executeQuery*: serve per quei comandi che producono un singolo result set, come ad esempio una query di selezione (SELECT);
2. *executeUpdate*: viene utilizzato per eseguire statement del tipo INSERT, UPDATE o DELETE, oppure comandi per la definizione dei dati (SQL DDL) come ad esempio CREATE TABLE, DROP TABLE, ALTER TABLE. Il valore di ritorno di executeUpdate è un intero che indica il numero di righe modificate. Per comandi che non operano sulle righe (CREATE TABLE,...) il valore di ritorno è sempre zero.

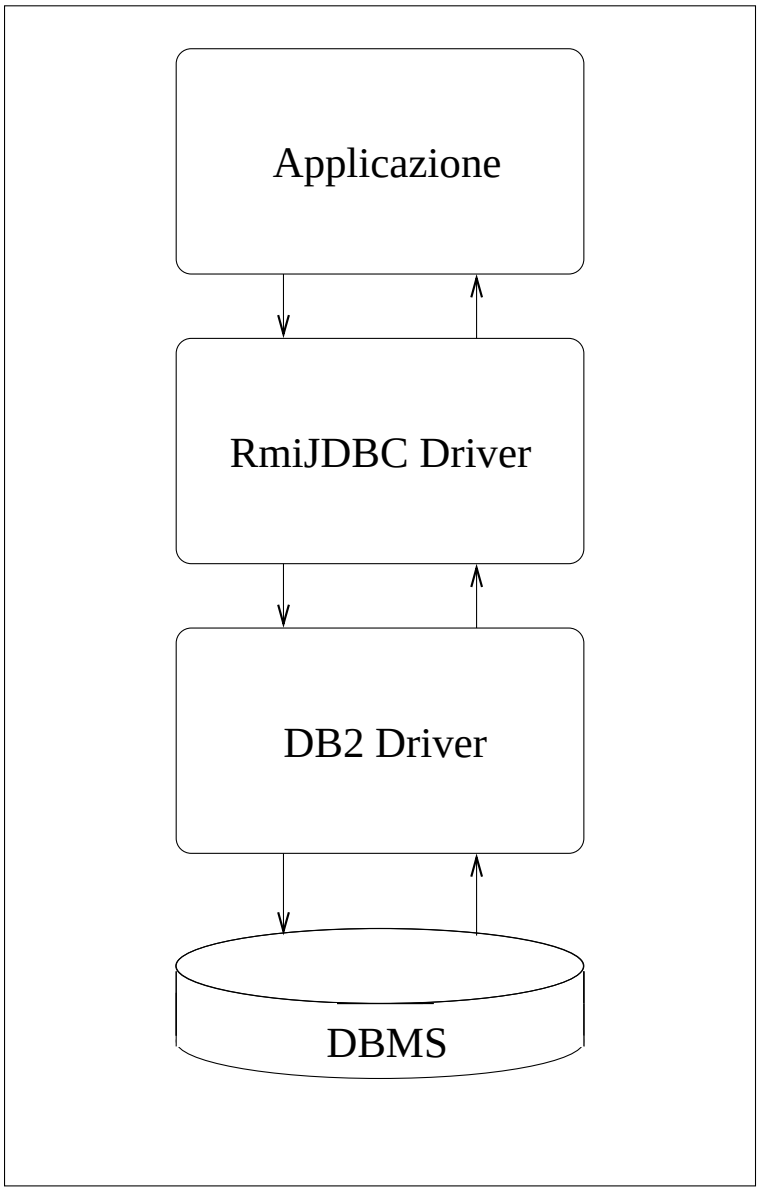

Figura 2.1: Il driver RmiJdbc

3. *execute*: Si usa per quei comandi che producono più di un result set.

Normalmente gli statement sono in modalità auto-Commit, per evitare ciò è necessario specificare *con.setAutoCommit(false)*.Una volta utilizzati, gli statement vengono chiusi o direttamente da programma o dal Java Garbage Collector.

2.1.3 DataBaseMetaData

Finora si è parlato di come realizzare la connessione ad un database e di come interrogarlo, le descrizioni ODL$_{I^3}$delle sorgenti dati, però, non contengono i dati memorizzati nelle sorgenti bensì informazioni riguardanti la struttura delle stesse. Le informazioni necessarie sono:

- nomi delle tabelle;
- nomi e tipi dei campi contenuti nelle tabelle;
- chiavi primarie di ogni entità;
- foreign key;
- candidate key.

Le API JDBC mettono a disposizione un'interfaccia chiamata *DataBaseMetaData* che dà la possibilità di ottenere questi dati, o meglio questi *metadati*. *DataBaseMetaData* include molti metodi che possono essere suddivisi sulla base delle informazioni che forniscono:

- informazioni generali sulla sorgente dati;
- limiti della sorgente dati;
- oggetti SQL contenuti nella sorgente e loro attributi;
- ...

Gli strumenti analizzati fino ad ora consentono al wrapper di accedere alla sorgente dati, interrogarla e ottenere le informazioni necessarie alla produzione della descrizione ODL$_{I^3}$. Nella sezione successiva 2.2 verranno trattati gli strumenti utilizzati per trasferire questa descrizione al mediatore.

2.2 La comunicazione tra moduli:CORBA

Per CORBA [21] si intende **C**ommon **O**bject **R**equest **B**roker **A**rchitecture, un'architettura standard, distribuita e ad oggetti sviluppata dall'**O**bject **M**anagement **G**roup, un consorzio di aziende ed altre organizzazioni fondato nel 1989 per promuovere lo sviluppo di software basato su componenti attraverso la ratificazione di standard e di linee guida per lo sviluppo. L'obiettivo delle specifiche CORBA è quello di definire un modello standard per la comunicazione tra oggetti indipendentemente dal linguaggio di implementazione e dal sistema operativo utilizzato. Il componente principale attorno a cui ruota tutta l'architettura è l'ORB (**O**bject **R**equest **B**roker), quest'ultimo infatti rappresenta l'entità software attraverso la quale oggetti diversi possono interagire attraverso rete indipendentemente da dove si trovino e dal linguaggio di programmazione utilizzato per implementarli.
Insieme alla definizione dello standard è stato creato anche un linguaggio di descrizione che consente di creare oggetti CORBA l'IDL (**I**nterface **D**efinition **L**anguage). Gli ORB comunicano tra di loro utilizzando il protocollo IIOP (**I**nternet **I**nter**O**rb **P**rotocol) anch'esso definito dall'OMG.

2.2.1 L'architettura

Ogni interazione tra due oggetti distribuiti ha due facce: una Client ed una Server. Il server fornisce l'interfaccia remota, mentre il client la invoca.
Questo tipo di modello è comune a molte tecnologie di comunicazione, comprese CORBA ed RMI. Si noti che , in questo contesto, i termini Client e Server sono definiti a livello di oggetti, mentre se consideriamo le applicazioni possono essere server per alcuni oggetti e client per altri. Infatti un oggetto può essere client per l'interfaccia di un oggetto remoto e , allo stesso tempo, implementare un'interfaccia che può essere chiamata da altri oggetti.
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.
L'invocazione del metodo di un oggetto CORBA avviene in questo modo:

- dal lato client l'object reference ha uno stub il quale, se chiamato, invoca i metodi di connessione dell'ORB che inoltrano le richieste all'applicazione server;

- l'ORB locale chiede all'ORB remoto di stabilire una connessione;

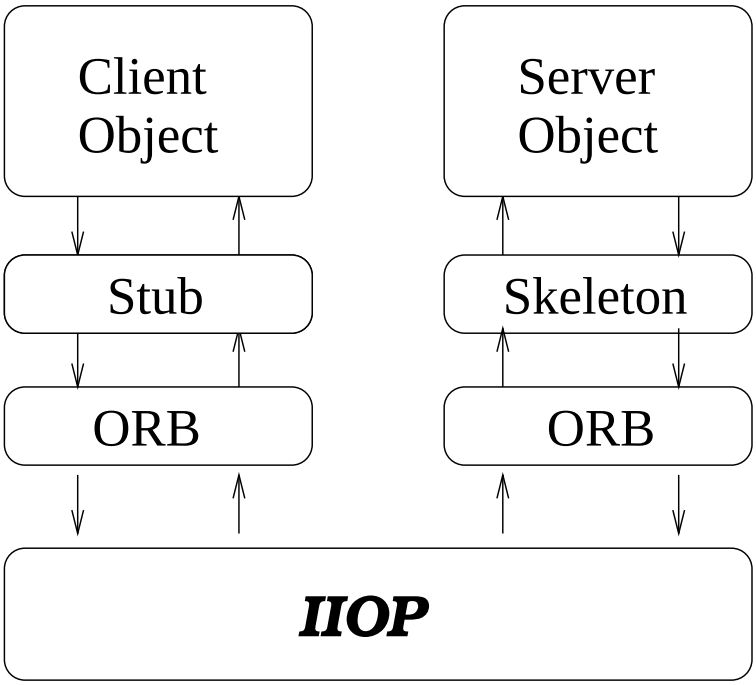

Figura 2.2: Architettura CORBA

- nel lato server l'ORB usa il codice skeleton, che in pratica ha la stessa funzione dello stub ma lato server, per convertire l'invocazione remota in una chiamata ad un metodo di un oggetto locale;
- una volta prodotto il risultato sarà lo skeleton a riconvertirlo e ad inviarlo al client attraverso gli ORB.

Come mostrato in figura 2.2 la comunicazione tra ORB avviene, nella maggior parte dei casi, attraverso un protocollo condiviso chiamato *IIOP* (**I**nternet **I**nter **O**rb **P**rotocol). La struttura di *IIOP* si basa sul protocollo standard TCP/IP e definisce come gli oggetti compatibili con CORBA devono scambiarsi le informazioni.

2.2.2 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*.Naming Service è un servizio standard per applicazioni CORBA, esso permette di associare nomi astratti ad oggetti CORBA e consente

ai client di trovare questi oggetti cercando i nomi corrispondenti. 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.

ORB

L'ORB è la chiave del funzionamento di CORBA, la colla che mantiene insieme gli stub, gli skeleton e i servizi di risoluzione dei nomi. Esso ha il compito di comunicare con lo stub, di trovare l'oggetto server e comunicare con gli skeleton.In pratica offre un'astrazione che permette ai vari oggetti di comunicare indipendentemente dai linguaggi di programmazione con i quali sono implementati e i sistemi su cui "girano".

Interoperabilità tra ORB

In una stessa applicazione distribuita possono essere utilizzate differenti implementazioni di ORB. Un client può utilizzare contemporaneamente due ORB diversi per accedere ad oggetti multipli, o un unico ORB per accedere ad un oggetto gestito da un altro ORB. Visto che possono esistere più ORB in una stessa applicazione, è necessario che siano in grado di comunicare tra loro.
L'interfaccia attraverso la quale comunicano genericamente due ORB si chiama GIOP (**G**eneral **I**nter **O**rb **P**rotocol).Essa specifica il formato e la sintassi dei messaggi per la comunicazione tra ORB ma non indica un protocollo di comunicazione. Come già accennato in precedenza nel caso il protocollo sia il TCP/IP si parla di IIOP (**I**nternet **I**nter**O**rb **P**rotocol.

2.2.3 Interface Definition Language

L' **I**nterface **D**efinition **L**anguage (IDL) è un linguaggio di tipo descrittivo nato in ambiente OMG. Offre un modo indipendente dal linguaggio per descrivere le interfacce degli oggetti. Esso definisce i metodi contenuti nell'interfaccia, i loro argomenti e i valori di uscita ma non specifica come le intefacce siano implementate.
L'interfaccia di un oggetto server è specificata in IDL, e le specifiche IDL sono compilate per produrre lo stub e lo skeleton da utilizzare per quell'oggetto. Una volta prodotta la specifica in IDL è possibile utilizzarla con qualunque linguaggio per cui sia stato prodotto un compilatore IDL, ad esempio si potrebbe compilare un'interfaccia in C++ per la struttura server e fare lo stesso in Java per creare lo stub.
I compilatori IDL sono disponibili per C, C++, Smalltalk, Ada e Java. Questi traducono il codice IDL nel codice sorgente di questi linguaggi; sarà poi un compilatore specifico a portare i codici da sorgente a binario. Ad esempio, in Java una volta definite le interfacce con un semplice tool messo a disposizione dalla Sun(idltojava), le definizioni delle interfacce 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.

Esempio 1 *Supponiamo di aver creato una descrizione IDL chiamata* esempio.idl*, contenente l'interfaccia esempio. Eseguendo il comando:*

idltojava esempio.idl

Si generano cinque file Java nella sottodirectory esempio:

1. esempio.java*: definisce l'interfaccia* esempio *risultante dalla mappatura da idl a Java*

2. _esempioImplBase.java*: è il file di skeleton per l'interfaccia* esempio.*È la superclasse della classe che viene utilizzata per implementare l'oggetto server remoto.*

3. _esempioStub.java*: il file di stub per l'interfaccia* esempio.

4. esempioHelper.java*: definisce metodi static ausiliari che possono essere utilizzati dal client.*

5. esempioHolder.java*: definisce una classe per contenere un oggetto dell'interfaccia esempio. Questa classe è utilizzata per supportare il passaggio dei parametri tra java e CORBA.*

In appendice C verranno riportate le interfacce IDL realizzate nella fase di sviluppo del software.

2.3 Il Database lessicale WordNet

WordNet è un database lessicale basato sulle più moderne teorie psicolinguistiche della memoria lessicale umana. Sviluppato sotto la direzione del professore George A.Miller presso l'università di Princeton.
WordNet è oggi considerato la più importante risorsa disponibile per i ricercatori nei campi della linguistica computazionale, dell'analisi testuale, e delle aree associate.

I dizionari cartacei sono organizzati con un ordinamento alfabetico perché è l'unico modo che permette ad un lettore di trovare le parole cercate sfogliandolo una sola volta. Questo approccio però è lontano dall'essere perfetto, infatti vengono accostate parole con significati diversi e i termini correlati sono sparpagliati in modo casuale. Per ogni lemma vengono presentati tutti i significati assieme, anche se possono non aver nulla in comune, per esempio `number` ha i significati di "cifra" o "quantità", ma anche di "performance teatrale" o di fascicolo di una "rivista".

WordNet non si pone in competizione con i dizionari tradizionali per le informazioni reperibili, per esempio non mostra la sillabazione, la pronuncia, le forme derivate, l'etimologia, le definizioni ed esempi di usi alternativi, note sugli usi speciali, immagini o grafici descrittivi, né per completezza lessicale: il numero di termini lo pone al livello di un dizionario da *college*.
Ciò che lo rende uno strumento innovativo è:

- la comprensione della differenza tra lemmi e significati;
- le relazioni, che permettono navigabilità alle informazioni;
- la strutturazione interna delle categorie sintattiche.

2.3.1 La Matrice Lessicale

Il principio della semantica lessicale è la constatazione dell'esistenza di un'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 2.3): l'elemento $M_{i,j}$ che assume il valore $E_{i,j}$ indica che la parola F_j

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 2.3: La Matrice Lessicale

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*: concetto 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. Durante una qualsiasi forma di comunicazione, il ”destinatario” riceve la parola, cercando di capirne il significato tra tutti quelli che la stessa può esprimere, il ”mittente” conosce il concetto che 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. L’utente deve quindi conoscere sufficientemente bene la lingua (inglese), e pertanto essere in grado di riconoscere il significato di una determinata parola in base ai vari sinonimi ad essa associati (senza che ne venga data esplicitamente una definizione).

2.3.2 Tipi di relazioni

Nel Database lessicale WordNet ogni categoria sintattica (nomi, verbi, avverbi ed aggettivi)è organizzata in insiemi di sinonimi (chiamati *synset*, insiemi di sinonimi) che rappresentano un concetto lessicale. Non si vedranno 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. La sinonimia tra termini è la relazione più importante, perché ogni insieme *synset* che ne scaturisce rappresenta la semantica di un concetto.

- **Antinomia.** *Due termini sono in relazione di antonimia se uno è il contrario dell'altro.*
  Da osservare che l'antinomo del termine x non sempre coincide con '$non\ x$': ad esempio 'gioioso' e 'triste' sono in relazione di antinomia, ma 'non triste' non coincide con 'gioioso', dal momento che esistono una serie di stati d'animo intermedi tra i due.
  Tra le relazioni citate, l'antinomia 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 antinomi, 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 'studente' è iponimo di 'persona' e 'persona' è a sua volta iponimo di 'essere vivente'. L'iponimia gode della proprietà transitiva: nell'esempio si deduce che 'studente' è iponimo di 'essere vivente'. Questa proprietà consente la costruzione di 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.

