UNIVERSITA' DEGLI STUDI DI MODENA
E REGGIO EMILIA
Facoltà di Ingegneria – Sede di Modena
Corso di Laurea in Ingegneria Informatica

Progetto MOMIS: il wrapper Access/ODLi3

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Correlatore
Ing. Maurizio Vincini

Tesi di Laurea di
Giovanni Lorandini

Anno Accademico 1999 – 2000

Parole chiave:
Integrazione dati
Database relazionali
ODL_I^3
Wrapper
Access

RINGRAZIAMENTI

Un sentito ringraziamento va alla Professoressa Sonia Bergamaschi e al Professor Domenico Beneventano per la disponibilità dimostrata e l'aiuto fornito durante la realizzazione della presente tesi
Un ringraziamento inoltre all'Ing. Maurizio Vincini, all'Ing. Francesco Guerra e all'Ing. Alberto Corni per l'insostituibile e prezioso aiuto.

Indice

Elenco delle figure

Introduzione

Negli ultimi anni si è assistito ad un diffondersi esponenziale della rete Internet come non mai prima d'ora, sia in termini di utenti finali sia di aziende che utilizzano questo mezzo di diffusione per promuovere mezzi e servizi. Peculiarità di Internet è l'essere in grado di raggiungere ogni più remoto angolo del pianeta in tempi pressochè nulli rapportati ai "vecchi" mezzi di comunicazione. Infatti, fattori da non sottovalutare sono la velocità con cui queste informazioni possono essere scambiate ed il tipo di informazioni. Un'alta velocità nello scambio delle informazioni spesso comporta anche grandi quantità di informazioni quindi aumento della quantità e del genere di informazioni reperibili.

Quest'enorme disponibilità di dati è senza dubbio un fatto estremamente positivo, poiché consente a chiunque il libero accesso ad informazioni sugli argomenti più disparati. Tuttavia questa situazione costringe l'utente a dover affrontare una nuova classe di problemi generati proprio dal fatto che le informazioni presenti in rete provengono spesso da sorgenti autonome ed eterogenee: da un lato, infatti, l'utente si trova a dover selezionare le sorgenti potenzialmente utili, formulando per ognuna l'interrogazione più appropriata; dall'altro deve essere in grado di filtrare, nella molteplicità delle risposte ottenute, i dati interessanti, scartando ridondanze e inconsistenze.

E' evidente come sia opportuno da parte dell'utente avere familiarità con i contenuti, le strutture ed i linguaggi di interrogazione propri di queste risorse. E' altresì difficile ipotizzare che l'utente medio possegga la conoscenza necessaria a portare a termine un compito di questo tipo, compito che col passare del tempo diventa sempre più complesso giacché il numero di sorgenti è in continua crescita.

Si avverte dunque l'esigenza di fornire un'applicazione in grado di rendere quanto più trasparenti all'utente queste operazioni. La possibilità di sviluppare applicazioni capaci di combinare ed utilizzare dati provenienti da una molteplicità di sorgenti è un tema di grande attualità, d'interesse non solo teorico ma anche applicativo, come dimostra la sempre maggiore presenza di sistemi commerciali, quali Datawarehouse, Dataminer, Sistemi di Workflow, Federazioni di Database, etc…

Il lavoro svolto in questa tesi fa parte di un progetto più ampio denominato MOMIS (Mediator EnvirOnment for Multiple Information Sources) nato a sua volta all'interno del progetto MURST 40% INTERDATA e come collaborazione tra l'unità operativa dell'Università di Modena e l'unità operativa dell'Università di Milano. Scopo del progetto MOMIS è quello di sviluppare un sistema Mediatore che integri diverse sorgenti di informazione eterogenee e distribuite, sia strutturate che semistrutturate. In particolare, in tale ambito si è sviluppato un componente wrapper per accedere a sorgenti dati di tipo strutturato.

Avendo a disposizione un insieme di schemi locali, l'intenzione è quella di definire un'unica vista integrata che contenga l'insieme delle informazioni che le singole sorgenti intendono condividere. Dato il notevole livello di ambiguità che inevitabilmente caratterizza i problemi legati all'integrazione di informazioni eterogenee, il processo di definizione dello schema integrato non potrà essere completamente automatico, ma necessiterà dell'interazione di un progettista che, attraverso un linguaggio dichiarativo opportunamente definito (ODL$_{I^3}$), fornirà al sistema la conoscenza in suo possesso. L'uso di tecniche di Intelligenza Artificiale consente di alleggerire e velocizzare la fase di integrazione al termine della quale sarà disponibile la vista globale. L'utente potrà interagire esclusivamente con la vista integrata: le interrogazioni, pertanto, saranno formulate ad un'unica fonte "virtuale" e sarà cura del gestore delle interrogazioni (Query Manager) scomporle in diverse query (Basic Query), ognuna indirizzata ad una sorgente locale diversa. Sarà sempre onere del Mediatore riunificare i risultati. Anche in questo caso l'uso di tecniche di Intelligenza Artificiale permette di ottimizzare i tempi di accesso alle risorse, pur mantenendo inalterato l'insieme risposta.

La tesi è organizzata nel seguente modo:

Nel Capitolo 1 viene introdotta l'architettura di riferimento I^3 per i sistemi di Integrazione di Informazioni per poter illustrare le scelte implementative fatte in MOMIS. L'ultima sezione del capitolo è dedicata ai wrapper, ai servizi che offrono ed alla loro interazione con il mediatore.

Nel Capitolo 2 viene descritto il sistema MOMIS, in particolare la sua architettura, conforme alla proposta internazionale introdotta nel Capitolo 1.

Nel Capitolo 3 vengono brevemente accennati i linguaggi e gli standard che interessano più da vicino i wrapper nell'ambito del progetto MOMIS. Viene fornita inoltre una descrizione dell'architettura CORBA.

Nel Capitolo 4 viene presentato il wrapper Access/ODL$_I^3$ realizzato, unitamente ad una descrizione sul suo uso.

Sono infine presenti cinque appendici in cui si riportano:
in appendice A il glossario dei termini usati in ambito I^3, in appendice B la descrizione in BNF del linguaggio ODL, in appendice C la descrizione in BNF dell'estensione ODL$_I^3$, in appendice D la descrizione completa in ODL$_I^3$ dell'esempio di riferimento, in appendice E la restrizione del linguaggio OQL per le BasicQuery.

Capitolo 1

L'Integrazione Intelligente di Informazioni

La presenza di un numero sempre maggiore di fonti di informazione, all'interno di un azienda come sulla rete Internet, ha reso possibile oggi accedere ad un vastissimo insieme di dati, sparsi su macchine diverse come pure in luoghi diversi. Parallelamente quindi all'aumento delle probabilità di trovare un dato sulla rete informatica, in qualsivoglia fonte e formato, va costantemente aumentando la difficoltà di recuperare questo dato in tempi e modi accettabili, essendo tra di loro le fonti di informazione fortemente eterogenee, sia per quanto riguarda i tipi di dati (testuali, immagini …), sia per quanto riguarda il modo di descriverli, e quindi di *segnalarli* ai potenziali utenti. Contestualmente alla difficoltà di reperire un dato, pur nella sicurezza di ritrovarlo, si va inoltre delineando un altro tipo di problema, che paradossalmente nasce dall'abbondanza di informazioni, e che viene percepito dall'utente come *information overload* (sovraccarico di informazioni): il numero crescente di informazioni (e magari la loro replicazione) genera confusione, rendendo pressocchè impossibile isolare efficientemente i dati necessari a prendere determinate decisioni.

In questo scenario, al momento fortemente studiato, e che coinvolge diverse aree di ricerca e di applicazione, si vanno oggi ad inserire i sistemi di supporto alle decisioni (DSS, Decision Support System), l'integrazione di basi di dati eterogenee, i *datawarehouse* (magazzino), fino ad arrivare ai sistemi distribuiti. I *decision maker* lavorano su fonti diverse (inclusi file system, basi di dati, librerie digitali, …) ma sono per lo più incapaci di ottenere e fondere le informazioni in un modo efficiente.

L'integrazione di basi di dati invece, e tutto ciò che va sotto il nome di datawarehouse, si occupa di materializzare presso l'utente finale delle viste, ovvero delle porzioni delle sorgenti, replicando però fisicamente i

dati, ed affidandosi a complicati algoritmi di "mantenimento" di questi dati, per assicurare la loro consistenza a fronte di cambiamenti nelle sorgenti originali. Con *Integration of Information* invece, come è descritto in [1], si rappresentano in letteratura tutti quei sistemi in grado di combinare tra di loro dati provenienti da intere sorgenti o parti selezionate di esse, senza fare uso della replicazione fisica delle informazioni, bensì basandosi sulle loro descrizioni. Quando inoltre questa integrazione utilizza tecniche di intelligenza artificiale, sfruttando le conoscenze acquisite, possiamo parlare di Intelligent Integration of Information (I^3), che si distingue quindi dalle altre forme di integrazione prefiggendosi non una semplice aggregazione di informazioni, bensì anche un aumento del loro valore, ottenendo nuove informazioni dai dati ricevuti.

Con questi obiettivi si è quindi inserita, nell'ambito dell'integrazione, l'Intelligenza Artificiale (IA), che già aveva dato buoni risultati in domini applicativi più limitati. Naturalmente, è ovvio come sia pressocchè impossibile pensare ad un sistema che vada bene per tutti i domini applicativi, e che magari integri un numero altissimo di sorgenti. Per questo motivo, per realizzare sistemi molto ampi, è stata proposta una partizione delle risorse e dei servizi che questi sistemi devono supportare, e che si articola su due dimensioni:

- orizzontalmente, in tre livelli: livello utente, moduli intermedi che fanno uso di tecniche di IA, risorse di dati;
- verticalmente: molti domini, con un numero limitato (e minore di 10) di sorgenti.