2.3.3 Legame tra le relazioni in WordNet e in MOMIS

I diversi ambienti da cui provengono WordNet e MOMIS, il primo linguistico ed il secondo informatico, fanno si che si abbiano sostanziali differenze per quanto riguarda le terminologie utilizzate per definire gli stessi concetti. Se ne propone quindi un quadro riassuntivo:

Relazione Lessicale

MOMIS	WordNet
Relazione estratta utilizzando il database lessicale WordNet	Relazione tra lemmi, in contrapposizione ad una relazione semantica

Relazione Semantica

MOMIS	WordNet
Le relazione che vengono incluse nel *Common Thesaurus* sono tutte semantiche, quelle estratte con l'ausilio di SLIM vengono promosse a semantiche dalla validazione del progettista	Relazioni tra significati

Delle relazioni che si possono ricavare da WordNet sono state prese in esame le relazioni di seguito riportate:

Sinonimia si trasforma in una relazione SYN, dove per SYN si intende che i due termini possono essere sostituiti senza modificare il concetto del mondo reale rappresentato. in.

Iperonimia si trasforma in una relazione BT (Broader-Term), relazione di generalizzazione.

Iponimia si trasforma in una relazione NT (Narrower-Term), relazione di specializzazione.

Olonimia è assunta come una relazione di tipo RT (Related-Term); cioè l'aggregazione è considerata un legame più debole rispetto alla specializzazione.

Meronimia è assunta come una relazione di tipo RT (Related-Term); La meronimia (contiene) è la relazione opposta alla olonimia (è contenuto).

Correlazione è assimilata ad una relazione di tipo RT (Related-Term); la correlazione è la relazione che lega 2 *synset* che condividono uno stesso iperonimo, cioè lo stesso padre.

2.4 La libreria per interagire con WordNet

Il pacchetto WordNet[1] oltre ai file di dati e di indice fornisce un programma per compiere semplici interrogazioni ed una libreria con i sorgenti in *C*.

[1]Reperibile all'indirizzo http://www.cogsci.princeton.edu/~wn/

SI Designer, invece, utilizza il linguaggio *Java*. Il problema della libreria fornita a corredo di WordNet è che presenta solo funzioni ad alto livello, cioè troppo flessibili per offrire un controllo preciso.

La scelta implementativa è stata quella di scrivere una libreria in Java. Questa libreria è molto versatile: ci permette di navigare tra i *synset*, oppure offre funzioni di alto livello che, a differenza di quelle di WordNet, riportano i dati in strutture dati native di Java.

In breve WordNet è organizzato in definizioni e (*entry*): una coppia $e = (f, m)$ dove f è la forma base e m (*meaning*) è il contatore del significato, esempio (`address`, `2`) si riferisce all'indirizzo presso il quale si può trovare una persona o un'organizzazione; mentre (`address`, `1`) si riferisce all'indirizzo di un computer in ambito informatico.

Alcuni esempi:

`Synset trovaSynset(String word, int senseNum)` dato una definizione e (forma base, contatore del significato) trova dai file di indice il *synset* y in cui è presente.

$$f_{trovaSynset} : e \longrightarrow y \qquad y : e \subset y$$

`Vector trovaRicorsivo(Synset inpynset, String tipo)`
a partire da un *synset* y trova tutti i *synset* [2] che si possono raggiungere con un puntatore[3] di tipo fissato, per esempio seguendo la gerarchia di specializzazione o aggregazione.

$$f_{trovaRicorsivo} : (y, r) \longrightarrow \{y_1, \ldots, y_n\} \qquad y_i : \langle y_i \, r \, y \rangle \quad \forall i = 1 \ldots n$$

Le prestazioni in definitiva sono buone: l'inefficienza per interpretare il bytecode Java viene controbilanciata da algoritmi con minore overhead.

2.4.1 L'algoritmo per trovare le relazioni tra termini

Per trovare le relazioni tra coppie di termini si è usato il seguente algoritmo:

1. Input.
   In ingresso si prende il vettore di termini (nomi di attributo e di interfaccia) e si suppone già definito il loro significato.

[2] Un insieme $\{y_1, \ldots, y_n\}$
[3] Che implica una relazione di tipo $r \in \{\text{SYN, BT, NT, RT}\}$

2. Creazione della struttura dati di lavoro.
   Viene creato un Hash di [synset, vettore dei termini sinonimi]. Per come è costruito Wordnet un *synset* può essere usato come chiave di un Hash senza ulteriori manipolazioni. Come dato associato alla chiave c'è un vettore contenente i termini trovati negli schemi che appartengono al *synset* chiave. In particolare:

   Per ogni termine
   viene trovato il *synset*
   si cerca nel Hash se e' gia' presente:
   se non c'e' si crea l'elemento del Hash
   se no si aggiunge al vettore corrispondente alla chiave

3. Sezione principale: trova le relazioni tra gli elementi.

   Per ogni elemento del Hash
   si scrivono le relazioni di sinonimia tra gli elementi associati alla chiave attuale
   per ogni elemento del vettore
   si trovano gli iperonimi
   per ogni iperonimo (che e' un *synset*) si cerca nel Hash se e' presente
   se c'e' si scrivono le relazioni di NT con tutti i termini del vettore
   si trovano gli iponimi e i termini correlati
   per ognuno di questi si cerca nel Hash se e' presente
   se c'e' si scrivono le relazioni di RT con tutti i termini del vettore

4. Eliminazione relazioni duplicate o inconsistenti.

   - Elimina le relazioni tra termini della stessa tabella.
   - Elimina relazioni duplicate, cioè con invertito il primo ed il secondo termine.

5. Output.
   Il risultato è un vettore di coppie di termini con la relazione che li lega.

2.5 Estrazione delle relazioni IntraSchema

L'ultimo modulo software che verrà descritto in questo capitolo si chiama SIM **S**ources **I**ntegrator **M**odule ed è stato realizzato dall'ing. Elisa Marri [22].
I compiti di SIM sono:

1. *estrazione automatica delle relazioni dagli schemi* ODL_{I^3}: vengono estratte, in modo automatico, tutte le relazioni intraschema che coinvolgono sorgenti ad oggetti, relazionali e semistrutturate;

2. *validazione delle relazioni*: in seguito all'intervento del progettista viene eseguita una fase di validazione delle relazioni fra attributi presenti nel Common Thesaurus, al fine di verificare la compatibilità dei domini coinvolti;

3. *inferenza di nuove relazioni*: partendo da tutte le relazioni già presenti nel Common Thesaurus (e validate), si deducono automaticamente nuove relazioni semantiche utilizzando le tecniche di inferenza messe a disposizione da ODB-Tools.

Dal punto di vista implementativo SIM è stato diviso in due sottomoduli:

- SIMA: si occupa della prima fase di estrazione automatica delle relazioni e del successivo popolamento del Thesaurus;

- SIMB: esegue la fase di validazione delle relazioni fra attributi presenti nel Thesaurus e la fase di inferenza di nuove relazioni.

Per quanto riguarda la realizzazione del Wrapper ci si è limitati all'utilizzo del sottomodulo SIMA, delegando al mediatore la validazione e l'inferenza di nuove relazioni.

Capitolo 3

Il processo di integrazione di sorgenti in MOMIS

In questo capitolo verrà illustrato un processo di integrazione eseguito con il sistema MOMIS usando come riferimento le sorgenti le cui descrizioni ODL$_{I^3}$sono contenute nell'appendice D.

3.1 Generazione del Common Thesaurus

Il Common Thesaurus del sistema MOMIS è una sorta di dizionario interno nel quale sono contenute relazioni terminologiche che legano tra loro classi ed attributi.

Queste relazioni sono di tipo intensionale ed esprimono la conoscenza inter-schema e intra-schema riguardo gli schemi delle sorgenti in esame; esse possono essere:

- SYN (SYNonym-of, *relazioni di sinonimia*): definita tra due termini che possono essere scambiati nelle sorgenti senza modificare il concetto del mondo reale rappresentato. Esempio: `faculty SYN faculty_name`.

- BT (Broader-Term, *relazioni di specializzazione)*: definita tra due termini t_i e t_j tali che t_i ha un significato più generale di t_j; NT (Narrower-Term) è la relazione opposta di BT. Esempio: `CS_Person BT Professor`, che è equivalente a `Professor NT CS_Person`.

- RT (Related-Term, *relazioni di aggregazione*): definita tra due termini t_i e t_j che sono generalmente usati nello stesso contesto e tra i quali esiste un legame generico (non meglio specificato). Esempio: `Research_Staff RT Department`.

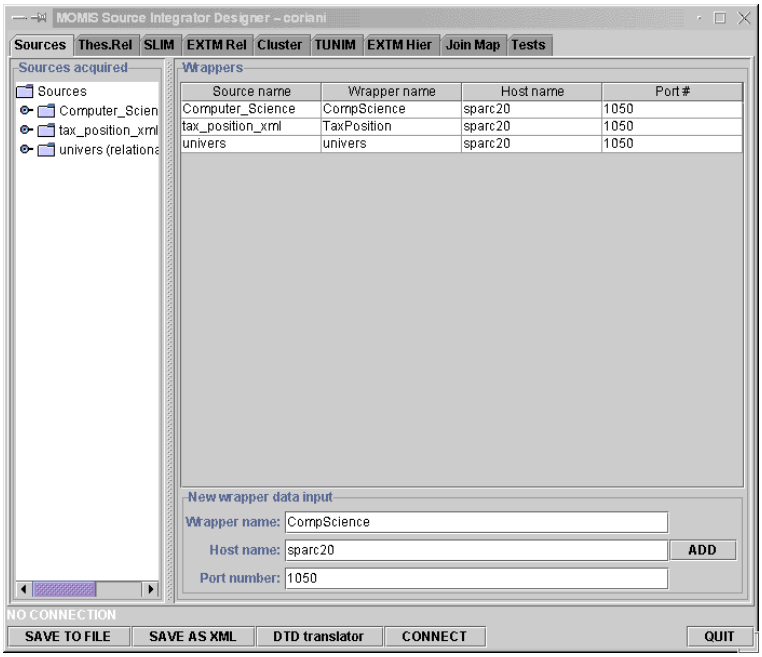

Figura 3.1: L'interfaccia grafica di MOMIS

La costruzione del Common Thesaurus è un processo che passa attraverso diverse fasi, durante le quali vengono aggiunte via via delle relazioni in quest'ordine:

1. *relazioni semantiche intra-schema*
2. *relazioni lessicali*
3. *relazioni aggiunte dal progettista*
4. *relazioni intensionali inferite*

In questa tesi sono stati creati due strumenti per automatizzare il più possibile il processo, purtroppo però l'intervento del progettista è ancora indispensabile.

3.2 Acquisizione delle sorgenti

La prima fase di integrazione consiste nell'acquisizione della descrizione ODL_{I^3} delle sorgenti. Il modulo preposto a tale compito è denominato SAM (Sources Acquisition Module) e la sua interfaccia grafica è mostrata in Figura 3.1.

Le informazioni ottenute dalle sorgenti vengono inserite, grazie al parser, come proprietà nella classe **GlobalSchemaProxy**.

Operativamente occorre indicare a SAM tutti i dati per raggiungere l'oggetto Wrapper cioè:

- Il **Nome** del Wrapper;
- l'**URL** della macchina che lo ospita;
- la porta di accesso.

Una volta inserite queste informazioni è sufficiente premere il tasto ADD per ottenere la descrizione ODL_{I^3} della sorgente.
Nel caso in esame i dati da inserire saranno:

- Prima sorgente:
  - Nome: univers;
  - Host: sparc20.ing.unimo.it;
  - Port: 1050;
- Prima sorgente:
  - Nome: TaxPosition;
  - Host: sparc20.ing.unimo.it;
  - Port: 1050;
- Prima sorgente:
  - Nome: Compu_Science;
  - Host: sparc20.ing.unimo.it;
  - Port: 1050;

Il modulo SAM esegue l'acquisizione secondo i seguenti passi:

1. *controllo dell'esistenza del wrapper*: il modulo richiede all'ORB 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.
2. *acquisizione dello schema ODL_{I^3}*: viene eseguito il *parsing* dello schema ODL_{I^3} fornito dal wrapper.

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 rileva che era stato acquisito lo schema di una sorgente avente lo stesso nome di quella descritta nello schema appena acquisito, viene richiesto al progettista di cambiare nome alla sorgente prima di mettere assieme le informazioni degli schemi.

Nel caso in cui si verifichi un malfunzionamento durante l'esecuzione di una di queste fasi il tool invia un messaggio d'errore. Dopo l'acquisizione di ogni schema, il modulo provvede all'aggiornamento dell'interfaccia grafica.

3.3 Acquisizione delle relazioni intra-schema

In questa fase si utilizza una delle nuove funzionalità introdotte nel wrapper da questa tesi: L'esportazione delle relazioni intra-schema. Col termine relazioni intra-schema si intendono tutte quelle relazioni che possono essere ricavate dalla struttura della fonte di dati. In precedenza l'estrazione delle suddette relazioni veniva fatta in questa fase ma, grazie ad una politica di spostamento verso il basso (quindi verso i Wrapper) del maggior numero di servizi possibile, ora SI-Designer si limita ad importare questi dati e ad inserirli nel *Common Thesaurus*. L'unico onere rimasto al progettista è quello di validare le relazioni importate, eseguire quindi il modulo SIMB.
L'importazione avviene premendo il pulsante "LoadRelations" nella toolbar della finestra di figura 3.2. Facendo riferimento all'esempio le relazioni importate sono: (si tenga presente che non tutte le sorgenti hanno a disposizione il modulo di esportazione delle relazioni, quindi solo per univers le relazioni vengono importate, per le altre sorgenti sono ricavare da SI_Designer)

⟨univers.COURSE RT univers.RESEARCH_STAFF⟩
⟨univers.DEPARTMENT RT univers.RESEARCH_STAFF⟩
⟨univers.SCHOOL_MEMBER RT univers.PERSON⟩
⟨univers.ROOM RT univers.COURSE⟩
⟨Computer_Science.Student NT Computer_Science.CS_Person⟩
⟨Computer_Science.Professor NT Computer_Science.CS_Person⟩
⟨tax_position_xml.Student RT tax_position_xml.ListOfStude⟩
⟨Computer_Science.Course RT Computer_Science.Student⟩
⟨Computer_Science.Office RT Computer_Science.Professor⟩
⟨Computer_Science.Location RT Computer_Science.Office⟩
⟨Computer_Science.Professor RT Computer_Science.Course⟩

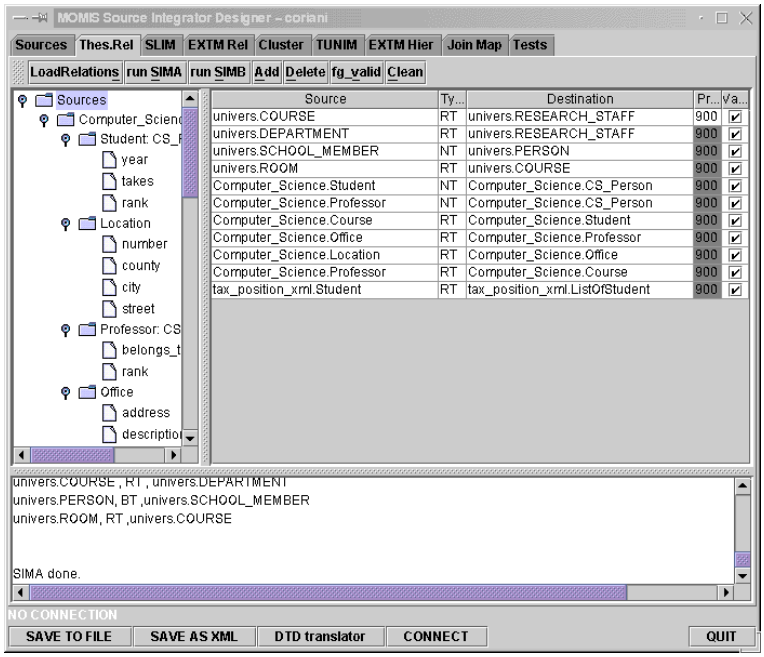

Figura 3.2: Modulo SIM: Acquisizione delle relazioni intra-schema

3.4 Estrazione delle relazioni inter-schema

Le relazioni terminologiche inter-schema sono estratte analizzando la totalità degli schemi ODL$_{I^3}$. La loro estrazione è basata sulle relazioni lessicali che sussistono tra nomi di classi ed attributi, derivanti dai significati delle parole usate: un tipo di conoscenza non esplicitata tramite costrutti di un linguaggio di definizione dei dati. Il modulo preposto ad attuare questa fase si chiama SLIM ed è stato realizzato dall'ing. Malvezzi. Anche in questo caso interviene una nuova funzionalità introdotta nel Wrapper, infatti in precedenza era il progettista ad attribuire nomi descrittivi ai vari elementi della descrizione; per cui c'era un'incertezza di interpretazione insita nell'ambiguità del linguaggio. Grazie all'utilizzo del nuovo Wrapper, invece, non è più il progettista ad assegnare i nomi descrittivi, bensì colui che intende esportare la sorgente, eliminando così ogni possibile ambiguità linguistica. Nonostante ciò è comunque indispensabile l'utilizzo del database lessicale WordNet, citato nel capitolo precedente 2.3, il significato infatti viene trasmesso dal wrapper come un oggetto con due proprietá : formaBase e Significati, spetta quindi al modulo SLIM associare a questi oggetti i rispettivi elementi delle sorgenti. Rimane comunque la possibilità per il progettista di modificare le scelte fatte dall'implementatore nel caso queste ultime risultino scorrette o imprecise. L'annotazione viene importata premendo il bottone "GetAnnotation" nella toolbar, mentre la modifica avviene in due fasi:

1. **Scelta della forma base.** In tale scelta il progettista è assistito dal sistema che gli propone la forma base (word form) usando il processore morfologico presente in WordNet. Per forma base si intende la parola tolti suffissi dovuti alla declinazione o coniugazione.

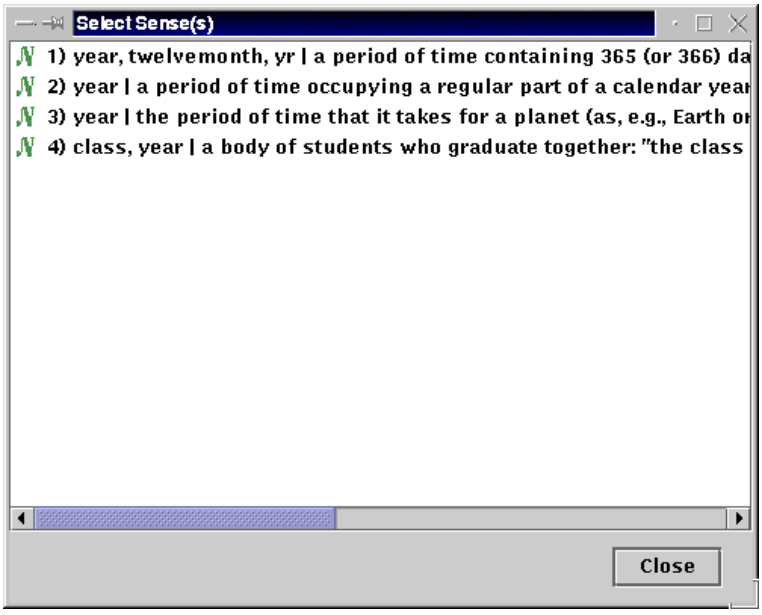

Figura 3.3: Significati di YEAR

2. **Scelta del significato.** Il progettista può decidere di far corrispondere ad un nome zero, uno o più significati. Ad esempio in Figura 3.3 per la forma base YEAR ottengo tutti i 4 significati che WordNet le attribuisce tra cui scegliere quello calzante con il contesto.

   Alla fine delle procedure di acquisizione dei significati, il progettista, premendo il pulsante "build", procede al calcolo delle relazioni interschema (vedi Figura 3.4). Le relazioni derivanti da WordNet vengono proposte come relazioni semantiche da inserire nel Common Thesaurus in base alla seguente corrispondenza:

   Sinonimia: corrisponde ad una relazione SYN.

   Iperonimia: corrisponde ad una relazione BT.

   Olonimia: corrisponde ad una relazione RT.

   Correlazione: corrisponde ad una relazione RT.

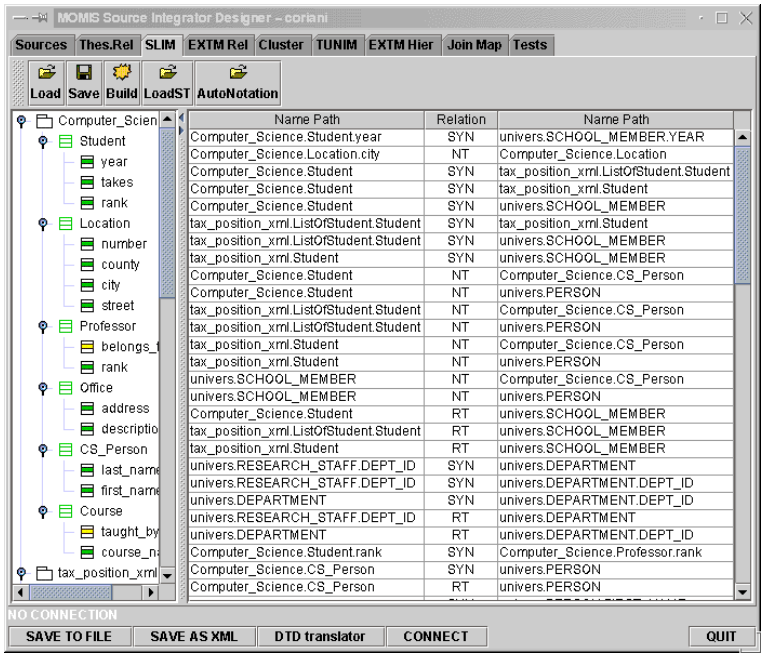

Figura 3.4: Relazioni intensionali ottenute dopo SLIM

Le relazioni così ottenute vengono poi esaminate dal progettista il quale può scartare quelle sbagliate o fuorvianti; tutte le altre vengono accettate ed immesse nel *Common Thesaurus* come relazioni semantiche intensionali.

3.4.1 Aggiunta di nuove relazioni

Il progettista, nel caso non siano state rilevate, può aggiungere “manualmente” nuove relazioni. Queste possono essere sia relazioni intensionali, che assiomi estensionali. Da ogni assioma estensionale si ottiene una relazione intensionale da inserire nel Common Thesaurus.
L’operazione di inserimento manuale è molto delicata in quanto nuove relazioni vengono forzate ad entrare nello schema globale, ogni errore quindi può condurre alla generazione di un GlobalSchema errato.
Per evitare l’insorgere di questo tipo di problemi si passa per le fasi di validazione delle relazioni tra attributi e tra classi.

Le relazioni tra attributi inserite nei passi precedenti devono essere analizzate per verificare la compatibilità dei loro domini; tale fase è detta *validazione* e viene svolta dal modulo SIMB. Le regole su cui si basa questa procedura sono:

- ogni relazione di sinonimia è validata se i domini dei due attributi coinvolti sono equivalenti, o se uno dei due è più specializzato dell'altro;

- ogni relazione di specializzazione è validata se i domini dei due attributi sono equivalenti, oppure se il dominio dell'attributo più generale comprende il dominio dell'altro.

3.4.2 Inferenza di nuove relazioni intensionali

Dopo la validazione si ha uno schema virtuale contenente tutta la conoscenza semantica ottenuta fino a questo momento.
Questo schema verrà inviato a ODB-TOOLS il quale eseguirà la fase di validazione tra classi ed inferirà nuove relazioni da inserire nel Common Thesaurus.

Le relazioni inferite sono inter-schema, cioè definite tra classi di sorgenti diverse.

3.5 Analisi di Affinità delle classi

In questa fase vengono analizzati gli schemi ODL$_{I^3}$delle diverse sorgenti che partecipano allo schema globale allo scopo di individuare le classi che descrivono le stesse informazioni, o comunqe informazioni semanticamente equivalenti, e che appartengono a sorgenti diverse. Le classi ODL$_{I^3}$ vengono analizzate e confrontate sulla base dei *coefficienti di affinità*, che permettono di determinare il loro livello di *similarità*. Per quanto riguarda le classi vengono analizzate le relazioni che esistono tra i loro nomi (attraverso il *Name Affinity Coefficient*) e tra i loro attributi (per mezzo dello *Structural affinity Coefficient*), consentendo così l'ottenimento di un valore globale denominato *Global Affinity Coefficient.*

Per il calcolo dei coefficienti di affinità il Thesaurus viene organizzato in una struttura simile alle Associative Networks [23], in cui i nodi (che possono rappresentare o il nome di una classe o quello di un attributo)sono uniti attraverso relazioni terminologiche. Le relazioni presenti nel network sono percorribili in entrambi i sensi (dunque anche le BT e NT)e si dice che due termini sono affini quando esiste un percorso che li unisce, formato da relazioni. Per dare una valutazione numerica della affinità tra due termini, a ogni tipo di relazione viene associato un peso (denominato *strength* e denotato da $\sigma_{\Re}$), che sarà tanto maggiore quanto più questo tipo di relazione contribuisce a legare due termini (sarà quindi $\sigma_{syn} \geq \sigma_{bt/nt} \geq \sigma_{rt}$). In questa sezione si userà $\sigma_{ij_{\Re}}$ per denotare il peso della relazione terminologica $\Re$ definita tra i termini t_i e t_j. Nel nostro esempio,

e nelle sperimentazioni precedentemente realizzate presso l'Università di Milano, si è adottato $\sigma_{syn} = 1$, $\sigma_{bt} = \sigma_{nt} = 0.8$ e $\sigma_{rt} = 0.5$.

Definizione 1 (Funzione di Affinità) Presi due termini, t_i e t_j, possono essere presenti nel Thesaurus zero o più cammini che li uniscono, formati da relazioni. Ad ognuno di questi cammini corrisponde naturalmente un valore, dato dal prodotto dei pesi delle relazioni in esso coinvolte. La Funzione di Affinità $A_{thes}(t_i, t_j)$ tra due termini, t_i e t_j, restituisce il valore maggiore tra questi, corrispondente al cammino più *stringente*, che unisce questi termini (che non sempre coincide col cammino più breve), definito come segue:

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

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

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

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 dei Affinità restituisce un valore maggiore o uguale ad un predefinito valore di soglia $\alpha > 0$, cioè:

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

3.5.1 Coefficienti di Affinità

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

Definizione 3 (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:

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

Definizione 4 (Structural Affinity coefficient) Lo Structural Affinity Coefficient di due classi c e c', scritto $SA(c,c')$, è la misura dell'affinità dei loro attributi, calcolata come segue:

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

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

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

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

Lo Structural Affinity Coefficient è valutato utilizzando la funzione di Dice, raffinato da un fattore di controllo F_c, e restituisce un valore compreso nell'intervallo [0,1] proporzionale al numero di attributi *affini* tra le classi considerate.
Il termine F_c realizza un controllo sui domini degli attributi coinvolti nella relazione da esaminare, permettendo quindi di non limitare la computazione di questo coefficiente alla sola analisi dei nomi che identificano gli attributi (analisi terminologica) e di estenderla alla considerazione dei domini che caratterizzano questi attributi. In pratica, una relazione che coinvolge attributi viene pesata in modo maggiore o minore nel calcolo del coefficiente a seconda che questa relazione trovi o meno riscontro anche nei tipi dei domini, e non solo nei nomi degli attributi. Il termine F_c va quindi a rifinire il coefficiente SA, moltiplicando la prima parte di questo per un termine compreso tra 0 e 1: in particolare, F_c è il rapporto tra numero di relazioni validabili memorizzate nel Thesaurus tra attributi delle due classi, e numero di queste relazioni che sono state validate.
In questo modo, maggiore sarà il numero di attributi affini tra le due classi, e maggiore il numero di controlli positivi, più alto risulterà il valore dello *Structural Affinity Coefficient*.

Definizione 5 (Global Affinity Coefficient) Il Global Affinity Coefficient di due classi c e c', denotato da $GA(c,c')$, è la misura della loro affinità calcolata come la somma pesata degli *Name Affinity Coefficient* e *Structural Affinity Coefficient*:

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

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

3.6 Generazione dei Cluster

In questa fase, grazie all'utilizzo di tecniche di clustering, si identificano gli insiemi di classi affini negli schemi considerati. Le classi quindi vengono automaticamente classificate in gruppi caratterizzati da differenti livelli di affinità arrivando alla costruzione di un albero.

Per la generalizzazione dei cluster, ARTEMIS utilizza tecniche attraverso le quali le classi sono automaticamente classificate in una struttura ad albero dove:

- le foglie rappresentano tutte le classi locali: foglie contigue sono classi caratterizzate da alta affinità, mentre foglie tra loro molto lontane rappresenteranno invece classi a bassa affinità;
- ogni nodo rappresenta un livello di clusterizzazione ed ha associato il coefficiente di affinità tra i due sottoalberi (cluster) che unisce.

Il risultato di questa operazione è riportato in figura 3.5 in cui notiamo anche la presenza di sei tasti:

- *Rename Class*: permette di associare al cluster un nome significativo (che sarà poi il nome della classe globale ad esso associata);
- *Delete Mapping*: permette di cancellare tutti i cluster correnti;
- *Delete Class*: permette di cancellare un cluster;
- *Add new Class*: rende possibile l'inserimento di un nuovo cluster;

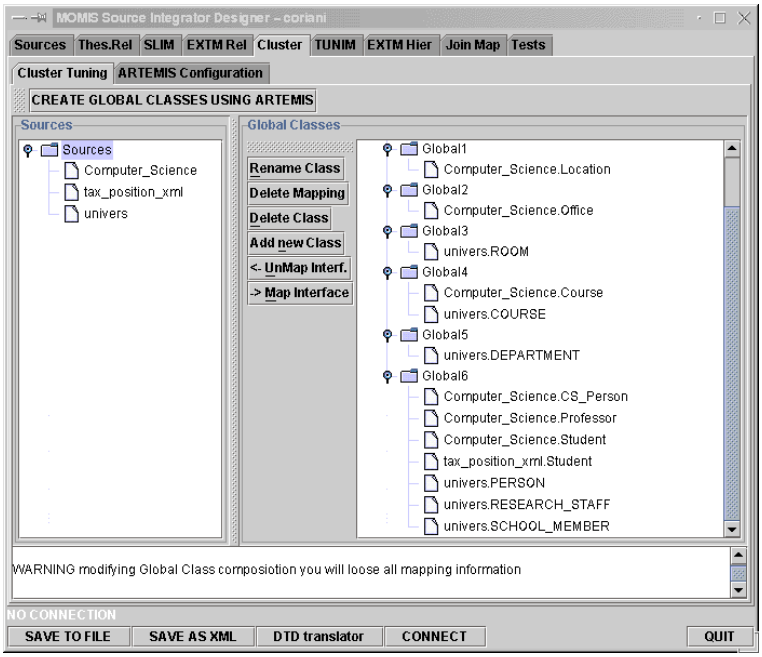

Figura 3.5: Pannello di visualizzazione dei cluster

- *UnMap Interf*: effettua la rimozione di una classe selezionata da un determinato cluster;
- *Map Interface*: permette di inserire una classe selezionata all'interno di un cluster.

Prima di avviare il procedimento di calcolo dei cluster è necessario inserire il peso associato ad ogni tipo di relazione e le soglie predefinite; a tale scopo si utilizza il pannello denominato "ARTEMIS Configuration". I cluster ottenuti nel nostro esempio sono mostrati in figura 3.5.

3.7 Generazione degli attributi globali e delle mapping-table

Per ogni cluster individuato viene creata una classe globale, la quale è caratterizzata da:

- un nome;
- un insieme di attributi globali;

- una mapping table che indica la corrispondenza tra gli attributi globali e quelli contenuti nelle sorgenti.

Sia Cl_i un cluster, definiamo *insieme unione* l'unione di tutti gli attributi delle classi locali appartenenti al cluster. L'insieme unione sarà dato da:

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

Dove c_j è una classe locale appartenente al cluster e $A(Cl_i)$ è l'insieme degli attributi appartenenti ad esso.