I domini nei vari livelli si scambieranno dati e informazioni tra di loro, ma non saranno strettamente collegati. Per esempio, in un sistema di recapito merci navale, le informazioni sulle navi saranno integrate da un modulo intermedio, quelle sul tempo nelle varie regioni da un altro modulo intermedio, ed un ulteriore modulo, ad un livello superiore, provvederà all'integrazione dei dati che gli verranno forniti dai *mediatori* (o *facilitatori*) sottostanti.

In questo quadro, dal 1992, si inserisce il progetto di ricerca I^3, fondato e sponsorizzato dall'ARPA, agenzia che fa capo al Dipartimento di Difesa americano [2]. I^3 si focalizza sul livello intermedio della partizione sopra descritta, livello che media tra gli utilizzatori e le sorgenti. All'interno di questo livello staranno diversi moduli tra i quali i più importanti sono:

?? *facilitator e mediator* (le differenze tra i due sono flebili ed ancora ambigue in letteratura), che ricercano le fonti "interessanti" e combinano i dati da esse ricevuti;

?? *query* processor, che riformulano le query aumentando le probabilità di successo di quest'ultime;

?? *data miner*, che analizzano i dati per estrarre informazioni intensionali implicite.

Oltre alla architettura di riferimento, muovendosi questo progetto in un campo di ricerca particolarmente giovane e in evoluzione, è riportato in Appendice A il glossario, a cui rifarsi per termini che risultino ambigui o poco chiari.

1.1 Architettura di riferimento per sistemi I^3

L'architettura di riferimento presentata in questo paragrafo è stata tratta dal sito web [2], e rappresenta una sommaria categorizzazione dei principi e dei servizi che possono e devono essere usati nella realizzazione di un integratore *intelligente* di informazioni derivanti da fonti eterogenee. Alla base del progetto I^3 stanno infatti due ipotesi:

?? la cosiddetta "autostrada delle informazioni" è oggigiorno incredibilmente vasta e, conseguentemente, sta per diventare una risorsa di informazioni utilizzabile poco efficientemente;

?? le fonti di informazioni ed i sistemi informativi sono spesso semanticamente correlati tra di loro, ma non in una forma semplice nè premeditata. Di conseguenza, il processo di integrazione di informazioni può risultare molto complesso.

In questo ambito, l'obiettivo del programma I^3 è di ridurre considerevolmente il tempo necessario per la realizzazione di un integratore di informazioni, raccogliendo e "strutturando" le soluzioni sino ad ora prevalenti nel campo della ricerca. Da sottolineare, prima di

passare alla descrizione dell'architettura di riferimento, che questa architettura non implica alcuna soluzione implementativa, bensì vuole rappresentare alcuni dei servizi che deve includere un qualunque integratore di informazioni, e le interconnessioni tra questi servizi. Inoltre, è opportuno rimarcare che non sarà necessario, ed anzi è improbabile, che ciascun sistema che si prefigge di integrare informazioni (o servizi, o applicazioni) comprenda l'intero insieme di funzionalità che verranno descritte, bensì usufruirà esclusivamente delle funzionalità necessarie ad un determinato compito.

1.1.1 A cosa serve la tecnologia I^3 e quali problemi deve risolvere

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

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

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

Così come possono essere differenti gli obiettivi di un sistema I^3, saranno differenti pure i problemi che si troverà ad affrontare. Tra questi, possono essere identificati:

- *la grande differenza tra le fonti di informazione*:
  - le fonti informative sono semanticamente differenti, e si possono individuare dei livelli di differenze semantiche [3];
  - le informazioni possono essere memorizzate utilizzando differenti formati, come possono essere file, DB relazionali, DB ad oggetti;
  - possono essere diversi gli schemi, i vocabolari usati, le ontologie su cui questi si basano, anche quando le fonti condividono significative relazioni semantiche;
  - può variare inoltre la natura stessa delle informazioni, includendo testi, immagini, audio, media digitali;
  - infine, può variare il modo in cui si accede a queste sorgenti: interfacce utente, linguaggi di interrogazione, protocolli e meccanismi ditransazione;
- *la semantica complessa ed a volte nascosta delle fonti*: molto spesso, la chiave per l'uso delle informazioni di vecchi sistemi sono i programmi applicativi su di essi sviluppati, senza i quali può essere molto difficile dedurre la semantica che si voleva esprimere, specialmente se si ha a che fare con sistemi molto vasti e quasi impossibili da interpretare se visti solo dall'esterno;
- *l'esigenza di creare applicazioni in grado di interfacciarsi con porzioni diverse delle fonti di informazione*: molto spesso, non è sempre possibile avere a disposizione l'intera sorgente di informazione, bensì una sua parte selezionata che può variare nel tempo;
- *il grande numero di fonti da integrare*: con il moltiplicarsi delle informazioni, il numero stesso delle fonti da integrare per una applicazione, ad esempio nel campo sanitario, è aumentato considerevolmente, e decine di fonti devono essere accedute in modo coordinato;
- *il bisogno di realizzare moduli I^3 riusabili*: benchè questo possa essere considerato uno dei compiti più difficili nella realizzazione di un integratore, è importante realizzare non un sistema ad-hoc, bensì un'applicazione i cui moduli possano facilmente essere riutilizzati in altre applicazioni, secondo i moderni principi di riusabilità del software. In questo caso, l'abilità di costruire valide funzioni di

libreria può considerevolmente diminuire i tempi e le difficoltà di realizzazione di un sistema informativo che si basa su più fonti differenti.

Passiamo ora ad analizzare l'architettura vera e propria di un sistema I^3, riportata in Figura 1.1.

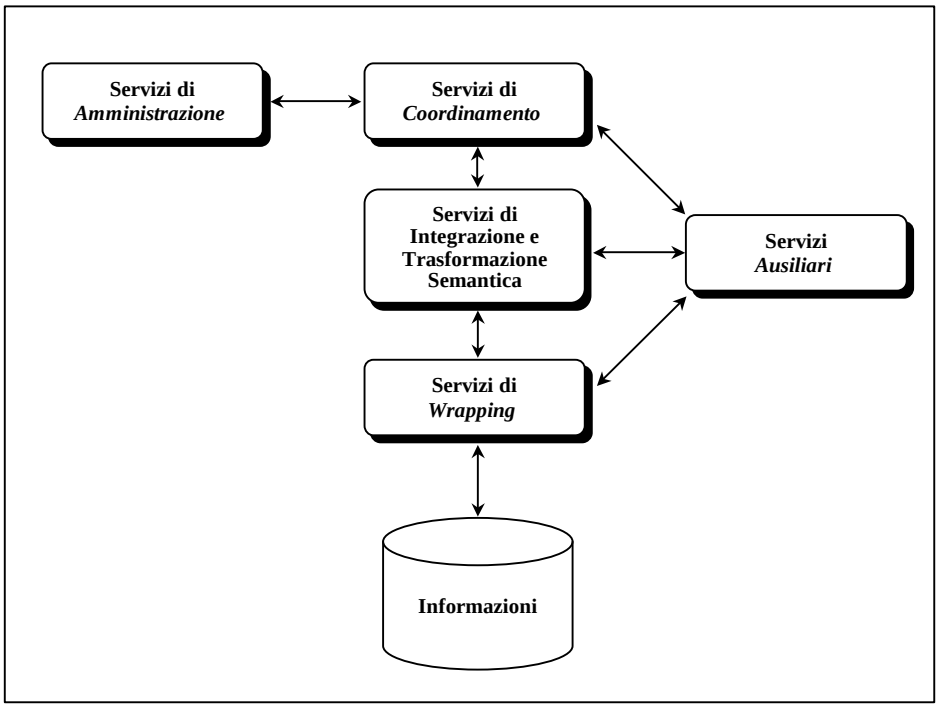


Figura 1.1: Diagramma dei servizi I^3

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

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

Analiziamone in dettaglio funzionalità e problematiche affrontate.

1.1.2 Servizi di Coordinamento

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

1. **Facilitation e Brokering Services**: l'utente manda una richiesta al sistema e questo usa un deposito di metadati per ritrovare il modulo che può trattare la richiesta direttamente. I moduli interessati da questa richiesta potranno essere uno solo alla volta (nel qual caso si parla di Brokering) o più di uno (e in questo secondo caso si tratta di facilitatori e mediatori, attraverso i quali a partire da una richiesta ne viene generata più di una da inviare singolarmente a differenti moduli che gestiscono sorgenti distinte, e reintegrando poi le risposte in modo da presentarle all'utente come se fossero state ricavate da un'unica

fonte). In questo ultimo caso, in cui una query può essere decomposta in un insieme di sottoquery, si farà uso di servizi di Query Decomposition e di tecniche di Inferenza (mutuate dall'Intelligenza Artificiale) per una determinazione dinamica delle sorgenti da interrogare, a seconda delle condizioni poste nell'interrogazione. I vantaggi che questi servizi di Coordinamento portano, consistono nel fatto che non è richiesta all'utente del sistema una conoscenza del contenuto delle diverse sorgenti, dandogli l'illusione di interagire con un sistema omogeneo che gestisce direttamente la sua richiesta. E' quindi esonerato dal conoscere i domini con i quali i vari moduli[3] hanno a che fare, ottenendone una considerabile diminuzione di complessità di interazione col sistema.

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

1.1.3 Servizi di Amministrazione

Sono servizi usati dai Servizi di Coordinamento per localizzare le sorgenti *utili*, per determinare le loro capacità, e per creare ed interpretare i *Template*. I Template sono strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task. Sono quindi utilizzati dai sistemi meno "intelligenti", e consentono all'operatore di predefinire le azioni da eseguire a seguito di una determinata richiesta, limitando al minimo le possibilità di decisione del sistema.