Alla creazione dell'insieme-unione seguono due fasi:

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

Nella fase di fusione degli attributi "simili", SI_Designer tenta di eliminare le ridondanze considerando le relazioni terminologiche del *Common Thesaurus*. Il procedimento di fusione è sempre automatico per gli attributi legati da relazioni validate, mentre lo è solo in certi casi se gli attributi sono legati da relazioni non validate. In particolare SI_Designer opera in questo modo:

- **Attributi in relazioni validate.** Per questi attributi la fusione è sempre automatica:
  - Agli attributi legati da relazioni `SYN` SI_Designer farà corrispondere un unico attributo globale: il dominio è lo stesso ed il nome può essere scelto dal progettista tra le proposte di SI_Designer, oppure inserito esplicitamente.
  - Gli attributi legati da relazioni `BT` vengono trattati da SI_Designer sostituendoli con un attributo globale che ha lo stesso nome e lo stesso dominio dell'attributo generalizzazione.
- **Attributi in relazioni non validate.** A questa categoria appartengono gli attributi delle relazioni del *Common Thesaurus* che non hanno superato la validazione: SI_Designer è in grado di individuare un attributo globale in modo automatico solo per un numero limitato di casi, lasciando al progettista il compito di aggiungere altri attributi globali per completare l'integrazione. In particolare, l'individuazione automatica di un attributo globale, in presenza di relazioni non validate, è possibile se gli attributi nelle relazioni soddisfano i seguenti requisiti:
  1. sono legati da relazioni `SYN` o `BT`;

2. le classi in relazione appartengono ad uno stesso cluster;
3. rappresentano gerarchie di aggregazione (sono attributi complessi o foreign key);

Il progettista può a questo punto intervenire per ampliare l'insieme degli attributi globali per meglio rappresentare le informazioni contenute nelle sorgenti locali.

Contemporaneamente alla creazione degli attributi globali, SI_Designer costruisce una *mapping-table*. Essa é una tabella $MT[CL][AG]$ dove CL é l'insieme delle classi locali che appartengono al cluster cui la mapping-table si riferisce e AG é l'insieme degli attributi globali creato da SI_Designer. Indicando con C il nome di una classe locale, con A il nome di un attributo globale e con AL il nome di un attributo locale, ogni elemento $MT[C][A]$ della tabella può assumere i seguenti valori:

- AL, con $AL \in C$.
  Questo valore viene inserito quando:
  - l'attributo globale A deve rappresentare l'informazione contenuta nel solo attributo locale AL.
  - ci sono relazioni di specializzazione che legano tra loro attributi appartenenti a classi diverse.

- AL_1 ***and*** AL_2 ***and ... and*** AL_n, con $AL_i \in C, i = 1, ..., n$.
  Usato quando il valore dell'attributo A é il concatenamento dei valori di piú attributi appartenenti alla medesima classe locale C.

- ***case of*** AL $\quad cost_1$**:** AL_1 $\quad cost_2$**:** AL_2 $\quad \ldots \quad cost_n$**:** AL_n
  dove $AL, AL_i \in C, i = 1, ..., n$ e $cost_i, i = 1, ..., n$ sono delle costanti.
  Questa situazione avviene quando l'attributo globale A può assumere il valore di uno tra un insieme di attributi locali $\{AL_i\}$ appartenenti alla medesima classe e la scelta avviene attraverso un terzo attributo locale AL, appartenente sempre alla stessa classe locale, che funge da selettore.

- $costante$.
  Si contempla il caso in cui l'attributo globale A non corrisponde ad alcun attributo della classe locale C. Il valore assunto da A viene attribuito dal progettista in base al significato dato all'attributo globale.

- $null$.
  Questo é il caso in cui l'attributo globale A, durante un accesso alla classe locale C, non assume alcun valore.

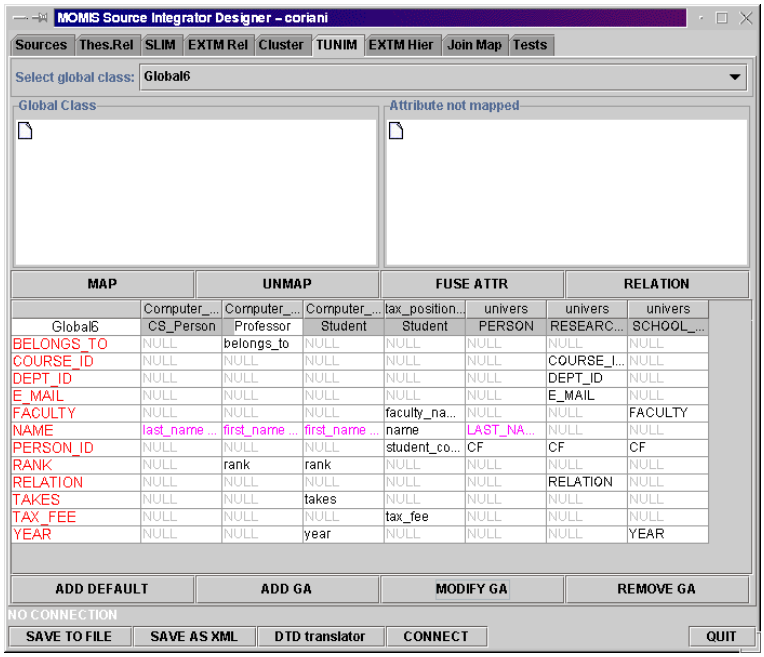

Figura 3.6: Modulo TUNIM per la generazione delle Mapping-Table

SI_Designer crea una *mapping-table* per ogni classe globale e mette a disposizione del progettista un' interfaccia che permette sia di avere una visione completa di tutte le classi globali (nomi ed attributi), che l'inserimento dei nomi delle classi globali e la modifica delle *mapping-table*. Al termine della fase di TUNIM, nel nostro esempio, vengono generate sei classi globali, le classi di maggior interesse sono riportate nelle figure 3.6 e 3.7.

3.7.1 Considerazioni

Nel caso si renda necessaria, una volta realizzata l'integrazione, l'aggiunta di una nuova sorgente occorre , dopo averla acquisita ed effettuate le procedure di importazione delle relazioni intra-schema e dei significati, ripetere le ultime due fasi del processo. Questa necessità è dettata dal fatto che l'aggiunta di nuove relazioni nel theraurus comune può modificare in modo sostanziale i cluster e la struttura delle mapping-table.

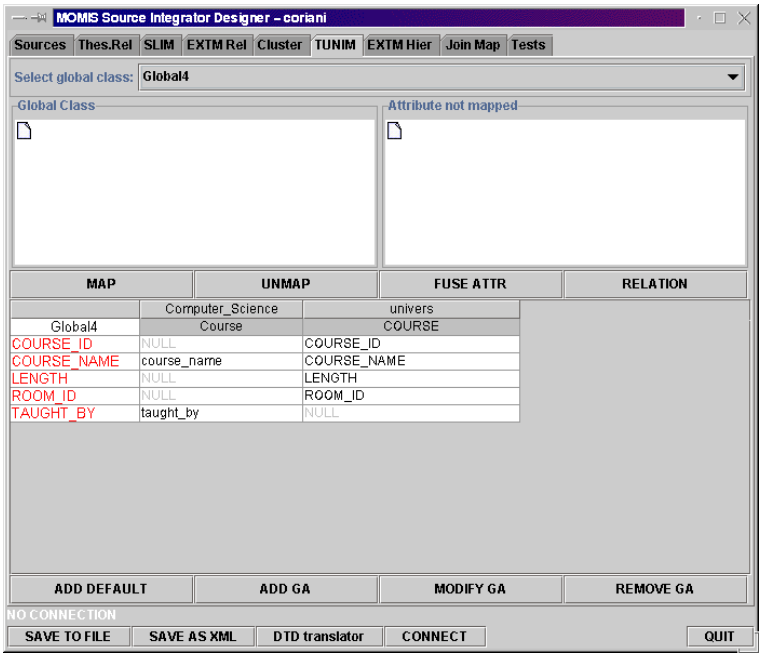

Figura 3.7: Modulo TUNIM per la generazione delle Mapping-Table

Capitolo 4

Il Wrapper JDBC

I wrapper rappresentano le interfacce tra le sorgenti dati e l'integratore, il loro compito è quello di presidiare le sorgenti a loro assegnate e fornire alcuni servizi al mediatore.
Oltre a consentire l'accesso ai dati da parte dell'applicazione client, nel caso specifico al Mediator di MOMIS, i servizi specifici di base che ogni wrapper deve essere in grado di fornire sono due:

- fornire la descrizione ODL_{I^3}della sorgente alla quale è connesso, tradurre quindi la struttura della base di dati in un linguaggio comprensibile al mediatore;
- consentire l'esecuzione delle query generate dal Query Manager.

Nella presente tesi sono state introdotte due nuove funzionalità, cui si è fatto cenno nel capitolo precedente 3.3, 3.4:

1. esportazione dei significati;
2. esportazione delle relazioni intra-schema.

L'introduzione di questi due nuovi servizi consente al wrapper di fornire molte più informazioni al mediatore, sia dal punto di vista strutturale si da quello lessicale, eliminando di fatto tutte quelle "indecisioni" derivanti dalle ambiguità linguistiche. Nel seguito del capitolo verranno analizzate la struttura e le modalità di funzionamento di un wrapper per sorgenti relazionali interfacciabili tramite JDBC. Il codice è stato implementato interamente in linguaggio Java ed è contenuto nella classe Wrapper_server.

4.1 L'esempio di riferimento

Nell'esempio di riferimento si cerca di modellare una semplice realtà universitaria; si tratta di un database relazionale realizzato con IBM DB2. Lo schema, pur essendo volutamente molto semplice, è completo di tutte le particolarità necessarie per mostrare il funzionamento del wrapper. Lo schema E/R è riportato in figura 4.1 ed è composto da sei tabelle: Person, Research-Staff, School-Member, Department, Course e Room. Diamo un breve descrizione dello schema:

Nell'entità persona (tabella PERSON) vengono memorizzati il Nome, il Cognome e il Codice Fiscale che è anche chiave primaria. Per ogni professore (rappresentato nell'entità Research-Staff subset di PERSON) sono memorizzati in più l'e-mail, che è anche chiave candidata, e il campo Relation. Gli studenti (entità School-Member anch'essa subset di PERSON) hanno come attributi supplementari Faculty(Facoltà) e Year(Anno di corso). Nella tabella Department vengono elencati, per ciascun dipartimento, il nome (Dept-Name), il codice (Dept-id, chiave primaria), il budget a disposizione e l'area d'appartenenza (scientifica, umanistica,...). I corsi relativi ad ogni facoltà sono memorizzati nella tabella Course e le loro caratteristiche sono: il nome (Course-Name), il codice (Course-id, chiave primaria) e la durata (length). Infine si ha l'entità Room nella quale si trovano i dati relativi alle aule: numero di posti (Seats-Number), indentificativo dell'aula (Room-id, chiave primaria) e note varie (Notes).

Per quanto riguarda invece le relazioni:

- Research-Staff e School-Member si presentano come subset dell'entità Person;
- Ogni componente dell'entità Research-Staff afferisce ad uno ed un solo dipartimento, mentre ad ogni dipartimento afferiranno uno o più componenti di Research-Staff.
- Ogni corso dovrà essere tenuto da almeno un componente di Research-Staff e ognuno dei suddetti componenti potrà tenere un solo corso.
- Ciascun corso si terrà in una sola aula, mentre ogni aula potrà ospitare un numero N di corsi.

4.2 Struttura del Wrapper

L'obiettivo principale del wrapper è quello di mettere in comunicazione due entità: la sorgente dati e il mediatore. Una delle problematiche principali da affrontare è quindi quella delle connessioni con i vari elementi.
L'approccio adottato prevede due modalità di collegamento differente:

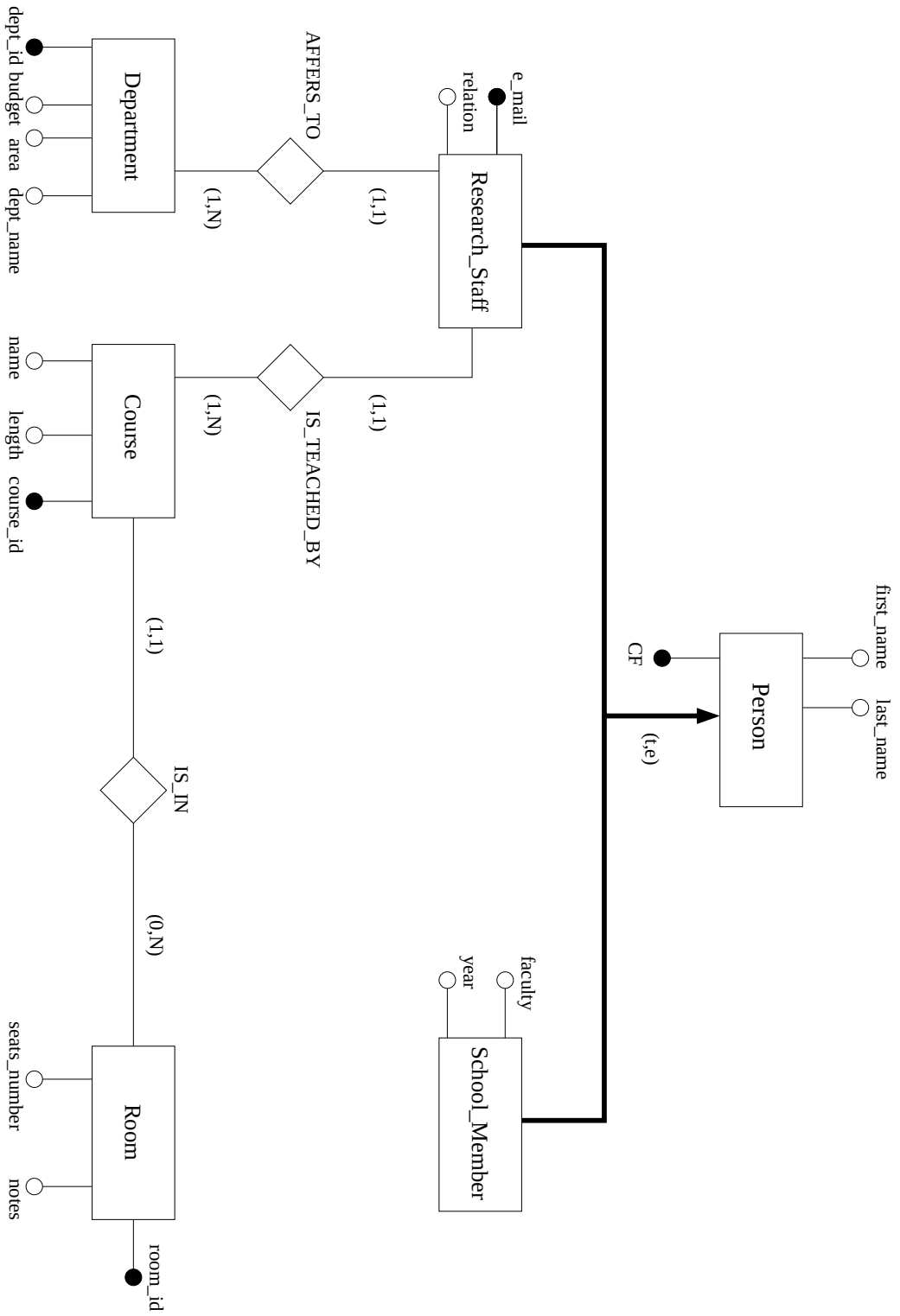

Figura 4.1: Schema ER dell'esempio di riferimento

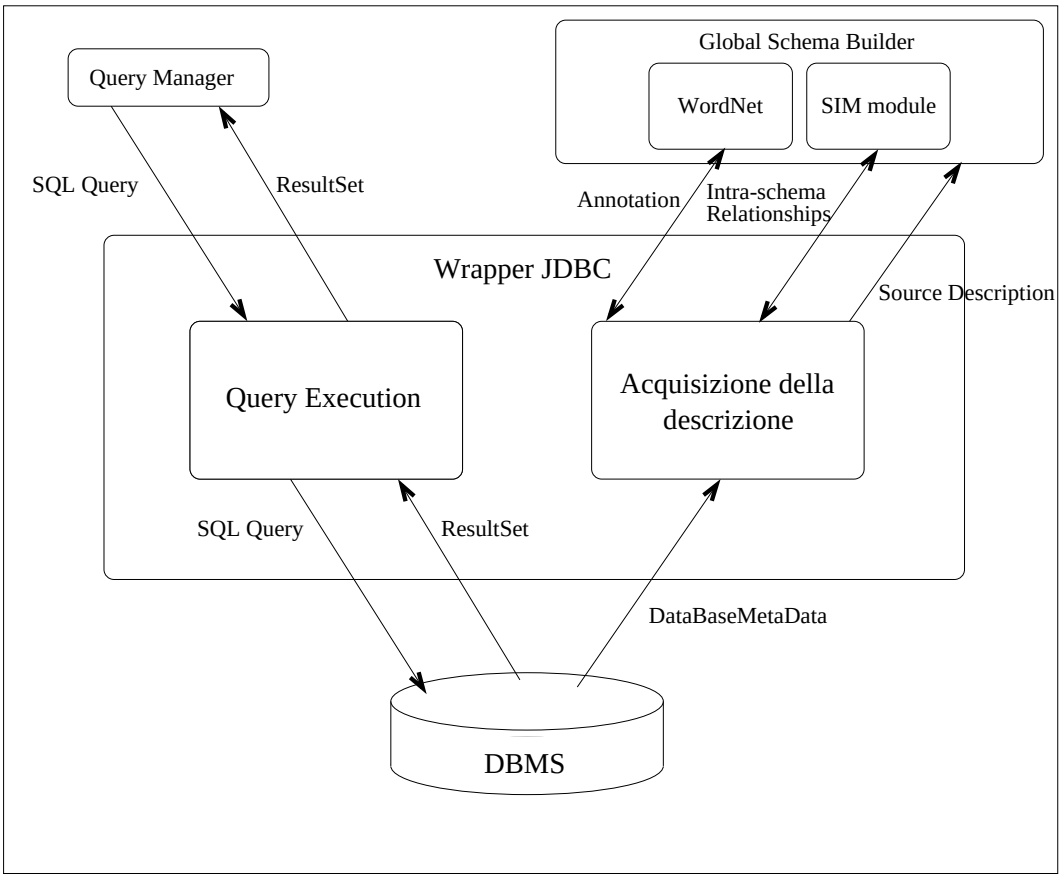

Figura 4.2: Struttura del Wrapper

1. JDBC per realizzare il collegamento con la fonte dati da integrare;
2. CORBA per permettere a MOMIS di invocare i metodi della classe Wrapper_server.

Strutturalmente il wrapper è suddiviso, come del resto anche MOMIS, in due parti che si occupano delle operazioni principali(vedi figura 4.2): acquisizione di dati relativi alla struttura della sorgente ed esecuzione di Query.

4.2.1 Traduzione dello schema sorgente in ODL$_{I^3}$

Uno dei principali compiti del wrapper è quello di interpretare la struttura della base di dati ed esprimerla nel linguaggio descrittivo ODL$_{I^3}$. Facendo riferimento all'appendice B e all'esempio di riferimento 4.1 si darà una descrizione delle regole

di traduzione utilizzate per trasporre in ODL$_{I^3}$una qualunque struttura relazionale. Si riporta qui di seguito la descrizione ODL$_{I^3}$dell'esempio di riferimento:

– begin description

```
// —— [COURSE]
interface COURSE(source relational univers
//[Primary Keys]
key(COURSE_ID)
//[Foreign Keys]
foreign_key (ROOM_ID) references ROOM)
{ attribute long /* INTEGER */ COURSE_ID;
attribute string /* VARCHAR */ COURSE_NAME;
attribute long /* INTEGER */ LENGTH;
attribute long /* INTEGER */ ROOM_ID;
};

// —— [DEPARTMENT]
interface DEPARTMENT(source relational univers
//[Primary Keys]
key(DEPT_ID))
{ attribute long /* INTEGER */ DEPT_ID;
attribute string /* VARCHAR */ DEPT_NAME;
attribute string /* VARCHAR */ BUDGET;
attribute string /* VARCHAR */ AREA;
};

// —— [PERSON]
interface PERSON(source relational univers
//[Primary Keys]
key(CF))
{ attribute string /* VARCHAR */ CF;
attribute string /* VARCHAR */ FIRST_NAME;
attribute string /* VARCHAR */ LAST_NAME;
};

// —— [RESEARCH_STAFF]
interface RESEARCH_STAFF(source relational univers
//[Primary Keys]
key(CF)
candidate_key SQL010619165155800 (E_MAIL)
//[Foreign Keys]
foreign_key (COURSE_ID) references COURSE
```

```
foreign_key (DEPT_ID) references DEPARTMENT)
{ attribute string /* VARCHAR */ CF;
attribute string /* VARCHAR */ RELATION;
attribute string /* VARCHAR */ E_MAIL;
attribute long /* INTEGER */ DEPT_ID;
attribute long /* INTEGER */ COURSE_ID;
};


    // —— [ROOM]
interface ROOM(source relational univers
//[Primary Keys]
key(ROOM_ID))
{ attribute long /* INTEGER */ ROOM_ID;
attribute long /* INTEGER */ SEATS_NUMBER;
attribute string /* VARCHAR */ NOTES;
};


    // —— [SCHOOL_MEMBER]
interface SCHOOL_MEMBER(source relational univers
//[Primary Keys]
key(CF)
//[Foreign Keys]
foreign_key (CF) references PERSON)
{ attribute string /* VARCHAR */ CF;
attribute string /* VARCHAR */ FACULTY;
attribute long /* INTEGER */ YEAR;
};


    – end description
```

Si può notare molto chiaramente come alle entità dello schema relazionale siano state associate interfacce recanti il medesimo nome, vediamo inoltre che gli attributi delle interfacce corrispondono ai campi delle relative tabelle nello schema relazionale. A scopo riassuntivo si propone la seguente tabella contenente il mapping tra gli elementi della sorgente e la loro descrizione ODL_{I^3}:

Mapping tra i componenti dello schema relazionale e loro descrizione ODL_{I^3}

ODL_{I^3}	Componenti dello schema relazionale
Tabella	Interface
Primary key	key
foreign key	foreign_key
alternate key	Candidate_key
Campo della tabella	Attribute

A questo punto, però, sorge un problema, i tipi di dato Java.sql.Types.* non sono supportati da ODL_{I^3}, è necessario dunque convertirli nei tipi corrispondenti. Le modalità di conversione sono riportate nella tabella seguente:

Mapping tra i tipi ODL_{I^3} e quelli Java.sql.Types.*

Tipi ODL_{I^3}	Tipi Java.sql.Types.*
Long	BIGINT
Char	CHAR
Long	DATE
Long	DECIMAL
Double	DOUBLE
Float	FLOAT
Long	INTEGER
Long	NUMERIC
Double	REAL
Long	SMALLINT
Long	TIME
Long	TIMESTAMP
Long	TINYINT
String	VARBINARY
String	VARCHAR

Si può notare , inoltre, che all'inizio della dichiarazione d'interfaccia sono presenti anche il nome della sorgente(nel caso in esame "univers"), il tipo di sorgente da integrare("relational"), key che rappresenta la chiave primaria della tabella, foreign_key e candidate_key. Per quanto riguarda la dichiarazione di foreign_key, la sintassi ODL_{I^3} permette di menzionare solamente l'entità a cui si fa riferimento e non all'attributo specifico.

4.2.2 Il collegamento con la sorgente

Il collegamento con la sorgente è stato realizzato grazie alla tecnologia JDBC. Come detto in precedenza (vedi capitolo 2) per utilizzare la suddetta tecnologia, è

indispensabile caricare il driver relativo alla sorgente da collegare, nel nostro caso db2 [1], ma il driver sopracitato non si è rivelato adatto all'utilizzo per MOMIS, in quanto rende difficoltoso l'accesso alla base di dati ad utenti diversi dal proprietario del database stesso.
Per ovviare al problema si e' scelto di usare RmiJdbc [2], un driver che, ponendosi come intermediario tra il driver standard e il wrapper, risolve questo problema.
Le informazioni necessarie alla formulazione della descrizione ODL_{I^3} vengono reperite mediante l'utilizzo di metodi delle interfacce *DataBaseMetaData* e *ResultSetMetaData*. In particolare:

- *getColumnCount()*:(ResultSetMetaData) restituisce il numero di colonne presenti in una tabella;
- *getColumnName(int column)*:(ResultSetMetaData) fornisce i nomi delle colonne della tabella;
- *getCoumnType(int column)*:(ResultSetMetaData) ritorna il tipo di dato contenuto in ciascuna colonna;
- *getPrimaryKeys(String catalog, String schema, String table)*:(DataBaseMetaData) ritorna un ResulSet contenente informazioni riguardanti la chiave primaria della tabella;
- *getImportedKeys*:(DataBaseMetaData) restituisce un ResultSet con i dati relativi alle foreign keys della tabella;
- *getIndexInfo*:(DataBaseMetaData) risponde all'invocazione con un ResultSet il cui contenuto è relativo alle candidate keys.

Per quanto riguarda l'esecuzione di query viene fatta nel modo classico dalla classe WMomisResultSet:

```
_stmt = _connection.createStatement();

_rs = _stmt.executeQuery( _oql );
```

I dati così ottenuti vengono poi trasferiti a SI_Designer mediante l'utilizzo di CORBA.

[1] COM.ibm.db2.jdbc.app.DB2Driver
[2] reperibile al sito: http://www.objectweb.org/RmiJdbc/

4.2.3 La comunicazione tra Wrapper e Mediator

La comunicazione tra questi due elementi avviene attraverso un'interfaccia CORBA chiamata "Wrapper", la quale permette di esportare alcuni dei metodi della classe Wrapper-server.
I metodi esportati sono:

- string getType() ritorna una stringa contenente il tipo di sorgente in esame(relazionale, ...);
- string getDescription() restituisce la descrizione ODL_{I^3}della sorgente presidiata.
- MomisResultSet runQuery(in string oql), questo metodo permette l'esecuzione di query sulla fonte di dati. La query viene inviata come parametro sotto forma di stringa mentre la risposta viene data sotto forma di un'interfaccia CORBA chiamata MomisResultSet di cui si riporta la struttura: interface MomisResultSet {

  long getColumnCount() raises (momisOqlException);

  string getColumnName(in long column) raises (momisOqlException);

  long getColumnType(in long column) raises (momisOqlException);

  boolean next() raises (momisOqlException);

  void close() raises (momisOqlException);

  long getColumnByName(in string column) raises (momisOqlException);

  long getLong(in long column) raises (momisOqlException);

  char getChar(in long column) raises (momisOqlException);

  double getDouble(in long column) raises (momisOqlException);

  float getFloat(in long column) raises (momisOqlException);

  string getString(in long column) raises (momisOqlException);

  string stringSet();

  /* Unsupported Type */
  const long TYPE-Unsupported = -1;
  /* Long Type */
  const long TYPE-Long = 1;
  /* Char Type */
  const long TYPE-Char = 2;
  /* Double Type */
  const long TYPE-Double = 3;

/* Float Type */
const long TYPE-Float = 4;
/* String Type */
const long TYPE-String = 5;
};

- string getSourceName() fornisce il nome del data base;
- SlimCarrier getAnnotation() restituisce l'annotazione lessicale dei componenti del database che si trova, sotto forma di file, nella directory in cui è contenuta la classe Wrapper-server. SlimCarrier è un'interfaccia CORBA così strutturata:
  interface SlimCarrier { string getType();
  string getName();
  string getformaBase();
  string getsense();
  string getInterface(); };
- RelationCarrier getSIMRelation() ritorna le relazioni intra-schema ricavate dall'interfaccia intelligente del wrapper(vedi capitolo successivo). RelationCarrier è così strutturata: interface RelationCarrier {
  string getSrc();
  string getRel();
  string getDst(); };
- void setFileSIM() imposta il file in cui sono memorizzare le relazioni intra-schema;
- void setFileSLIM() imposta il file in cui è memorizzata l'annotazione lessicale;

L'utilizzo di CORBA consente a SI_Designer di invocare questi metodi indipendentemente da dove si trovino, infatti al mediatore è sufficiente ottenere, mediante il *Naming Service*, un *object reference* del wrapper e riferirsi a questo come fosse in locale.
SlimCarrier, RelationCarrier e MomisResultSet sono classi create direttamente da CORBA, più precisamente da idltojava, e servono unicamente per trasportare i dati attraverso la rete; si noti che nella descrizione IDL non sono presenti le proprietà delle classi ma solo i metodi. Infatti l'utilizzo delle suddette classi è subordinato alla creazione, lato server, di un'estensione delle stesse che contenga anche le

proprietà a cui i metodi fanno riferimento. Le estensioni in questione si chiamano: WSlimCarrier, WRelationCarrier e WMomisResultSet.
Il client si disinteressa dei dettagli implementativi, per il suo funzionamento è sufficiente la conoscenza dei parametri da passare al metodo e il tipo del suo valore di ritorno.

4.2.4 Esportazione delle relazioni intra-schema e dell'annotazione dei significati

L'acquisizione della descrizione ODL$_{I^3}$rappresenta una fase molto importante nel processo di integrazione, ODL$_{I^3}$però , per sua natura, contiene unicamente elementi di conoscenza riguardanti la struttura della base di dati, elementi molto importanti ma non sufficienti per ottenere una buona integrazione delle sorgenti.
In questa fase del processo di wrapping si forniscono a SI_Designer una serie di informazioni non solo strutturali, ma lessicali e ottenute dalla struttura mediante tecniche di estrazione.
Le tecniche individuate per l'esportazione di questi dati, fermo restando l'utilizzo di CORBA per il loro trasferimento, sono state sostanzialmente due: la prima è di salvare le informazioni su file che verranno lette dal wrapper il quale si preoccuperà del trasferimento, la seconda è attivare un agente software nel quale memorizzare le informazioni , ad esempio all'interno di un vettore, ed attendere l'interrogazione da parte di SI_Designer.
Pur essendo probabilmente più veloce la seconda soluzione si è scelto di utilizzare la prima per diverse ragioni:

- l'esiguità della quantità di dati da leggere dal file fa si che la velocità di risposta non venga di fatto influenzata. A ciò si aggiunge il fatto che MOMIS è un sistema utilizzato in rete, un supporto con velocità di trasferimento molto più basse rispetto, anche solo, al sottosistema dischi di una "normale " workstation.

- Un ulteriore processo attivo sull'host ospitante il wrapper comporta senz'altro un appesantimento del carico di lavoro sulla macchina che si è scelto di evitare, in particolare perchè non si è ritenuto portasse reali benefici dal punto di vista prestazionale.

I file in questione sono residenti sulla macchina ospitante, per quanto riguarda la loro generazione verrà trattata nel capitolo 5.

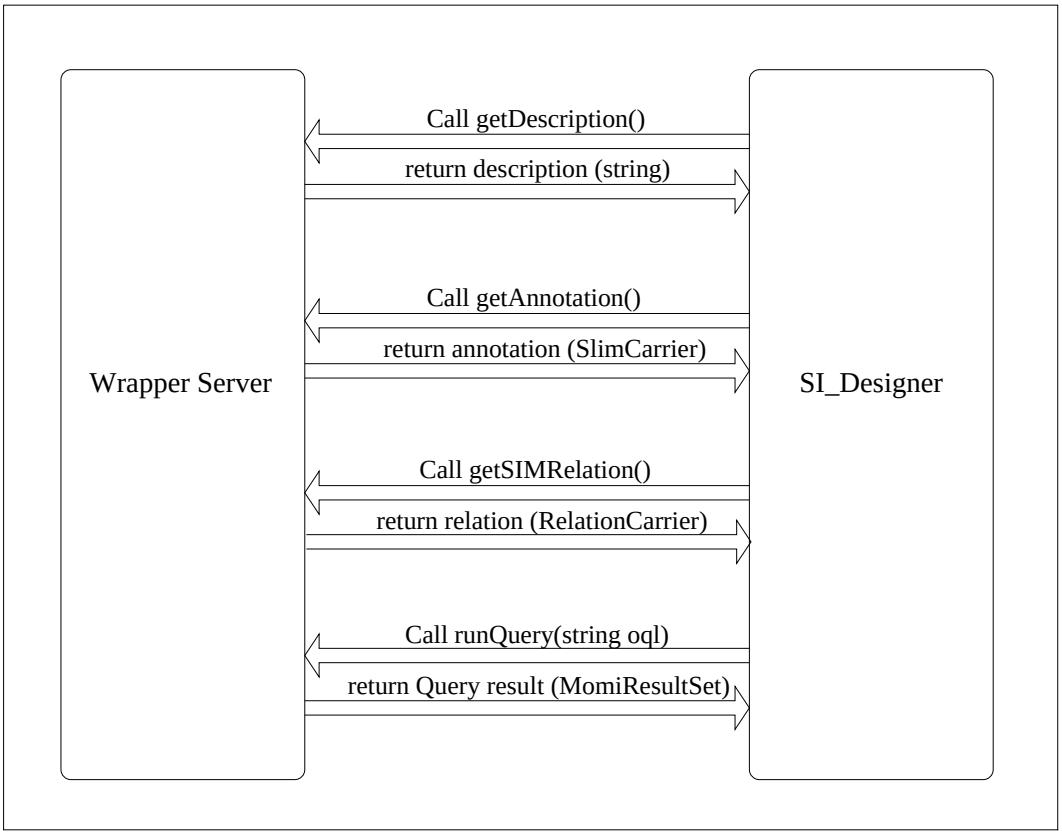

Figura 4.3: Servizi del Wrapper

4.3 Considerazioni

Il wrapper si presenta quindi come un fornitore di servizi (Server) indispensabile per realizzare il processo di integrazione. Viene riportata in figura 4.3 una descrizione grafica delle funzionalità fornite dalla classe Wrapper_server nella quale sono messi in evidenza , in particolare, i collegamenti con MOMIS.