In alternativa a questi metodi dei Template, sono utilizzate le **Yellow Pages**: servizi di directory che mantengono le informazioni sul contenuto delle varie sorgenti e sul loro stato (attiva, inattiva, occupata). Consultando queste Yellow Pages, il mediatore sarà in grado di spedire alla giusta sorgente la richiesta di informazioni, ed eventualmente di rimpiazzare questa sorgente con una equivalente nel caso non fosse disponibile. Fanno parte di questa categoria di servizi il Browsing: permette all'utente di "navigare" attraverso le descrizioni degli schemi delle sorgenti, recuperando informazioni su queste. Il servizio si basa

sulla premessa che queste descrizioni siano fornite esplicitamente tramite un linguaggio dichiarativo leggibile e comprensibile dall'utente. Potrebbe fornirsi a sua volta dei servizi Trasformazione del Vocabolario e dell'Ontologia, come pure di Integrazione Semantica. Da citare sono pure i servizi di Iterative Query Formulation: aiutano l'utente a rilassare o meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

1.1.4 Servizi di Integrazione e Trasformazione Semantica

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

1. Servizi di **integrazione degli schemi**. Supportano la trasformazione e l'integrazione degli schemi e delle conoscenze derivanti da fonti di dati eterogenee. Fanno parte di essi i servizi di trasformazione dei vocaboli e dell'ontologia, usati per arrivare alla definizione di un'ontologia unica che combini gli aspetti comuni alle singole ontologie usate nelle diverse fonti. Queste operazioni sono molto utili quando devono essere scambiate informazioni derivanti da ambienti differenti, dove molto probabilmente non si condivideva un'unica ontologia. Fondamentale, per creare questo insieme di vocaboli condivisi, è la fase di individuazione dei concetti presenti in diverse fonti, e la riconciliazione delle diversità presenti sia nelle strutture, sia nei significatidei dati.
2. Servizi di **integrazione delle informazioni**. Provvedono alla traduzione dei termini da un contesto all'altro, ovvero dall'ontologia di partenza a quella di destinazione. Possono inoltre occuparsi di uniformare la "granularità" dei dati (come possono essere le discrepanze nelle unità di misura, o le discrepanze temporali).

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

1.1.5 Servizi di Wrapping

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

1. permettere ai servizi di coordinamento e di mediazione di manipolare in modo uniforme il maggior numero di sorgenti locali, anche se queste non erano state esplicitamente pensate come facenti parte del sistema di integrazione;
2. essere il più riusabili possibile. Per fare ciò, dovrebbero fornire interfacce che seguano gli standard più diffusi (e tra questi, si potrebbe citare il linguaggio SQL come linguaggio di interrogazione di basi di dati, e CORBA come protocollo di scambio di oggetti). Questo permetterebbe alle sorgenti estratte da questi wrapper "universali" di essere accedute dal numero maggiore possibile di moduli mediatori.

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

1.1.6 Servizi Ausiliari

Aumentano le funzionalità dei 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.

1.2 Il mediatore

L'obiettivo del progetto di ricerca è la progettazione e realizzazione di un **mediatore**, ovvero del modulo intermedio dell'architettura precedente-mente descritta, che si pone tra l'utente e le sorgenti di informazioni. Secondo la definizione proposta da Wiederhold in [4] "un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per una applicazione di livello superiore… Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti."

Compiti di un mediatore sono allora:

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

La prima ipotesi che è stata fatta, per restringere il campo applicativo del sistema da progettare (e di conseguenza per restringere il campo dei problemi a cui dare risposta) è di avere a che fare, per il momento, esclusivamente con sorgenti di dati testuali strutturati, come possono essere basi di dati relazionali, ad oggetti e file di testo. L'approccio architetturale scelto è stato quello *classico*, che consta principalmente di 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, convertendo le richieste del mediatore in una forma comprensibile dalla sorgente, e le informazioni da essa estratte nel modello usato dal mediatore.

Facendo riferimento ai servizi descritti nelle sezioni precedenti, l'architettura del mediatore che si è progettato è riportata in Figura 1.2.

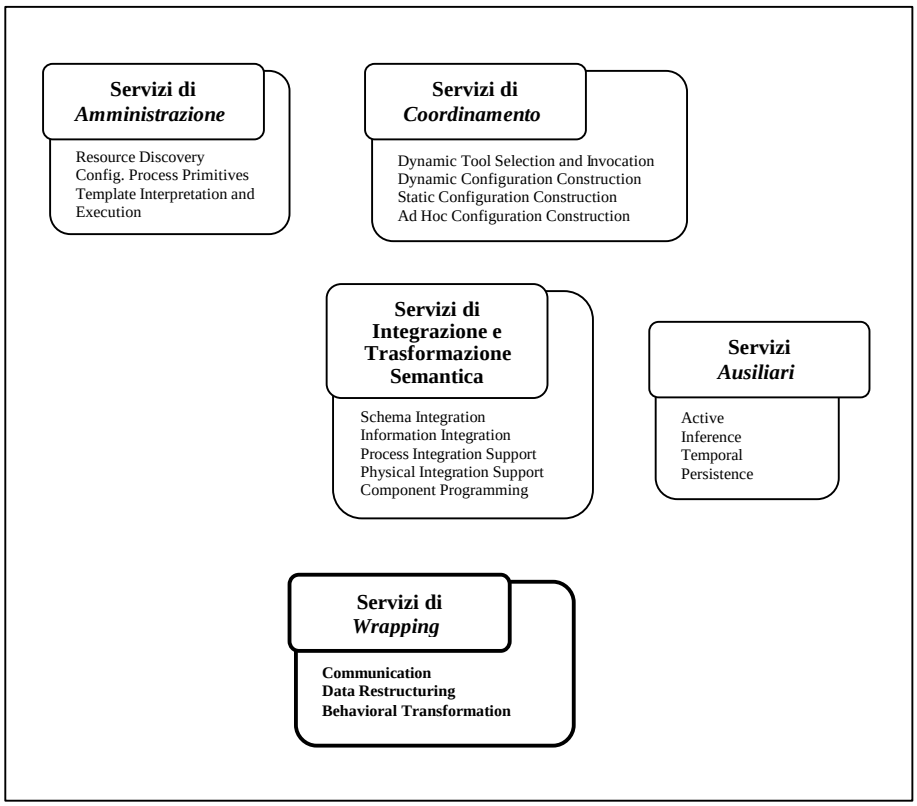


Figura 1.2: Servizi I^3 presenti nel mediatore

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

- la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo: il mediatore non si basa infatti su una descrizione predefinita degli 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 il nostro, seguono invece un approccio definito *semantico*, che è caratterizzato da questi punti:

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

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

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

1. problemi ontologici;
2. problemi semantici.

Vediamoli più in dettaglio.

1.2.2 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. Non è certamente l'obiettivo nè di questo paragrafo, nè della tesi in generale, dare una descrizione esaustiva di cosa si intenda per ontologia e dei problemi che essa comporta (ancorchè ristretti al campo dell'integrazione delle informazioni), ma mi limito a riportare una semplice classicazione delle ontologie (mutuata da Guarino [5, 6]), per inquadrare l'ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate) sono essenzialmente quattro:

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

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

1.2.3 Problemi semantici

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

Se la persona P1 disegna una fonte di informazioni (per esempio DB1) e un'altra persona P2 disegna la stessa fonte DB2, le due basi di dati avranno sicuramente differenze semantiche: per esempio, le coppie sposate possono essere rappresentate in DB1 usando degli oggetti della classe COPPIE, con attributi MARITO e MOGLIE, mentre in DB2 potrebbe esserci una classe PERSONA con un attributo SPOSA.

Come riportato in [3] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma non è l'unica. Le differenze nei sistemi di DBMS possono portare all'uso di differenti modeli per la rappresentazione della porzione di mondo in questione: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate attraverso un modello relazionale, o ad oggetti.

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

1. **eterogeneità tra le classi di oggetti**: benchè due classi in due differenti sorgenti rappresentino lo stesso concetto nello stesso contesto, possono usare nomi diversi per gli stessi attributi, per i metodi, oppure avere gli stessi attributi con domini di valori diversi o ancora (dove questo è permesso) avere regole differenti su questi valori;
2. **eterogeneità tra le strutture delle classi**: comprendono le differenze nei criteri di specializzazione, nelle strutture per realizzare una aggregazione, ed anche le *discrepanze schematiche*, quando cioè valori di attributi sono invece parte dei metadati in un altro schema (come può essere l'attributo SESSO in uno schema, presente invece nell'altro implicitamente attraverso la divisione della classe PERSONE in MASCHI e FEMMINE);
3. **eterogeneità nelle istanze delle classi**: ad esempio, l'uso di diverse unità di misura per i domini di un attributo, o la presenza/assenza di valori nulli.

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

1.3 I wrapper

Come già anticipato nei paragrafi precedenti, la gran varietà di fonti d'informazione disponibili ed alle quali è possibile accedere rende di fatto impossibile la realizzazione di un unico modulo software che permetta l'accesso a queste fonti in maniera semplice ma soprattutto trasparente. Infatti, a sorgenti diverse spesso corrispondono modelli di dati differenti e non sempre compatibili tra loro. Un unico modulo di gestione delle interazioni con tutte le diverse sorgenti sarebbe molto complesso sia in termini di sviluppo che di successiva manutenzione (aggiornamento). Per questi motivi ci si indirizza alla realizzazione di moduli distinti per sorgenti distinte: i wrapper [7, 8].

Il wrapper consiste in un componente che si frappone tra una ben determinata tipologia di sorgenti ed il middleware, ossia la parte di sistema che, in questo caso, si occupa di integrare le diverse sorgenti in un'unica vista globale.

In particolare il wrapper deve garantire alcuni servizi (servizi di wrapping) che permettano al *middleware* di avere una visione dei contenuti della sorgente indipendentemente da come questi sono organizzati nella sorgente stessa o, addirittura, da dove questi siano memorizzati. I servizi di wrapping agiscono da trasduttori tra sorgenti di informazioni esistenti a priori (*legacy information sources*) e servizi di alto livello e devono occuparsi di un ampio range di incompatibilità, poiché le diverse sorgenti di informazione possono avere natura assai diversa. Questo implica il coinvolgimento di interfacce di comunicazione, strutture dati e semantica del programma.