Il sistema utilizzato può essere inserito in qualuque wrapper, compresi quelli già esistenti, semplicemente aggiungendo alle classi Wrapper_server dei vari wrapper i metodi:

- SlimCarrier getAnnotation()
- RelationCarrier getSIMRelation()
- void setFileSim()

- void setFileSlim()

Una possibile ipotesi di ricerca potrebbe essere quella di spostare i wrapper verso l'integratore, cioè non più avere una classe wrapper su ogni sorgente, ma avere una sola classe wrapper(una per ogni tipo di sorgente) residente sull'host che ospita l'integratore e da questa creare tanti oggetti quante sono le fonti di quel tipo da integrare.
Il principio è quello di non appesantire il carico di lavoro dei DBMS server o comunque delle macchine che ospitano le sorgenti e non solo, questo tipo di approccio facilita notevolmente la manutenzione e l'aggiornamento dei wrapper dovendo in questo caso lavorare su una sola anzichè su più classi.

Capitolo 5

L'interfaccia Intelligente: WrapperInterface

L'adozione di un'interfaccia intelligente per il wrapper, nasce dall'esigenza di spostare verso la fonte dati da integrare tutte quelle fasi che consentono l'acquisizione di elementi cognitivi utili al processo d'integrazione. Un approccio di questo tipo si rende necessario sostanzialmente per due ragioni:

- Ottenere un'automatizzazione sempre più spinta del processo di integrazione vero e proprio, limitando al massimo l'intervento umano, quello cioè del progettista.
- La seconda è legata, invece, alla qualità delle informazioni acquisite. Una volta creata la sorgente dati infatti sarà il suo stesso creatore ad annotarne i significati e ad estrarre le relazioni intra_schema. Questo porterà sicuramente ad un incremento della significatività delle informazioni fornite, basti pensare all'annotazione lessicale: solamente colui che ha creato la sorgente può conoscere con assoluta certezza il significato delle parole da lui stesso utilizzate.

Nella fase di esportazione dell'annotazione lessicale e delle relazioni intra_schema descritta nel capitolo 4, si è detto che le informazioni riguardanti tale esportazione sono contenute in file, questi vengono creati utilizzando, appunto, il tool WrapperInterface.
Per eseguire l'applicazione si usa il seguente comando:

java WrapperInterface wrapperInterface.conf

Dove wrapperInterface.conf è il file di configurazione del tool e contiene informazioni indispensabili per il suo funzionamento.

5.1 Acquisizione della sorgente: modulo WSAM

Una volta avviata l'applicazione ci troviamo di fronte una schermata simile a quella del modulo di acquisizione delle sorgenti di MOMIS. Il modulo in questione si chiama WSAM, cioè **W**rapper **S**ource **A**cquisition **M**odule, il suo compito è quello di acquisire la descrizione ODL_{I^3} della sorgente quindi metterla a disposizione delle fasi successive.
Il fatto che venga acquisita solamente la descrizione ODL_{I^3} fa si che l'intera interfaccia sia compatibile con qualunque tipo di fonte dati, quindi, nonostante in questa tesi si faccia riferimento unicamente a sorgenti relazionali, è possibile estendere le considerazioni anche agli altri tipi di strutture.
WSAM differisce da SAM in due aspetti fondamentali, il primo è che WSAM permette di acquisire una sola sorgente, mentre SAM consente l'acquisizione multipla di sorgenti, la seconda differenza è legata all'importazione di una cache di significati preesistenti e la sua spiegazione è rimandata al paragrafo 5.3.3.
In figura 5.1 è riportato lo screenshot iniziale di WrapperInterface, per acquisire la sorgente è necessario inserire una serie di informazioni negli appositi riquadri.
I dati da inserire sono: il nome del wrapper a cui collegarsi, il nome dell'host che lo ospita e la porta di comunicazione. Una volta digitate queste informazioni è sufficiente premere il tasto "ADD" e la sorgente viene acquisita. Acquisita la sorgente si passa alla fase successiva, quella di WSIM.

5.2 Estrazione delle relazioni intra_schema:modulo WSIM

Il compito del modulo WSIM (**W**rapper **S**ource **I**ntegrator **M**odule) è quello di analizzare gli schemi delle sorgenti per poi ricavarne delle relazioni.
Queste relazioni derivano direttamente dalla struttura della sorgente, quindi già implicite all'interno della descrizione ODL_{I^3}.
Nel caso relazionale l'estrazione delle relazioni intra-schema che riguardano le sorgenti viene effettuata mediante l'analisi delle foreign key. Ogni volta che è presente una foreign key è immediato dedurre una relazione RT fra la classe che la definisce e quella che viene da essa referenziata.

Esempio 2 Si consideri ad esempio la tabella `SCHOOL_MEMBER`, così descritta in ODL_{I^3}:

```
    // —— [COURSE]
interface COURSE(source relational univers
//[Primary Keys]
```

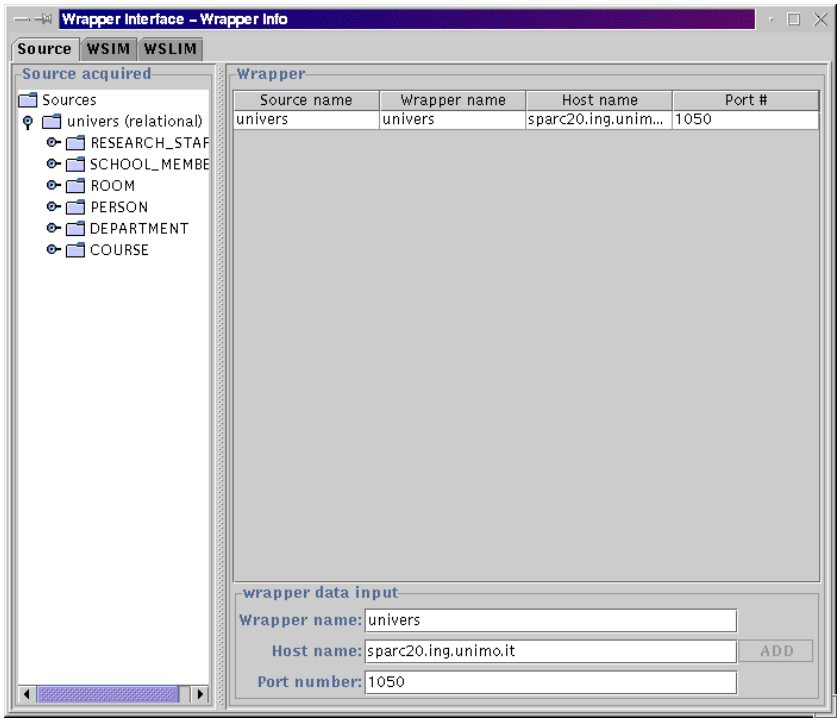

Figura 5.1: Modulo WSAM di WrapperInterface

key(COURSE_ID)
//[Foreign Keys]
foreign_key (ROOM_ID) references ROOM)
{ attribute long /* INTEGER */ COURSE_ID;
attribute string /* VARCHAR */ COURSE_NAME;
attribute long /* INTEGER */ LENGTH;
attribute long /* INTEGER */ ROOM_ID;
};

Analizzando questa descrizione il sistema riconosce che vi è una relazione che lega `COURSE` a `ROOM` attraverso la definizione della foreign key sull'attributo `ROOM_ID`, generando in questo modo la relazione:
⟨`univers.COURSE` RT `Univers.ROOM`⟩

Se oltre alle foreign key si studiano anche altri metadati, come ad esempio primary key e candidate key, è possibile ottenere ulteriori relazioni. In particolare:

- **relazione** NT_{ext}: si ricava quando una foreign key è anche chiave (primaria o candidata) della tabella che la definisce. In questo caso è presente una

relazione NT tra la tabella che contiene la chiave esterna e quella riferita (in questo ordine);

- **relazione** SYN_{ext}: quando fra due tabelle è presente una relazione reciproca di NT (o equivalentemente di BT). In questo caso le due tabelle sono sinonime.

- **relazione** RT_{partof}: si ha quando la foreign key è parte di una chiave (primaria o candidata) della tabella che la definisce. In questo caso è presente una relazione RT_{partof} fra la tabella in cui è definita la foreign key e quella riferita.

- **relazioni RT derivate per estensione di relazioni già presenti**: talvolta è possibile dedurre nuove relazioni RT estendendo relazioni RT o RT_{partof} già esistenti. Più precisamente si può dire che, considerata una determinata tabella e tutte le relazioni RT (o RT_{partof}) che la interessano al primo membro, si possono stabilire nuove relazioni RT fra ogni coppia di tabelle presenti al secondo membro delle suddette relazioni.

- **relazione SYN fra attributi**: Se una foreign key è formata da un attributo il cui nome non corrisponde a quello dello stesso attributo nella classe riferita, si può segnalare tra i due una relazione di sinonimia.

L'idea di aggiungere un nuovo tipo di relazione rispetto a quelle tradizionali nasce dalla necessità di esprimere un legame fra tabelle che sia più forte di una semplice relazione di aggregazione (ovvero di una RT).

5.2.1 Uso di WSIM

Per passare alla fase di estrazione delle relazioni, WSIM, è sufficiente cliccare sull'etichetta del folder WSIM, nella figura 5.2 in alto.
Una volta selezionato, WSIM si presenta diviso in due parti principali: la prima, sulla sinistra, rappresenta la sorgente dati come una struttura gerarchica, mentre la seconda, sulla destra, provvede alla visualizzazione delle relazioni estratte, quindi inizialmente vuota.

Il processo di estrazione delle relazioni viene attivato premendo il bottone "run WSIM" sulla toolbar, una volta terminato vengono visualizzate all'utente le relazioni estratte nel riquadro a destra già citato.
È possibile, a questo punto, salvare lo stato del sistema premendo il tasto "SaveStatus" collocato anch'esso nella toolbar.
Il procedimento di salvataggio è molto semplice, compare infatti una finestra che

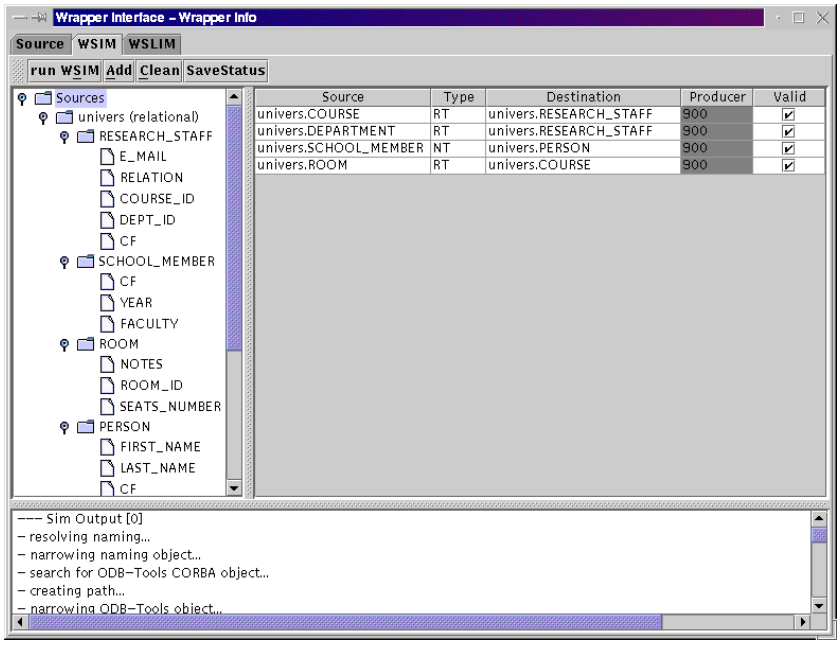

Figura 5.2: Modulo WSIM di WrapperInterface

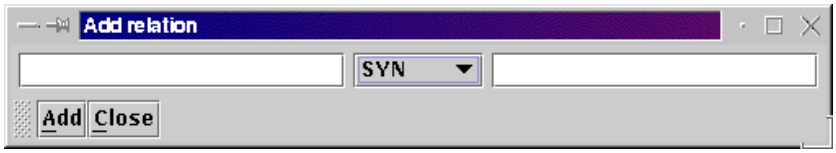

Figura 5.3: Modulo WSIM: aggiunta manuale di una relazione

richiede il nome che si vuole dare al file e la directory di destinazione. Una volta assegnato il nome al file è necessario inserire tale nome nel file di configurazione di Wrapper_server, che procederà poi all'esportazione.

Nella toolbar è presente un terzo pulsante denominato "Add", questo serve per dare la possibilità all'utente di inserire relazioni supplementari non rilevate in modo automatico(vedi figura 5.3).
Una volta aggiunte manualmente le relazioni occorre premere il tasto "SaveStatus" per memorizzare il nuovo stato. L'utilizzo di questa tecnica rafforza il ruolo dell'interfaccia intelligente nell'acquisizione di elementi cognitivi utili al processo di integrazione, consente infatti all'utente di inserire eventuali relazioni non rilevabili in modo automatico.

5.3 Il problema dell’annotazione lessicale: modulo WSLIM

I sistemi di gestione delle basi di dati sono nati per dare la possibilità all’uomo di memorizzare, e rendere di rapido accesso, grandi quantità di informazioni.
Una delle definizioni di informazione è :”Dato che si affida alla memoria del calcolatore”, da ciò si desume che i dati inseriti debbano essere necessariamente leggibili dalla macchina, leggibili però non significa “comprensibili”.
Supponiamo ad esempio di dover integrare due sorgenti di tipo relazionale per la gestione dei clienti di un’attività commerciale. Pur assumendo che non ci siano differenze strutturali tra i due schemi E/R, cosa molto improbabile, potremmo trovarci in una situazione di questo tipo:

Esempio 3 Struttura della tabella CLIENTI della prima sorgente:
TABLE CLIENTI
NOME Varchar(30)
COGNOME Varchar(30)
RECAPITO Varchar(50)

Struttura della tabella CLIENTI della seconda sorgente:
TABLE CLIENTI
NOME Varchar(30)
COGNOME Varchar(30)
INDIRIZZO Varchar(50)

Risulta evidente che l’ipotetico tool d’integrazione non sarebbe in grado, utilizzando semplicemente confronti booleani, di cogliere la sinonimia tra i concetti RECAPITO ed INDIRIZZO, rendendo indispensabile l’intervento umano per risolvere l’ambiguità.
I casi che si possono presentare sono moltissimi infatti, come scrive nel suo articolo [24] Bates:*”the probability of two persons using the same term in describing the same thing is less than 20%”*. Ciò rende necessaria la creazione di strumenti linguistici che permettano alle macchine di interpretare ed elaborare i termini utilizzati.

Questo problema è molto sentito anche in ambiente internet in cui troviamo una grande mole di informazioni spesso disomogenee, oltre che dal punto di vista linguistico, anche da quello strutturale.

Allo scopo di chiarire il panorama delle soluzioni possibili al problema riportiamo brevemente una delle proposte del W3C chiamata RDF.

5.3.1 Il Modello RDF

RDF [25] , **R**esource **D**escription **F**ramework, è un progetto W3C volto a fornire strumenti per l'interoperabilità tra applicazioni che scambiano informazioni "*machine-understandable*" sul Web. Le informazioni trattate sono delle descrizioni di risorse Web. Lo scambio di descrizioni avviene principalmente in XML anche se le risorse descritte possono essere di qualunque genere (anche non XML).

La soluzione propone l'uso di *metadati*, i quali sono *"data about data"*, definizione che in ambiente internet diventa: *"dati che descrivono risorse Web"*. La distinzione tra dati e metadati è spesso molto sottile e dipende dalla particolare applicazione in uso, molte volte infatti una stessa informazione può essere considerata nei due modi contemporaneamente.

RDF può essere usato in molte aree di applicazione; ad esempio:

- *resource discovery* per migliorare le capacità dei motori di ricerca;
- *cataloging* per descrivere il contenuto di un particolare sito Web o di una libreria digitale utilizzando agenti software intelligenti.
- ...

Il principio di base è quello di associare ad ogni risorsa una serie di proprietà ed assegnare loro dei valori. Le proprietà RDF possono essere intese come attributi delle risorse stesse e in questo senso corrispodere alle tradizionali coppie attributo-valore.