1.3.1 Servizi di wrapping

Questi servizi sono utilizzati per rendere le diverse sorgenti di informazione conformi ad uno standard interno o esterno. Questo standard può comprendere l'interfaccia verso la sorgente di informazioni o il comportamento della sorgente stessa. Alcuni wrapper semplicemente trasformano l'output o l'interfaccia della sorgente, altri modificano il significato o il comportamento di una sorgente o componente software: questo può comportare la creazione di nuove interfacce interne o addirittura l'esposizione ai servizi delle interfacce interne, utilizzando le

informazioni delle sorgenti e/o componenti tradotte. I Servizi di Wrapping Primari includono Comunicazione, Ristrutturazione dei Dati e Comportamento.

Riportiamo qui di seguito la gerarchia dei servizi di wrapping di alto livello, o primari, ed i loro sotto-servizi, unitamente ad una breve descrizione per ciascuno di essi.

W: Wrapping Services

W.1: **Communication Wrapping Service**

W.1.a: Calling Interface
questi servizi sono per la trasformazione dell'interfaccia delle chiamate tra due programmi scritti nel medesimo o in differenti linguaggi.

W.1.b: Event Management
questi servizi sono per la gestione del traffico di eventi tra una sorgente di informazioni ed un servizio di più alto livello incompatibile con essa.

W.1.c: Method and Function Calling
questi servizi gestiscono la trasformazione di chiamate di metodi e funzioni tra una sorgente di informazioni ed un servizio di più alto livello incompatibile.

W.2: ***Data Restructuring Wrapping***

W.2.a: Data Format Restructuring
questi servizi sono per la traslazione tra differenti formati dei dati a livello macchina o file.

W.2.b: Metadata Format Restructuring
questi servizi sono per la trasformazione di formati metadata a livello macchina o file.

W.3: **Behavioral Transformation Wrapping Service**

W.3.a: Application Program Modification
questi servizi sono per la modifica della semantica del programma, come l'aggiunta della precisione nei calcoli; questo spesso coinvolge moduli interni di wrapping parte di un programma applicativo ed "espone" le loro interfacce, così che i moduli selezionati possano essere modificati e/o riutilizzati indipendentemente.

W.3.b: Protocol Modification
questi servizi sono per la modifica dei protocolli del sistema, come i protocolli di concorrenza; questo spesso implica il "wrapping" di un componente interno di un sistema altrimenti chiuso e quindi l'esposizione di un protocollo interno.

W.3.c: Data Language Transformation
questi servizi permettono la traslazione tra differenti linguaggi di manipolazione dati, come tra SQL incompatibili.

Una questione significativa è la separazione tra servizi di wrapping e altri servizi: non è sempre chiaro dove assegnare una determinata funzione. L'esatta separazione è punto fermo di ricerca, ma l'euristica usata in questo documento è che il servizio appartiene alla gerarchia Wrapping Services se la trasformazione *a)* è relativamente a piccola grana e a basso livello dal punto di vista semantico e *b)* produce una traduzione della sorgente di informazione che è probabile venga largamente usata.

Per esempio, se il servizio fornisce un programma che restituisce la radice quadrata di un intero e permette di selezionarne la precisione, allora questo è un Servizio di Wrapping e non un Servizio di Mediazione. D'altro canto, se un servizio fornisce un insieme di valori e produce un'aggregazione complessa per l'esportazione dalla sorgente di informazioni, questo è probabilmente un Servizio di Astrazione ed Aggregazione.

Capitolo 2

Il progetto MOMIS

Il progetto MOMIS (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) [9, 10, 11] è il progetto di un sistema I^3 finalizzato all'integrazione di sorgenti dati strutturate e semi-strutturate. Nasce all'interno del progetto MURST 40% INTERDATA e come collaborazione tra il gruppo di lavoro della Professoressa Sonia Bergamaschi e lo staff della Dottoressa Silvana Castano (del Dipartimento di Scienze dell'Informazione dell'Università di Milano) con loscopo di sviluppare un sistema di integrazione di sorgenti eterogenee di informazione. Questa tesi contribuisce al progetto mediante lo sviluppo di un modulo wrapper per sorgenti dati di tipo strutturato.

In questo capitolo viene fornita una breve descrizione del sistema MOMIS al fine di rendere più chiaro il tipo di interazione del modulo wrapper con il modulo mediatore.

Come è stato visto nel capitolo introduttivo, l'aumento di informazioni disponibili (sia sulle rete Internet, che all'interno di una azienda) ha evidenziato la necessità di avere a disposizione dei sistemi che non solo facilitino il loro reperimento (in quanto questo può avvenire con dei motori di ricerca), ma che diano anche una visione integrata di queste, in modo da eliminare incongruenze e ridondanze necessariamente presenti tra di loro. In genere le architetture di tali sistemi di integrazione di dati sono basate sul concetto di mediatore, un modulo il cui compito è quello di reperire ed integrare informazioni presenti in una molteplicità di basi eterogenee di dati [4]. In letteratura sono già stati presentati diversi sistemi diretti all'integrazione di database convenzionali, come pure applicabili all'integrazione di dati semi-strutturati. Di questi, secondo quanto esposto in [12], si può realizzare una categorizzazione sulla base del diverso tipo di approccio utilizzato, distinguendoli in *semantici* e *strutturali.*

Per quanto riguarda l'approccio *strutturale* esso è caratterizzato dai seguenti punti:

- per trattare tutti i singoli oggetti presenti nel sistema è utilizzato un modello comune dei dati di tipo *self-describing*, rendendo inutili gli schemi concettuali delle sorgenti.
- l'utilizzo di un linguaggio self-describing facilita l'integrazione soprattutto di dati semi-strutturati;
- le uniche informazioni semantiche sono inserite manualmente da un operatore attraverso l'utilizzo di linguaggi descrittivi;
- l'assenza di uno schema globale non permette, soprattutto in caso di database di grandi dimensioni, una ottimizzazione semantica delle interrogazioni;
- l'intero onere dell'integrazione è a carico del progettista del mediatore, in quanto sono eseguibili solo le interrogazioni predefinite dall'operatore.

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

- per ogni sorgente sono a disposizione dei metadati, codificati nello schema concettuale;
- nello schema concettuale sono pure presenti le informazioni semantiche, che possono essere utilmente sfruttate sia nella fase di integrazione delle sorgenti, sia in quella di ottimizzazione delle interrogazioni;
- deve essere presente nel sistema un modello di dati comune per poter descrivere in modo uniforme tutte le informazioni che devono essere condivise;
- è realizzata effettivamente una unificazione (parziale o totale) degli schemi concettuali (in modo semi-automatico), per arrivare alla definizione di uno schema globale.

Per MOMIS si è stabilito di ricorrere, per la rappresentazione dello schema globale, all'adozione del modello ad oggetti standard ODMG-93 [23], esteso per le problematiche di integrazione. Diverse sono le

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

- la presenza di uno schema globale permette all'utente di formulare qualsiasi interrogazione che sia consistente con lo schema;
- la possibilità, usando le informazioni semantiche comprese nello schema globale, di una eventuale ottimizzazione di queste interrogazioni;
- l'uso delle primitive di generalizzazione e di aggregazione, proprie dei modelli ad oggetti, permette la riorganizzazione delle conoscenze estensionali;
- l'adozione di una semantica *type as a set* per gli schemi permette di controllarne la consistenza, facendo riferimento alle loro descrizioni;
- notevoli sforzi sono stati realizzati per lo sviluppo di standard rivolti agli oggetti: CORBA per lo scambio di oggetti attraverso sistemi diversi; ODMG-93 (e con esso i modelli ODM e ODL per la descrizione degli schemi, e OQL come linguaggio di interrogazione) per lo sviluppo di object-oriented database;
- l'adozione di una semantica di *mondo aperto* permette di poter utilizzare tale ambiente anche in presenza di dati semistrutturati: in tal caso gli oggetti di una classe condividono una struttura minima comune (che è quindi la descrizione della classe stessa), ma possono avere ulteriori proprietà non esplicitamente comprese nella struttura della classe di appartenenza.

Per la descrizione degli schemi delle sorgenti e del Mediatore sono stati proposti in MOMIS un modello di dati comune (CORBA) e un linguaggio di definizione comune (ODL_I^3) il quale verrà descritto nel prossimo capitolo. Come vedremo ODL_I^3 costituisce un estensione del corrispondente linguaggio di definizione ODL proposto dal gruppo di standardizzazione ODMG-93.

L'approccio proposto in MOMIS è costituito principalmente di due fasi:

1. **unificazione degli schemi**: è la fase presentata in [11] e [13]. In questa fase l'uso della logica descrittiva OCDL [13], insieme alle tecniche di clustering riportate in [13] ed in [14], permettono la realizzazione di una processo semi-automatico di integrazione degli schemi, fino a pervenire alla definizione di uno schema globale, direttamente interrogabile dall'utente, che rappresenti l'unionedi tutti gli schemi locali, rimuovendone differenze e ripetizioni;
2. **query processing**: in questa fase a partire da una interrogazione dell'utente posta sullo *schema globale*, si giunge alla definizione di un insieme di interrogazione indirizzate alle varie sorgenti, ed alla presentazione di un'unica risposta.