Il modello dati fondamentale di RDF è costituito da tre tipi di oggetti:

1. *Resources*: si intende per risorsa tutto ciò che può essere descritto con espressioni RDF . Una risorsa può essere una pagina Web, una parte di essa, un insieme di pagine,ecc...
2. *Properties*: Una proprietà è un aspetto specifico, un attributo o una relazione usata per descrivere una risorsa.
3. *Statement*:L'insieme risorsa,proprietà e valori delle proprietà è chiamato statement. Prese individualmente le tre parti di uno statement sono chiamate rispettivamente:*subject*, *predicate* e *object*. L'object di una risorsa può essere un'altra risorsa o una variabile letterale; per esempio una stringa o altri tipi primitivi definiti in XML.

L'utilizzo di metadati accomuna l'approccio RDF all'approccio adottato in questa tesi, anche se nel nostro caso i metadati vengono associati alle singole entità ed attributi degli schemi.

5.3.2 L'approccio adottato

Nel caso in esame viene trattato il problema dell'annotazione lessicale di una sorgente relazionale e, per estensione, di un qualunque tipo di sorgente.
Il processo di annotazione prevede l'assegnazione, mediante l'utilizzo del database lessicale WordNet, di due metadati ad ogni entità e ad ogni attributo dello schema. I metadati utilizzati sono:

1. *formabase*, cioè la parola che identifica l'oggetto tolti i suffissi dovuti alla declinatione o alla coniugazione;

2. *significati*, un vettore di interi che contiene l'offset dei significati riferito al synset della relativa formabase.

L'approccio adottato risulta quindi molto simile a RDF, soprattutto per la presenza di metadati, anche se applicato in un ambito diverso da quello per cui era stato pensato.
L'obiettivo del modulo è la creazione di un file, nel caso in esame un file di stream(serializzato), il cui contenuto sia costituito dai metadati necessari alla descrizione della sorgente. Ciò pone in evidenza un'altra differenza tra il metodo utilizzato e RDF, mentre in RDF i metadati vengono "incollati" alla risorsa, in WSIM innanzitutto si lavora su una descrizione ODL_{I^3}della sorgente, e non sulla sorgente stessa, in secondo luogo si crea un file, costituito da oggetti chiamati SlimNode (serializzati), contenente informazioni su tutto il database non solo metadati.
Questo tipo di approccio è imposto dalla struttura stessa di MOMIS, in quanto conferisce all'intero sistema una maggiore flessibilità e la possibilità di utilizzo su sorgenti che non sono state concepite per essere integrate.

5.3.3 Il modulo WSLIM

Per effettuare il processo di annotazione si è realizzato un tool grafico semiautomatico che consente all'operatore preposto di scegliere i significati corretti all'interno del contesto.
Una parola infatti può avere più significati a seconda del contesto in cui è inserita, ad esempio la parola *coniglio* può significare l'animale coniglio, una persona codarda, ecc ...

Per trovare la forma base a partire da un nome di identificatore declinato o coniugato WordNet fornisce un processore morfologico, il processore morfologico però non può interpretare le abbreviazioni o gli identificatori formati da più parole. Si rende necessario quindi l'intervento dell'operatore che deve immettere manualmente la forma base.
Ogni forma base, come abbiamo già visto, può avere più di un significato, si richiede dunque di selezionarlo tra una serie di significati ottenuti interfacciandosi con WordNet.
L'operatore ha la possibilità di impostare un filtro sulla base della categoria sintattica e poter scegliere tra ad esempio solo i nomi, o solo i verbi, ecc...
L'operatore ha comunque la possibilità di ignorare un termine se troppo ambiguo o fuorviante.

Per velocizzare la fase di annotazione ci si serve di una cache di significati già decisi, durante un processo di annotazione infatti si fa riferimento, molto spesso, a sorgenti dati che trattano di argomenti omogenei ad esempio di automobili, di prodotti alimentari, ecc...
Ci sono quindi ottime possibilità che ad una stessa forma base si associno gli stessi significati, riportiamo di seguito alcuni casi in cui la cache agisce come ausilio all'operatore:

- Nell'ambito delle **Basi di dati**, per esempio, l'attributo name è molto probabilmente utilizzato per indicare una cosa o una persona;
- in uno stesso **contesto** una determinata parola è usata sempre con la stessa accezione, ad esempio in ambito commerciale il termine *fattura* è sempre inteso come documento contenente i dati necessari a identificare un'operazione commerciale rilasciato dal venditore al compratore, e non come pratica di stregoneria.

Ci sono casi però in cui la cache non può essere d'aiuto all'operatore rallentando il processo di annotazione.
L'efficienza della cache dipende dalla quantità di significati in essa memorizzati, quindi nell'ipotesi di installazione di un wrapper su di una nuova sorgente la sua utilità sarebbe molto bassa. Si è scelto dunque di importare la cache direttamente da SI_Designer in quanto, essendo il destinatario di tutte le annotazioni esportate, ha la possibilità di creare una cache più completa ed efficiente.
Per importare la cache è stata utilizzata l'architettura CORBA: all'avvio l'interfaccia, che si comporta come CORBA Client, richiede la cache ad una applicazione CORBA server, chiamata exportSlimCache residente sull'host che ospita SI_Designer, la quale serializza la cache (che è una HashMap memorizzata in un

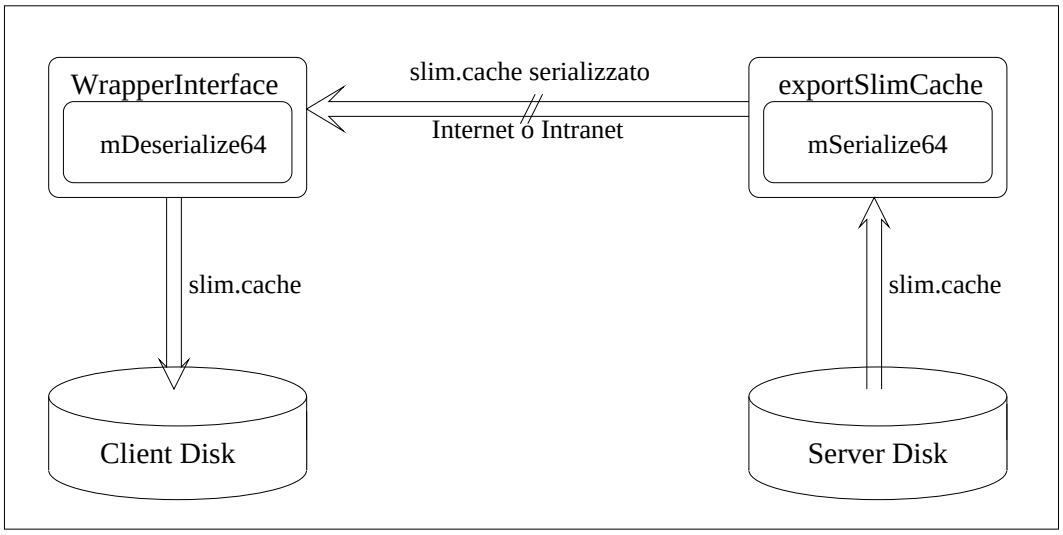

Figura 5.4: Processo di esportazione di slim.cache

file chiamato slim.cache) e la invia all’interfaccia che la salva in locale su un file (di nuovo slim.cache).
I metodi utilizzati per serializzare e deserializzare la cache, realizzati dall’ing. Alberto Corni, sono:

- String mSerialize64(java.lang.Object ser)
- Object mDeserialize64(java.lang.Object strm)

In figura 5.4 è rappresentato graficamente il processo di esportazione. Lato server per lanciare il server CORBA che esporta la cache occorre impartire da shell i seguenti comandi:

tnameserv -ORBInitialPort 1150

java exportSlimCache_server -ORBInitialHost sparc20.ing.unimo.it -ORBInitialPort 1150

Il primo comando imposta la porta di comunicazione, mentre il secondo lancia il server vero e proprio.

5.3.4 Uso di WSLIM

Per chiarire meglio le potenzialità del modulo se ne propone un esempio d’uso. Una volta acquisita la sorgente e superata la fase WSIM cliccando sull’etichetta

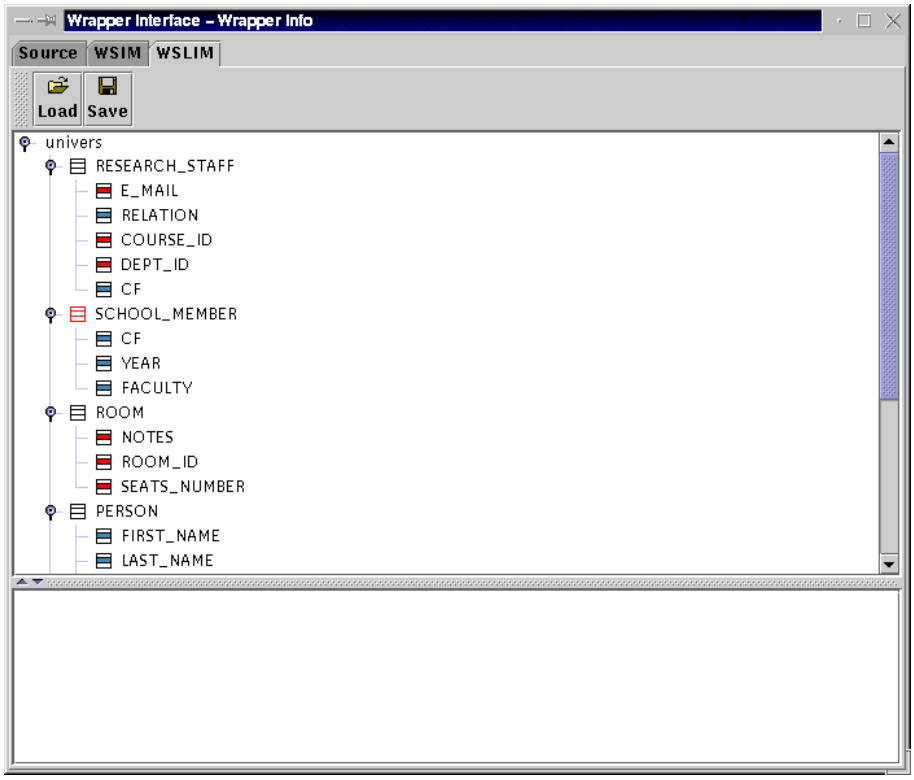

Figura 5.5: Modulo WSLIM di WrapperInterface

relativa a WSLIM si apre la schermata relativa(vedi figura 5.5): Notiamo la presenza nella toolbar di due pulsanti chiamati “Load” e “Save”. Save serve per salvare lo stato dell’annotazione su file mentre Load serve per caricare lo stato di un’annotazione incompleta.
Lo sviluppo della struttura si ottiene cliccando sui nodi dell’albero. I nodi della struttura si distinguono tra interfacce e attributi, che a loro volta si distinguono in diverse categorie. Analizziamo dapprima gli attributi:

- **Attributi di colore rosso**: attributi per i quali non si è trovata nessuna forma base né con l’ausilio del processore morfologico né grazie alla cache;
- **Attributi di colore blu**: attributi per i quali è stato possibile individuare una forma base o per merito della cache o del processore morfologico di WordNet.
- **Attributi di colore verde**: attributi ai quali l’operatore ha già associato un significato.

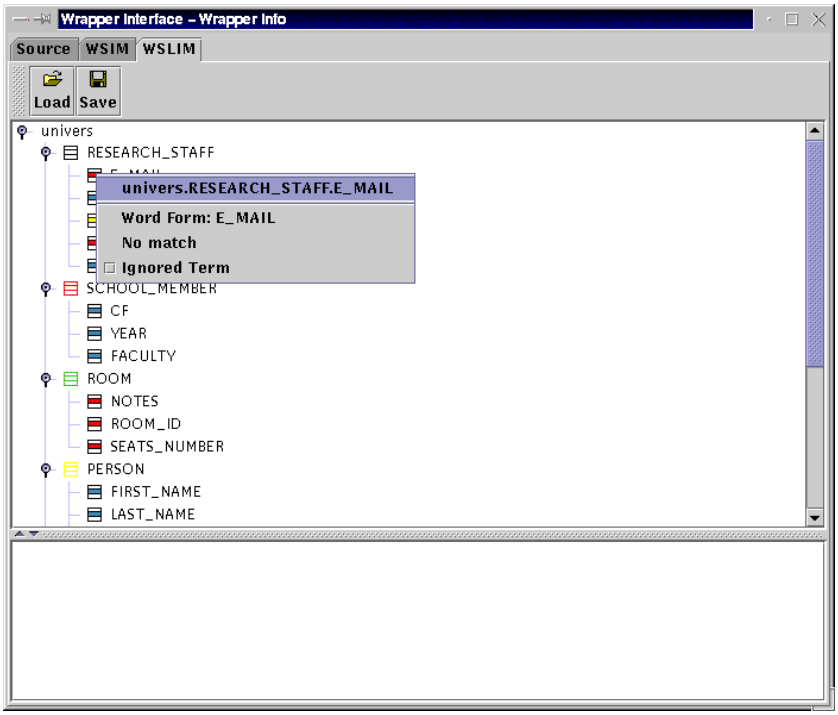

Figura 5.6: Modulo WSLIM menu di scelta

- **Attributi di colore giallo**: attributi che l’operatore ha deciso di ignorare.

Il colore distingue anche il tipo di interfaccia:

- **Interfacce di colore rosso**: interfacce per le quali non si è trovata nessuna forma base né con l’ausilio del processore morfologico né grazie alla cache;
- **Interfacce di colore nero**: interfacce per le quali è stato possibile individuare una forma base o per merito della cache o del processore morfologico di WordNet.
- **Interfacce di colore verde**: interfacce alle quali l’operatore ha già associato un significato.
- **Interfacce di colore giallo**: interfacce ignorate.

Premendo il tasto destro del mouse su uno dei nodi compare un menu a tendina(fig 5.6) che consente di effettuare tutte le operazioni precedentemente elencate: inserire la forma base, assegnare i significati e ignorare un termine. Per inserire la forma base in un nodo di colore rosso o per modificare quella selezionata occorre eseguire le seguenti operazioni(vedi figura 5.7):

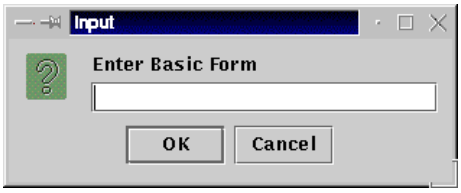

Figura 5.7: Modulo WSLIM: scelta della forma base

- cliccare con il tasto destro sul nodo;
- selezionare dal popup menu la voce ”Word Form”;
- scrivere nella finestra che compare la parola che diventerà la forma base e premere il tasto OK.

L’assegnamento dei significati avviene grazie ad una finestra di popup che compare premendo, nel menu a tendina, la voce “select sense”.
La finestra di selezione dei significati(fig 5.8 significati di YEAR) permette di selezionare, cliccando sui significati visualizzati, di assegnare ad ogni nodo dell’albero uno o più significati.

Una volta terminata la procedura non resta altro che salvare lo stato dell’albero su file per rendere disponibile l’annotazione al Wrapper. La procedura di salvataggio si avvia premendo il tasto “save” sulla toolbar. Premuto il tasto comparirà la classica finestra di salvataggio dei file che permette di scegliere sia il nome del file sia la directory in cui deve essere salvato.

5.4 Considerazioni

WrapperInterface si è rivelata un’interfaccia molto potente e versatile, infatti può essere utilizzata su un qualunque tipo di sorgente sia essa ad oggetti, relazionale, semistrutturata o semplici file di testo. Il suo utilizzo consente di fornire al mediator nuovi elementi di conoscenza tali da rendere più efficente il processo di integrazione.