2.1 Architettura

L'architettura del sistema MOMIS è mostrata in figura 2.1. Come si può notare è stata mantenuta la struttura classica di sistema I^3 [2]. Vi si possono individuare quattro livelli:

1. ***sorgenti***: sono al livello più basso e rappresentano le fonti di informazione che devono essere integrate dal sistema. Possono essere sia dei database tradizionali (ad oggetti, o basati sul modello relazionale), che semplici file system, che dati di tipo semi-strutturato;
2. ***wrapper***: rappresentano l'interfaccia tra il mediatore e le varie sorgenti locali di dati, perciò devono essere sviluppati appositamente per il tipo di sorgente che andranno ad interfacciare. I Wrapper hanno una doppia funzione:
   - ?? nella fase di integrazione, essi sono responsabili della descrizione degli schemi delle sorgenti nel linguaggio ODL_{I^3};
   - ?? nella fase di query processing, devono tradurre le query ricevute dal mediatore (espresse nel linguaggio comune di interrogazione OQL_{I^3}) in interrogazioni comprensibili dalla sorgente con cui ognuno di essi opera. Inoltre devono presentare al mediatore, attraverso il modello comune di dati utilizzato dal sistema, i dati ricevuti in risposta alla query presentata

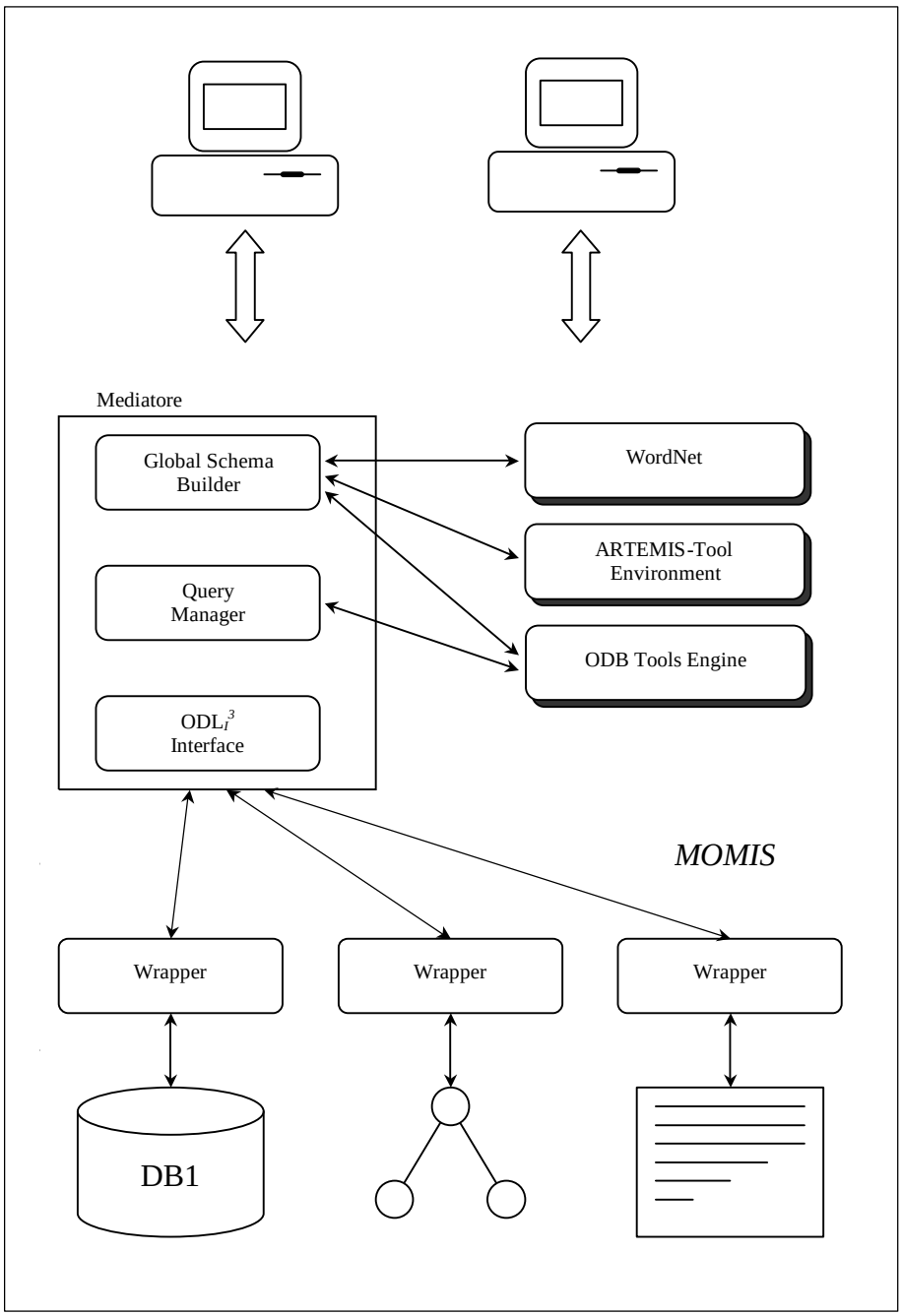


Figura 2.1: Architettura del sistema I^3 MOMIS

3. ***mediatore***: è un modulo software che combina, integra e ridefinisce gli schemi ODL_I^3 ricevuti dai wrappers. Può essere definito il cuore del sistema, ed è costituito da diversi sottomoduli quali:
   - *Global Schema Builder*: è il modulo di integrazione degli schemi locali. Partendo dalle descrizioni delle sorgenti espresse mediante il linguaggio descrittivo ODL_I^3, genera un unico schema globale da presentare all'utente. Questa fase di integrazione, realizzata in modo semi-automatico con l'interazione del progettista del sistema, utilizza gli ODB-Tools, le tecniche di clustering e quelle di fusione delle gerarchie;
   - *Query Manager:* è il modulo che gestisce le interrogazioni. Provvede a gestire la query dell'utente, prima deducendone da essa un insieme di sottoquery da spedire alle fonti locali, poi ricomponendo le informazioni da esse ottenute; inoltre ha il compito di provvedere, utilizzando gli ODB-Tools, all'ottimizzazione semantica delle interrogazioni;
   - ODL_I^3 *interface*: è l'interfaccia che il mediatore usa per interfacciarsi con i wrapper in modo uniforme, utilizzando il modello comune. In questo modo l'onere delle traduzioni (dei dati, delle interrogazioni e delle descrizioni degli schemi) è lasciato ai rispettivi wrappers;
4. ***utente***: è colui che interagisce con il mediatore e al quale viene presentato lo schema globale, nel cui interno sono rappresentati tutti i concetti che possono essere interrogati. In pratica per l'utente è come se si trovasse di fronte ad un unico database da interrogare in modo tradizionale, mentre le sorgenti locali restano completamente trasparenti.

Con il Mediatore cooperano inoltre tre strumenti ausiliari:

- *WordNet*, un database lessicale della lingua inglese, capace di individuare relazioni lessicali e semantiche tra termini [15];
- ARTEMIS-*Tool Environment*, un tool che compie analisi e clustering di schemi [16, 17];
- *ODB-Tools Engine*, un tool basato sulla **OLCD** Description Logics [18, 19] che compie la validazione di schemi e l'ottimizzazione di query [20, 21, 22].

Utilizzando questa architettura, ci si è preposti con MOMIS la realizzazione di un sistema di mediazione che contribuisca a realizzare un *accesso integrato* alle sorgenti di informazione al fine di permettere all'utente di porre una singola query ed ottenere un unica risposta unificata.

Per realizzare l'integrazione degli schemi, sono state arricchite con un componente *intelligente* le tecniche di integrazione basate sul clustering, ampliandole con fasi automatiche realizzate sfruttando gli ODB-Tools, in modo da diminuire il più possibile l'intervento esterno del progettista del sistema.

2.2 Aspetti generali dell'approccio

In questa sezione viene data una breve descrizione dell'approccio all'integrazione intelligente di schemi, al fine di avere una visione generale del comportamento di MOMIS. L'approccio *semantico* utilizzato si articola nei seguenti passi:

- *Generazione di un Thesaurus comune*: l'obiettivo di questa fase è la costruzione di un thesaurus di *relazioni terminologiche* riferite a termini di classi appartenenti a sorgenti diverse. In pratica si cerca di ottenere, attraverso l'utilizzo degli ODB-Tools, in modo semi-automatico (in quanto è richiesta l'interazione col progettista) nuove relazioni di sinonimia e corrispondenza tra i termini (intendendo sia nomi di classi che di attributi) delle diverse sorgenti.
- *Analisi delle Affinità delle classi*: in questa fase le relazioni terminologiche memorizzate nel thesaurus sono utilizzate per valutare l'*affinità* tra classi di una stessa sorgente o di sorgenti diverse. L'affinità tra due classi è stabilita attraverso appropriati *coefficienti di affinità*, che prendono in considerazione sia i nomi delle classi che i loro attributi (considerando per ogni attributo sia il nome che il dominio).
- *Generazione dei clusters*: utilizzando i coefficienti di affinità calcolati nella fase precedente, le classi tra loro affini vengono raggruppate in cluster, usando delle tecniche di clusterizzazione

gerarchica. L'obiettivo di questa fase è di identificare le classi che devono essere integrate, in quanto descrivono le informazioni identiche o semanticamente simili;

?? *Generazione dello schema globale del mediatore*: l'unione di classi affini porta alla creazione dello schema globale del mediatore. Per ogni cluster è definita una classe *globale* che è rappresentativa di tutte le classi appartenti ad esso, ed è caratterizzata dall'unione degli attributi di ogni singola classe. In questo modo lo schema globale sarà composto da tutte queste classi globali e sarà, inoltre, la base per porre le query sui dati delle sorgenti.

Una volta che lo schema globale del mediatore è stato costruito (da tener presente che questa operazione deve essere eseguita solo in fase di inizializzazione del sistema o di acquisizione di una nuova sorgente da integrare), il sistema può essere interrogato; l'utente può porre le proprie query senza doversi preoccupare delle sorgenti dove verranno recuperate le informazioni. Inoltre, l'utente sarà pure esonerato dal conoscere i diversi linguaggi di interrogazione locali delle fonti integrate: è compito del Query Manager ottimizzare le query ricevute e spedirle alle sorgenti che da queste devono essere interrogate.

2.3 Query Reformulation

In questa sezione verrà descritto il processo che, sfruttando le informazioni memorizzate nelle mapping table (struttra dati tabellare che contiene la definizione di una classe globale in ODL$_I^3$ e le informazioni in esse contenute), realizza la Query Reformulation. Scopo di questo processo, che verrà attivato ogniqualvolta l'utente pone una interrogazione sullo schema globale (esprimendola dunque in termini di classi e attributi globali), è arrivare alla definizione automatica delle interrogazioni che devono essere spedite alle diverse fonti di informazione per dare risposta alla query originale.

In accordo con altri approcci *semantici*, questo processo consiste di tre fasi distinte:

- *ottimizzazione semantica globale*: sfruttando le informazioni semantiche presenti a livello di schema globale, ed eventuali regole di integrità definite dal progettista, viene realizzata un'ottimizzazione semantica delle interrogazioni poste dall'utente;
- *individuazione delle sorgenti coinvolte*: analizzando la query vengono individuate le classi globali coinvolte e per ognuna di esse si determina a quali classi sorgenti si deve accedere;
- *formulazione del query plan*: sfruttando le regole di mapping tra rappresentazione globale e schemi locali viene prodotto un insieme di sottoquery direttamente eseguibili sulle sorgenti;
- *ottimizzazione basata sulla Conoscenza Estensionale* (ottimizzazione semantica locale): una volta generate le subquery per ogni sorgente, si può pensare di sfruttare la presenza di vincoli di integrità sugli schemi delle sorgenti, unitamente alle capacità di ODB-Tools, per ridurre ulteriormente il costo di accesso ai dati. Questa opportunità può essere sfruttata purchè siano rappresentate a livello di Mediatore le conoscenze semantiche relative agli schemi locali.

Ottimizzazione Semantica Globale

In questa fase il mediatore MOMIS opera sulla query dell'utente sfruttando le tecniche di ottimizzazione semantica supportate dagli ODB-Tools, al fine di ridurre il costo del piano di accesso che sarà generato. La query è rimpiazzata da una nuova la quale contiene ogni possibile restrizione che non è presente nella query originale ma che è logicamente implicata da essa stessa e dalle regole definite sullo schema, per poter poi sfruttare eventuali indici ausiliari presenti nelle sorgenti locali.

Per questa fase di ottimizzazione sono necessarie le regole di integrità definite sullo schema globale; anche se questa ipotesi può sembrare molto forte (infatti le regole sono definite dal progettista, il quale deve essere a conoscenza di condizioni valide su tutte le estensioni di tutte le classi appartenenti al cluster da cui deriva una determinata classe globale) non è improbabile che in determinati domini applicativi (facendo riferimento a databases appartenenti a domini fortemente regolamentati) la definizione di queste regole sia possibile.

Formulazione del Query Plan

Una volta che il sistema MOMIS ha prodotto la query eventualmente ottimizzata, deve essere costruito un piano di accesso, il quale consiste nel formulare un insieme di subquery da fornire alle sorgenti d'informazione locale, per andare a recuperare le informazioni da dare in risposta all'utente; si rimanda alla Sezione 2.4 per la presentazione della fase di riunificazione dei dati.

Si supponga che la query sia stata posta facendo riferimento ad una classe globale, alla quale corrisponderà in memoria una mapping table: per ogni classe appartenente al cluster da cui la dasse globale ha avuto origine, deve essere formulata una apposita interrogazione, che ne utilizzi la terminologia adeguata. In particolare, il sistema deve riformulare la query di partenza, per ogni classe, esprimendola utilizzando i termini della classe locale che si sta considerando.

In questa fase il sistema opera un'ulteriore ottimizzazione eliminando dalla lista di classi da interrogare quelle che forniscono unicamente dati non utili o non verificabili, sulla base di un'analisi dei valori contenuti nella mapping table.

Ottimizzazione basata sulla Conoscenza Estensionale

Fino ad ora all'interno di MOMIS sono state analizzate informazioni inerenti le intensioni, ovvero le strutture delle classi che devono essere integrate (siano esse informazioni semantiche o terminologiche). Potrebbero essere sfruttate anche eventuali informazioni riguardanti le estensioni di queste classi fino ad ora escluse dalla trattazione. Infatti esse potrebbero essere fornite in due modi:

- manuale
- automatico

Nel primo caso il progettista dovrebbe controllare fisicamente tutte le estensioni di tutte le classi coinvolte nel sistema, per vedere se vi sono tra loro dati duplicati, o in qualche modo correlati: ciò comporterebbe un enorme mole di lavoro.

Nel secondo caso, invece, bisognerebbe ideare un algoritmo di controllo e confronto dei dati memorizzati nelle varie sorgenti (si pensi ad esempio che questo confronto dovrebbe essere riattivato successivamente

ad ogni operazione di aggiornamento, o di inserimento, …), il che non è comunque un compito banale.

Vi sono però casi in cui, magari grazie a conoscenze a priori (può essere il caso di più database derivati da un'unica sorgente di partenza) o a un confronto tra i progettisti stessi delle sorgenti locali, si può supporre che sia possibile definire relazioni che intercorrono tra le estensioni delle classi che sono state integrate dal mediatore. Per questi casi, sono state definite quattro tipologie di proprietà *inter-schema* atte a esprimere mutue relazioni esistenti tra le estensioni: equivalenza tra classi, contenimento di una classe in un'altra, intersezione tra classi, disgiunzione di due classi.

2.4 Unificazione dei dati

Una volta ricevuti i dati richiesti dalle sorgenti, in risposta alle varie query locali a loro spedite, si pone il problema di come riorganizzare queste informazioni per presentarle all'utente.

Posto in altri termini, il problema è il riconoscimento automatico di tutte le informazioni che appartengono ad uno stesso *oggetto globale*. Dal punto di vista teorico, la soluzione ideale sarebbe la presenza di una *chiave universale*, o di un *oid* comune, condivisa da tutte le sorgenti (o almeno da tutte le sorgenti che fanno parte del sistema) che individui univocamente al loro interno tutti gli oggetti. E' però impensabile che questo sia realmente realizzabile, dal momento che, anche nel migliore dei casi, questo identificatore sarebbe comunque valido e unico solo all'interno di un contesto limitato (si può per esempio prendere il codice fiscale, che perde però significato all'esterno di una determinata nazione…). Un'altra soluzione potrebbe essere il riunificare le risposte ricevute non sulla base di un oid universale, ma sulla base delle chiavi locali (quando dichiarate) delle singole sorgenti, ma il problema rimane in altri termini:

- supponendo che esista per esempio una chiave nome in tutte le tabelle interrogate, niente ci assicura che persone con nomi uguali siano effettivamente la stessa persona in contesti diversi;
- supponendo invece che come chiave interna delle tabele si utilizzi un codice (pensiamo ad un esempio magazzino), il problema è più complicato: è molto probabile che si usino codici diversi per

identificare lo stesso concetto, pur se il dominio degli attributi codice è uguale in tutte le sorgenti.

E' quindi un problema aperto, la cui unica soluzione, fino a questo momento, sembra essere lo spostare questo tipo di decisione al progettista rimettendo a lui questa responsabilità. E' allora ragionevole pensare che, congiuntamente alla definizione di proprietà interschema (in cui magari si definisce che due sorgenti hanno gli stessi dati, o dati parzialmente duplicati tra loro), sia compito del progettista indicare esplicitamente il campo (o l'insieme di campi) che dovrà essere utilizzato dal sistema per riunire le informazioni che fanno riferimento ad una unica entità.

Capitolo 3

Tecnologie, linguaggi e standard

3.1 CORBA

Nell'era di Internet e delle grandi Intranet aziendali, il modello computazionale distribuito è chiaramente dominante. Un tipico ambiente distribuito vede la presenza di mainframe, server Unix e macchine Windows e pone quindi seri problemi di interoperabilità tra piattaforme, sistemi operativi e linguaggi differenti. Ecco quindi nascere la necessità di un software la cui tecnologia ci permetta di astrarre un applicativo dall'ambiente in cui viene utilizzato: il middleware. A dieci anni dalla sua prima pubblicazione, CORBA rimane il middleware più completo ed affidabile per la realizzazione di applicazioni distribuite.

L'acronimo CORBA significa **C**ommon **O**bject **R**equest **B**roker **A**rchitecture e quindi non rappresenta un prodotto in particolare, bensì un'insieme di specifiche volte a definire un'architettura possibilmente completa e standardizzata. Lo scopo dichiarato delle specifiche CORBA [23, 24] è proprio quello di definire un'infrastruttura standard per la comunicazione tra e con oggetti remoti ovunque distribuiti, indipendentemente dal linguaggio usato per implementarli e dalla piattaforma di esecuzione. Ribadiamo che a differenza di altre tecnologie distribuite non si sta parlando di una tecnologia legata ad una specifica piattaforma, ma di uno standard indipendente dal linguaggio adottato che consente per esempio ad oggetti Java di comunicare con altri oggetti sviluppati in altri linguaggi quali COBOL, C++ o altri ancora.

Le specifiche sono pubblicate dall' Object Managment Group (OMG) [25], un consorzio di oltre 800 aziende che include i più illustri marchi dell'industria informatica. Le specifiche CORBA 1.1 sono state pubblicate da OMG nell'autunno del 1991 e definivano un'Application Program Interface (API) ed un linguaggio descrittivo per la definizione

delle interfacce degli oggetti CORBA, l'Interface Definition Language (IDL). In questa prima versione delle specifiche è stato dato maggior peso alla portabilità delle soluzioni piuttosto che ai dettagli implementativi, lasciando alle ditte produttrici dei diversi ORB la possibilità di scelta di questi ultimi. Soltanto nel Dicembre del 1994 con CORBA 2.0 sono stati definiti i dettagli della comunicazione tra differenti implementazioni di ORB. Questo mediante la definizione dei protocolli General Inter-ORB Protocol (GIOP) e successivamente Internet Inter -ORB Protocol (IIOP).

Sebbene CORBA possa essere utilizzato con la maggior parte dei linguaggi di programmazione, Java appare essere un linguaggio privilegiato per implementare le sue specifiche in un ambiente eterogeneo in quanto permette agli oggetti CORBA di essere eseguiti indifferentemente su qualunque piattaforma su cui sia possibile far "girare" una JVM (Java Virtual Machine): mainframe, network computer, telefoni cellulare...