Lo strumento più utile di tutta l’interfaccia è senza dubbio il modulo WSLIM in quanto consente di acquisire informazioni altrimenti non ottenibili, un possibile miglioramento al modulo, che interessa anche SI_Designer, potrebbe essere quello di dare la possibilità al progettista di cambiare la cache di slim durante la fase di utilizzo, avere cioè più cache. La presenza di più cache consentirebbe di associarne una ad ogni GlobalSchema, darebbe quindi la possibilità ad ogni

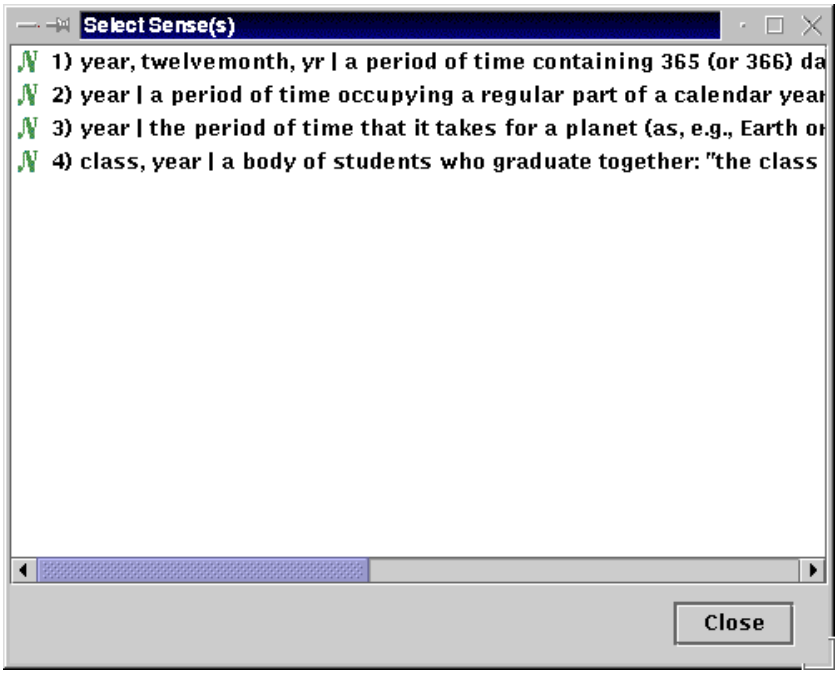

Figura 5.8: Modulo WSLIM scelta dei significati

sorgente di accedere solamente alla cache relativa al suo ambito di integrazione, assicurando così una maggiore coerenza nella scelta dei significati.

Facendo riferimento a quanto detto nella sezione 3.7.1 a proposito dell’aggiunta di nuove sorgenti ad un global schema preesistente, oltre alle problematiche già citate sorgono altre difficoltà. Ogni sorgente infatti può fornire informazioni fuorvianti per l’attività di SI_Designer in fase di integrazione. Sarebbe possibile limitare questi effetti associando ad ogni fonte dati un coefficiente e sfruttare le informazioni da essa ottenute sulla base di questo parametro.

Conclusioni

MOMIS si colloca nell'ambito dell'integrazione delle sorgenti di informazioni, come ampiamente ribadito nei precedenti capitoli l'obiettivo del progetto è quello di realizzare un'integrazione "*intelligente*" delle informazioni. Con la parola "*intelligente*" si intende che durante il processo di integrazione vengono utilizzati strumenti di intelligenza artificiale.
Il progetto mira a creare una *vista virtuale* delle fonti coinvolte nel processo, cioè , al contrario per esempio delle *datawarehouse*, non vuole materializzare una vista su una macchina locale, infatti nessun dato viene "*copiato*", le uniche informazioni che contribuiscono alla creazione della vista sono *metadati*.

In questa tesi è stato realizzato un wrapper intelligente per sorgenti relazionali interfacciabili tramite JDBC, l'aggettivo "intelligente" è dovuto al fatto che il wrapper in questione assolve alcuni dei compiti che rendono non completamente automatico il processo, in particolare: l'estrazione delle relazioni intra-schema e l'annotazione lessicale delle sorgenti. Queste due operazioni sono fondamentali per ottenere un buon risultato nella generazione della vista virtuale, ma rallentano notevolmente il processo di integrazione.
Si è ampiamente ribadito che l'obiettivo di questa tesi non era solo ridurre il lavoro del progettista in fase di integrazione, ma anche di migliorare la qualità e la quantità delle informazioni a disposizione del GlobalSchemaBuilder. L'avvicinamento al wrapper delle suddette operazioni ha infatti notevolmente migliorato la conoscenza delle sorgenti stesse, basti pensare alla fase di annotazione lessicale: chi meglio di colui che ha creato la base di dati può conoscere il significato delle parole che ha usato.
Ovviamente anche il fatto che l'estrazione delle relazioni intra-schema venga fatta direttamente sul database è un vantaggio: il suo creatore ha così la possibilità di controllarle ed eventualmente aggiungerne di nuove, incrementando così la conoscenza della fonte stessa.
L'implementazione del software si è articolata nelle seguenti fasi:

- Sviluppo di un wrapper per sorgenti JDBC compatibili;
- Progettazione di una GUI *intelligente* per l'annotazione lessicale e l'estrazione delle relazioni intra-schema.

Come già sottolineato in precedenza le funzionalità sviluppate in questa tesi possono essere utilizzate, nell'ambito di MOMIS, per qualunque operazione di wrapping; infatti l'interfaccia è del tutto autonoma dalla natura della sorgente oggetto della suddetta operazione.

Appendice A

Glossario I^3

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

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

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

A.1 Architettura

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

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

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

A.2 Servizi

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

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

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

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

A.3 Risorse

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

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

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

A.4 Ontologia

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

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

Appendice B

The ODL$_{I^3}$ description language

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

Descrizioni IDL

La creazione di un oggetto CORBA passa per la definizione della sua interfaccia IDL, in questa appendice verranno mostrate le interfacce IDL degli oggetti CORBA utilizzati in questa tesi, vale a dire MomisApplic.idl e exportSlimCache.idl.

C.1 MomisApplic.idl

```
module MomisApplic {

    /* Used to transfer serialized java objects
*/
typedef sequence¡octet¿ str;

    /*
* ———————————————————-
* external objects
* ———————————————————-
*/

    /*
* Generic exception throwed by this object.
*
* @return The number of columns in this result set. */
exception momisOqlException{
string message;
};
```

```
    /*
* The interface for any Query output.
*
* This is a very simple interface to handle the results of a query.
* It is possible to discover:
*
* Number of columns
* type of each column
* name of each column
* the value of a cell in the current row and specified column
* go to the next row
* close the returnset and free all allocated resources
*
*/
interface MomisResultSet {
/*
* Get the number of column in the resultset.
*
* @return The number of columns in this result set. */
long getColumnCount() raises (momisOqlException);
/*
* Get the name of the specified column in the resultset.
*
* @param column the number of the chosen column.
* @return The name of the chosen column. */
string getColumnName(in long column) raises (momisOqlException);
/*
* Get the Type of the specified column in the resultset.
*
* @param column the number of the chosen column.
* @return The type of the chosen column. */
long getColumnType(in long column) raises (momisOqlException);
/*
* Moves the cursor down one row from its current position.
*
* A ResultSet cursor is initially positioned before the
* first row; the
* first call to next makes the first row the current row;
* the second
* call makes the second row the current row, and so on.
*
```

```
* @return true if the new current row is valid;
* false if there are no more rows */
boolean next() raises (momisOqlException);
/*
* Releases this ResultSet object's database and resources
* immediately
*
* instead of waiting for this to happen when it is automatically
* closed. */
void close() raises (momisOqlException);
/*
* Get the position of a column identified by name.
*
* @param column the name of the chosen column.
* @return The position of the chosen column or -1 if such name
* does not exists. */
long getColumnByName(in string column) raises (momisOqlException);
/*
* Get the Long value of the specified column in the resultset.
*
* @param column the number of the chosen column.
* @return The long value of the choosen cell. */
long getLong(in long column) raises (momisOqlException);
/*
* Get the Char value of the specified column in the resultset.
*
* @param column the number of the chosen column.
* @return The char value of the choosen cell. */
char getChar(in long column) raises (momisOqlException);
/*
* Get the double value of the specified column in the resultset.
*
* @param column the number of the chosen column.
* @return The double value of the choosen cell. */
double getDouble(in long column) raises (momisOqlException);
/*
* Get the float value of the specified column in the resultset.
*
* @param column the number of the chosen column.
* @return The float value of the choosen cell. */
float getFloat(in long column) raises (momisOqlException);
```

```
/*
* Get the string value of the specified column in the resultset.
*
* @param column the number of the chosen column.
* @return The string value of the choosen cell. */
string getString(in long column) raises (momisOqlException);
/*
* String representation.
*
* Try to put the whole recordset in a string.
* @param column the number of the chosen column.
* @return The string value of the ResultSet. */
string stringSet();
/*
*
* Interface constants.
*
*/
/*
* ResutSet TYPE.
*/
/* Unsupported Type */
const long TYPE_Unsupported = -1;
/* Long Type */
const long TYPE_Long = 1;
/* Char Type */
const long TYPE_Char = 2;
/* Double Type */
const long TYPE_Double = 3;
/* Float Type */
const long TYPE_Float = 4;
/* String Type */
const long TYPE_String = 5;
};
/*
*/
interface RelationCarrier {
string getSrc();
string getRel();
string getDst();
};
```

```
interface SlimCarrier {
string getType();
string getName();
string getformaBase();
string getsense();
string getInterface();
void setInterface(in string sourceInterface);
void setType(in string sourceType);
void setName(in string sourceName);
void setformaBase(in string sourceformaBase);
void setsense(in string sourcegetsense);
};
/*
* The interface for any MOMIS Wrapper.
*
* A wrapper is the object that allows the access
* to a given data source.
*/
interface Wrapper {
/*
* Get the source type.
*
* @return The string describing the soruce.
* Nowaday yypes can be:
* - JDBC
* - XML
*/
string getType() raises (momisOqlException);
/*
* Get the description of the source in ODLi3.
*
* @return The string describing the soruce.
*/
string getDescription() raises (momisOqlException);
/*
* Executes a Query OQL on the GlobalSchema specified
*
* @param oql the oql description of teh query.
* @return return the recordset of the
*/
// string runQuery( in string oql ) raises (momisOqlException);
```

```
MomisResultSet runQuery( in string oql ) raises (momisOqlException);
/*
* Return the source name.
*/
string getSourceName() raises (momisOqlException);
/*
* Return the Annotation from the Source
*/
SlimCarrier getAnnotation() raises (momisOqlException);
void setFileSIM();
void setFileSLIM();

    RelationCarrier getSIMRelation() raises (momisOqlException);
};

    /*
* Internal MOMIS ojbects
*
* All the following ojects are instantiable
* using the object MomisFactory
*/
interface GlobalSchema {
//
// METHODS
//
/*
* Sets the global schema name.
*
* @param name The new global schema name.
*/
void setName(in string name);
/*
* Return the global schema name.
*
* @return The global schema name.
*/
string getName();

    /*
* Return a java serialization with all the objects representing
* the status of GlobalSchema.
```

```
*
* @return The java serialization of some objects.
*/
string getStatus();
/*
* Sets the internal status of the GlobalSchema with the deserialization
* of the parameter that must be a string produced by
* getStatus
* of this or another GlobalSchema CORBA object.
*
* @param status The java serialization of some objects.
*/
void setStatus(in string status);
/*
* This method kill the servant object.
*/
long killObject();
};

    /*
* The interface for the MOMIS QueryManager.
*
* A wrapper is the object that allows
*/
interface QueryManager {
/*
* Returns the schema on which the query manager runs.
*
* @return Returns the schema on which the query manager runs.
*/
GlobalSchema getSchema();
/*
* Executes a Query OQL on the GlobalSchema specified
*
* @param oql the oql description of teh query.
* @return return the recordset of the
*/
MomisResultSet runQuery(in string oql);
// MomisCollection runQuery(string oql);
/*
* Kills the current object and free all resources.
```

```
*
* @return ever returns the 0 (zero) value.
*/
long killObject();
};


    /*
* Main SERVER Object.
*
* Each MOMIS object can be created by this unique reference object.
* The SERVER object is unique, it serves new request
* of services creating new SERVANT objects.
* This object also
* manages existing SERVANT killing them for example after a
* time out period.
*
* The only functionality provided by the server object is the
* possibility of creating new SERVANT objects.
*/
interface MomisFactory {
/*
* Creates new GlobalSchema CORBA object and register it in
* the naming service.
*
* @return Reference to the CORBA object.
*/
GlobalSchema newGlobalSchema();


    /*
* Creates new QueryManager on a given GlobalSchema.
*
* @return Reference to the CORBA object.
*/
QueryManager newQueryManager(in GlobalSchema globalSchema);
};


    };
```

C.2 MomisSlimCache.idl

```
module MomisSlimCache {

    interface exportSlimCache{
string getCache();

    void setCacheFile();

    };

    };
```

Appendice D

Esempio di riferimento per l'integrazione

Nel capitolo 3 è riportato un esempio di integrazione fatto con il sistema MOMIS, si riportano di seguito le descrizioni ODL_{I^3} delle sorgenti coinvolte

D.1 Sorgente univers

```
// —— [COURSE]
interface COURSE(source relational univers
//[Primary Keys]
key(COURSE_ID)
//[Foreign Keys]
foreign_key (ROOM_ID) references ROOM)
{ attribute long /* INTEGER */ COURSE_ID;
attribute string /* VARCHAR */ COURSE_NAME;
attribute long /* INTEGER */ LENGTH;
attribute long /* INTEGER */ ROOM_ID;
};

    // —— [DEPARTMENT]
interface DEPARTMENT(source relational univers
//[Primary Keys]
key(DEPT_ID))
{ attribute long /* INTEGER */ DEPT_ID;
attribute string /* VARCHAR */ DEPT_NAME;
attribute string /* VARCHAR */ BUDGET;
```

```
attribute string /* VARCHAR */ AREA;
};

    // —— [PERSON]
interface PERSON(source relational univers
//[Primary Keys]
key(CF))
{ attribute string /* VARCHAR */ CF;
attribute string /* VARCHAR */ FIRST_NAME;
attribute string /* VARCHAR */ LAST_NAME;
};

    // —— [RESEARCH_STAFF]
interface RESEARCH_STAFF(source relational univers
//[Primary Keys]
key(CF)
candidate_key SQL010619165155800 (E_MAIL)
//[Foreign Keys]
foreign_key (COURSE_ID) references COURSE
foreign_key (DEPT_ID) references DEPARTMENT)
{ attribute string /* VARCHAR */ CF;
attribute string /* VARCHAR */ RELATION;
attribute string /* VARCHAR */ E_MAIL;
attribute long /* INTEGER */ DEPT_ID;
attribute long /* INTEGER */ COURSE_ID;
};

    // —— [ROOM]
interface ROOM(source relational univers
//[Primary Keys]
key(ROOM_ID))
{ attribute long /* INTEGER */ ROOM_ID;
attribute long /* INTEGER */ SEATS_NUMBER;
attribute string /* VARCHAR */ NOTES;
};

    // —— [SCHOOL_MEMBER]
interface SCHOOL_MEMBER(source relational univers
//[Primary Keys]
key(CF)
//[Foreign Keys]
```

```
foreign_key (CF) references PERSON)
{ attribute string /* VARCHAR */ CF;
attribute string /* VARCHAR */ FACULTY;
attribute long /* INTEGER */ YEAR;
};
```

D.2 Sorgente TaxPosition

```
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; };
```

D.3 Sorgente Computer_Science

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

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

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

```
attribute string rank;
attribute string home_email;
attribute string phd_email;};

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

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

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

Bibliografia

[1] S. Montanari. Un approccio intelligente all'Integrazione di Sorgenti Eterogenee di Informazione. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[2] A. Rabitti. Architettura di un Mediatore per un Sistema di Sorgenti Distribuite ed Autonome. Tesi di Laurea, Università di Modena, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 1997-1998.

[3] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. An Intelligent Approach to Information Integration. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, june 1998.

[4] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. Exploiting schema knowledge for the integration of heterogeneous sources. In *Sesto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD98, Ancona*, pages 103–122, 1998.

[5] S.Bergamaschi, S.Castano, D.Beneventano, and M.Vincini. Semanitc Integration and Query of Heterogenous Information sources. *Journal of Data and Knowledge Engineering 1*, 1999.

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

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

[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] N.Guarino. Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. Technical report, Summer School on Information Extraction, Frascati, Italy, July 1997.

[13] N.Guarino. Understanding, Building, and Using Ontologies. A commentary to 'Using Explicit Ontologies in KBS Development', by van Heijst, Schreiber, and Wielinga.

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

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

[16] Francesco Venuta. Il componente query manager di momis: esecuzione di interrogazioni, 1999-2000.

[17] 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.

[18] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. Odb-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Sesto Convegno AIIA - Roma*, 1997.

[19] A.G. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[20] S. Castano and V. De Antonellis. Deriving Global Conceptual Views from Multiple Information Sources. In *preProc. of ER'97 Preconference Symposium on Conceptual Modeling, Historical Perspectives and Future Directions*, 1997.

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

[22] Elisa Marri. Sviluppo di tecniche di estrazione ed inferenza di relazioni terminologiche nel sistema momis, 1999-2000.

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

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

[25] Bray T. Rdf and metadata. *http://www.xml.com/pub/a/2001/01/24/rdf.html*, 2001.