Nello sviluppo di applicazioni distribuite Java e CORBA si completano a vicenda: CORBA affronta e risolve il problema della trasparenza della rete, Java quello della trasparenza dell'implementazione rispetto alla piattaforma di esecuzione.

3.1.1 Object Management Group

L'OMG è un consorzio no-profit interamente dedicato alla produzione di specifiche e di standard , la cui attività cominciò nel 1989 con soli 8 membri tra cui Sun Microsystems, Philips, Hewlett-Packard e 3Com. Le specifiche più conosciute prodotte dal consorzio sono sicuramente UML e CORBA che comunque, nella logica OMG, rappresentano strumenti strettamente cooperanti nella realizzazione di applicazioniEnterprise OO.

Lo scopo di OMG è principalmente quello di produrre e mantenere una suite di specifiche di supporto per lo sviluppo di software distribuito, coprendo l'intero ciclo di vita di un progetto: analisi, design, sviluppo, run-time e manutenzione.

Con lo scopo di ridurre complessità e costi di realizzazione di nuove applicazioni distribuite, il consorzio ha definito un framework: Object Management Architecture (OMA). OMA è il centro di tutte le attività del consorzio, al cui interno convivono tutte le tecnologie OMG. Nell'ottica OMG la definizione di un framework prescinde dall'implementazione e si

"limita" alla definizione dettagliata delle interfacce di tutti i componenti individuati.

I componenti OMA sono:

?? Object Request Broker (ORB): è l'elemento fondamentale dell'architettura: è il canale che fornisce l'infrastruttura che permette agli oggetti nell'ambiente distribuito di comunicare tra loro. Questo indipendentemente dal linguaggio e dalla piattaforma adottata. La comunicazione tra tutti i componenti OMA è sempre mediata e gestita dall'ORB.

?? Object Services: standardizzano la gestione e la manutenzione del ciclo di vita degli oggetti e sono indipendenti dal singolo dominio applicativo. Forniscono le interfacce base per la creazione e l'accesso agli oggetti e possono essere usati da più applicazioni distribuite (Es. Naming e Trading Service).

?? Common Facilities: comunemente conosciute come CORBA Facilities. Forniscono due tipologie di servizi, orizzontali e verticali. Quelli orizzontali sono funzionalità applicative comuni quali gestione stampe, gestione documenti, database e posta elettronica, mentre quelli verticali sono invece destinati ad una precisa tipologia di applicazioni (Es. telecomunicazioni).

?? Domain Interfaces: possono combinare common facilities ed object services e forniscono funzionalità altamente specializzate per ristretti domini applicativi.

?? Application Objects: è l'insieme di tutti gli altri oggetti sviluppati per una specifica applicazione: sono gli oggetti applicativi. Non è un'area di standardizzazione OMG.

Di ciascuno di questi componenti OMG fornisce una definizione formale, definendone sia le interfacce (mediante IDL), sia la semantica. La definizione mediante interfacce lascia ampio spazio al mercato di componenti software di agire sotto le specifiche con differenti implementazioni e consente la massima interoperabilità tra componenti di vendor differenti.

3.1.2 I servizi CORBA

I CORBA Services sono una collezione di servizi system-level descritti dettagliatamente mediante un'interfaccia IDL e sono destinati a completare ed estendere le funzionalità fornite dall'ORB. Forniscono un supporto che va a coprire lo spettro completo delle esigenze di una qualunque applicazione distribuita.

Alcuni dei servizi standardizzati da OMG (ad esempio il Naming Service), sono diventati fondamentali nella programmazione CORBA e sono presenti in tutte le implementazioni. Altri servizi appaiono invece meno interessanti nella pratica comune e assumono un significato solo dinanzi ad esigenze particolari. Non sono presenti nella maggior parte delle implementazioni sul mercato.

OMG ha pubblicato le specifiche di ben 15 servizi:

1. Collection Service: fornisce meccanismi di creazione ed utilizzo per le più comuni tipologie di collection.
2. Concurrency Control Service: definisce un lock manager che fornisce meccanismi di gestione dei problemi di concorrenza nell'accesso ad oggetti agganciati all'ORB.
3. Event Service: fornisce un event channel che consente ai componenti interessati ad uno specifico evento di ricevere una notifica pur non conoscendo nulla del componente generatore.
4. Externalization Service: definisce meccanismi di streaming per il trattamento di dati da e verso i componenti.
5. Licensing Service: fornisce meccanismi di controllo e verifica dell'utilizzo di un componente. (pensato per l'implementazione di politiche "pago per quel che uso")
6. Lyfe Cycle Service: definisce le operazioni necessarie a gestire il ciclo di vita di un componente sull'ORB (creare, copiare e rimuovere).
7. Naming Service: consente ad un componente di localizzare risorse (componenti o altro) mediante un nome univoco ed inoltre permette di interrogare sistemi di directory e naming già esistenti (NIS, NDS, X.500, DCE, LDAP). E' il servizio più utilizzato.

8. Persistence Service: mediante un'unica interfaccia fornisce le funzionalità necessarie alla memorizzazione di un componente su più tipologie di server (ODBMS, RDBMS e file system).
9. Properties Service: permette la definizione di proprietà legate allo stato di un componente.
10. Query Service: fornisce meccanismi di interrogazione basati su un'estensione di SQL chiamata Object Query Language.
11. Relationship Service: permette di definire e verificare in modo dinamico varie tipologie di associazioni e relazioni tra componenti (associazioni dinamiche e contenimento).
12. Security Service: è un framework per la definizione e la gestione della sicurezza in ambiente distribuito. Copre ogni possibile aspetto: autenticazione, definizione di credenziali e gestione per delega, definizione di access control list e non-repudiation.
13. Time Service: definisce un'interfaccia di sincronizzazione tra componenti in ambiente distribuito.
14. Trader Service: fornisce un meccanismo modello "Yellow Pages" per i componenti.
15. Transaction Service: fornisce un meccanismo di two-phase commit sugli oggetti agganciati all'ORB che supportano il rollback, definisce transazioni flat o innestate.

3.1.3 Le basi CORBA

CORBA ed OMA in genere si fondano su alcuni fondamentali principi di design:

- separazione tra interfaccia ed implementazione: un client è legato all'interfaccia di un oggetto CORBA, non alla sua implementazione.
- location transparency ed access transparency: l'utilizzo di un qualunque oggetto CORBA non presuppone alcuna conoscenza sulla sua effettiva localizzazione.

?? typed interfaces: ogni riferimento ad un oggetto CORBA ha un tipo definito dalla sua interfaccia.

Parlando di CORBA, è particolarmente significativo il concetto di trasparenza, inteso sia come location transparency, sia come trasparenza del linguaggio di programmazione adottato. La location transparency è garantita dalla mediazione dell'ORB.

3.1.4 Architettura CORBA

L'architettura CORBA ruota intorno al concetto di Objects Request Broker (ORB). L'ORB è il servizio che gestisce la comunicazione in un ambiente distribuito agendo da intermediario tra gli oggetti remoti: individua l'oggetto sulla rete, comunica la richiesta all'oggetto, attende il risultato e lo restituisce al client. L'ORB opera in modo tale da nascondere al client tutti i dettagli sulla effettiva localizzazione degli oggetti sulla rete e sul loro linguaggio d'implementazione; è quindi l'ORB ad individuare l'oggetto interessato da una richiesta e ad effettuare le opportune traslazioni nel linguaggio d'implementazione. Queste traslazioni sono possibili solo per quei linguaggi per cui è stato definito un mapping verso IDL (questa definizione è stata operata per i linguaggi più comuni tra cui Java).

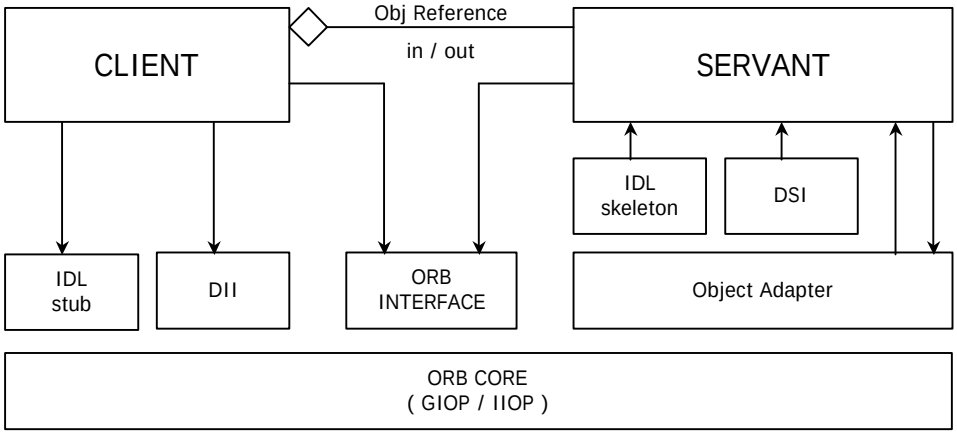


Figura 3.1: Architettura CORBA

In figura 3.1 si può osservare l'architettura CORBA nel suo complesso:

- Object: è l'entità composta da identity, interface ed implementation (Servant nel gergo CORBA).
- Servant: è l'implementazione dell'oggetto remoto. Definisce i metodi specificati dall'interfaccia in un linguaggio di programmazione.
- Client: è l'entità che invoca i metodi (operation nel gergo CORBA) del Servant. L'infrastruttura dell'ORB opera in modo tale da rendere trasparenti i dettagli sulla comunicazione remota.
- ORB: l'ORB è l'entità logica che fornisce i meccanismi per comunicare, in modo trasparente, le richieste da un client all'oggetto remoto. Le chiamate del client sono assimilabili a semplici invocazioni locali, questo grazie all'operato dell'ORB che svincola completamente il client dai dettagli di comunicazione.
- ORB Interface: essendo un'entità logica, l'ORB può essere implementato in molti modi. Le specifiche CORBA definiscono l'ORB mediante un'interfaccia astratta, nascondendo completamente alle applicazioni i dettagli d'implementazione.
- IDL stub e skeleton: lo stub opera da collante tra client ed ORB; lo skeleton ha la stessa funzione per il server. Stub e skeleton sono generati nel linguaggio adottato da un compilatore apposito che opera partendo da una definizione IDL.
- Dynamic Invocation Interface (DII): è l'interfaccia che consente ad un client di inviare dinamicamente una request ad un oggetto remoto senza conoscerne la definizione dell'interfaccia e senza avere un legame con lo stub. Consente inoltre ad un client di effettuare due tipi di chiamate asincrone: deferred synchronous (separa le operazioni di send e di receive) e oneway (solo send).
- Dynamic Skeleton Interface (DSI): è l'analogo lato server del DII. Consente ad un ORB di recapitare una request ad un oggetto che non ha uno skeleton statico, per cui cioè non è stato definito precisamente il tipo a tempo di compilazione. Il suo utilizzo è totalmente trasparente per un client.

?? Object Adapter: assiste l'ORB nel recapitare le request ad un'implementazione valida e nelle operazioni di attivazione/disattivazione degli oggetti. Il suo compito principale è quello di legare l'implementazione di un oggetto all'ORB.

3.1.5 Invocazione CORBA

Utilizzando l'ORB, un client, può inviare una Request in modo trasparente ad un oggetto CORBA che risieda sulla stessa macchina o ovunque sulla rete. Per raggiungere questo livello di astrazione, ogni oggetto remoto è dotato di uno stub e di uno skeleton; questi due elementi agiscono rispettivamente da collante tra client ed ORB e tra ORB ed oggetto CORBA.

Il client quindi invoca i metodi non sull'oggetto remoto, bensì sul suo stub locale. L'effettiva invocazione remota viene operata dallo stub. Lo stub effettua la traslazione dei data types dal linguaggio di programmazione client-side ad un generico formato CORBA; quest'ultimo è convogliato via rete dal messaggio di Request. In maniera speculare allo stub, la ritraduzione dei dati è eseguita sul server dallo skeleton. In questo caso il formato della Request viene traslato nel linguaggio di programmazione server-side.

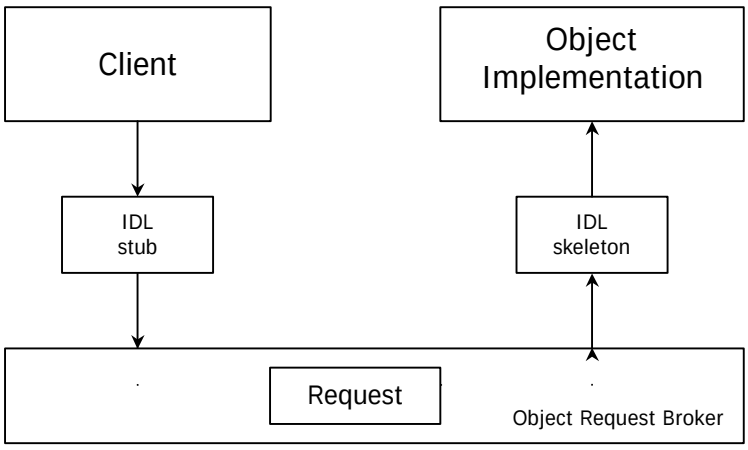


Figura 3.2: Invocazione di un metodo da client ad oggetto CORBA

Stub e skeleton vengono generati automaticamente da un compilatore a partire dalla definizione IDL dell'oggetto CORBA

3.1.6 Interoperabilità tra ORB

Le specifiche CORBA 1.1 si limitavano a fornire le basi per la portabilità di oggetti applicativi e non garantivano affatto l'interoperabilità tra differenti implementazioni di ORB. Le specifiche 2.0 colmarono questa significativa lacuna con la definizione di un protocollo (GIOP), espressamente pensato per interazioni ORB-to-ORB. Il General Inter-ORB Protocol specifica un insieme di formati di messaggi e comuni rappresentazioni dati per la comunicazione tra ORB.

Vista la diffusione di TCP/IP, comunemente viene usato l'Internet Inter-ORB Protocol (IIOP) che specifica come i messaggi GIOP vengono scambiati su TCP/IP. IIOP viene considerato il protocollo standard CORBA e quindi ogni ORB deve connettersi con l'universo degli altri ORB traslando le request sul e dal backbone IIOP.

GIOP definisce un formato multi ORB di riferimento ad un oggetto remoto, l'Interoperable Object References (IORs). L'informazione contenuta e specificata dalla struttura dello IOR assume significato indipendentemente dall'implementazione dell'ORB, consentendo ad un'invocazione di transitare da un ORB ad un altro. Ogni ORB, fornisce un metodo *object_to_string* che consente di ottenere una rappresentazione stringa dello IOR di un generico oggetto.

3.1.7 Interface Definition Language

In un middleware distribuito tutti gli oggetti, compresi quelli di infrastruttura, sono trattati come interfacce. Questo è sia una valida scelta di design, sia un'esigenza di distribuzione: un client tipicamente non conosce e non deve avere la necessità di conoscere l'implementazione di un oggetto destinato ad essere eseguito su una macchina server. Ciò consente ad esempio il dialogo tra oggetti Java e procedure C++ che per

natura probabilmente risiederanno addirittura su macchine ad architetture differenti.

CORBA fornisce una chiara separazione tra ciò che è relativo all'interfaccia di un oggetto e la sua implementazione. Il client non si deve occupare in modo diretto dei dettagli di implementazione, ma solo dell'interfaccia implementata dall'oggetto che intende utilizzare.

Poiché CORBA è trasparente rispetto al linguaggio, OMG ha definito nelle sue specifiche un nuovo linguaggio interamente descrittivo (IDL), destinato alla definizione delle interfacce degli oggetti CORBA. In momenti successivi sono stati definiti i differenti mapping tra i vari linguaggi di programmazione ed IDL. E' da notare che in molti dei linguaggi utilizzabili con CORBA non esiste il concetto di interfaccia (ad esempio C).

Un oggetto remoto quindi, indipendentemente dal fatto che sia applicativo o appartenente all'infrastruttura (l'ORB, i servizi, …), per essere utilzzato in un middleware CORBA, deve essere in primo luogo definito mediante IDL: nel caso di un oggetto applicativo la definizione sarà a carico dello sviluppatore, nel caso di un oggetto di infrastruttura ci viene fornita da OMG.

In figura 3.3 possiamo vedere, ad esempio, parte della definizione IDL dell'ORB.

```
// IDL
module CORBA {

   interface ORB {

      string object_to_string (in Object obj);
      Object string_to_object (in string str);
      Object resolve_initial_references (in ObjectId
      identifier) raises (InvalidName);
      // ecc…

   };

};
```

Figura 3.3: Parte della definizione IDL dell'ORB

La definizione IDL di un oggetto permette di specificare solo gli aspetti relativi alla sua interfaccia. Si potranno quindi definire: le signature dei metodi, le eccezioni che questi rilanciano, l'appartenenza ai package, costanti e strutture dati manipolate dai metodi.

Data la definizione IDL, sarà necessario utilizzare il compilatore fornito a corredo dell'ORB. Dalla compilazione si otterranno un buon numero di file .java, fra cui stub, skeleton ed altri contenenti codice di supporto per l'aggancio all'ORB. A partire dai file generati, sarà possibile realizzare l'opportuna implementazione Java.

3.2 Il linguaggio ODL$_I^3$

Il linguaggio ODL$_I^3$ è il linguaggio di definizione attraverso il quale i wrapper comunicano al mediatore (ed in particolare al Global Schema Builder) le descrizioni delle sorgenti da loro servite. Punto di partenza per la definizione di questo linguaggio, è stato l'attenersi alle raccomandazioni della proposta di standardizzazione per linguaggi di mediazione[26]. Parallelamente a questa proposta (secondo la quale i diversi sistemi di mediazione avrebbero potuto supportare sorgenti con modelli complessi, come quelli ad oggetti, e sorgenti più semplici, come file di strutture), si è cercato di discostarsi il meno possibile dal linguaggio ODL, a sua volta proposto dal gruppo di standardizzazione ODMG-93 [23, 24].

Le caratteristiche peculiari di ODL, al pari di altri linguaggi basati sul paradigma ad oggetti, possono essere così riassunte:

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

Il linguaggio ODL rappresenta il punto di partenza nel progetto di integrazione; pur essendo, infatti, ben progettato per rappresentare la conoscenza relativa ad un singolo schema ad oggetti, è certamente incompleto se calato in un contesto di integrazione di basi di dati eterogenee quale è quello descritto in questa tesi. Si è reso pertanto necessario definirne un'estensione, denominata ODL_{I^3}, in accordo con le raccomandazioni della proposta di standardizzazione per i linguaggi di mediazione, risultato del lavoro di workshop I^3 svoltosi presso l'università del Maryland [26].

Tuttavia, partendo da questa proposta, secondo cui i diversi sistemi di mediazione dovrebbero poter supportare sorgenti con modelli complessi (come quelli ad oggetti) e modelli più semplici (come file di strutture), si è comunque cercato di discostarsi il meno possibile dal linguaggio ODL. I propositi raggiunti dall'estensione a ODL che hanno portato al linguaggio ODL_{I^3} sono i seguenti:

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

capacità nei controlli di consistenza e nell'ottimizzazione semantica delle interrogazioni.

E' da notare come nella fase di progettazionesia compito dei wrapper eliminare le eterogeneità legate al tipo di sorgenti e alla sintassi con cui sono definiti, traducendo la descrizione degli schemi, siano essi relazionali, ad oggetti, semistrutturati, etc.... in un unico linguaggio descrittivo comune. Utilizzando questo linguaggio, sono fornite dal wrapper al mediatore le descrizioni di tutte le classi da integrare: le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema e quindi rendere interrogabili. Non è detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente: ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa.

Poiché nel sistema MOMIS si è scelto di descrivere i sorgenti basandosi sul paradigma a oggetti, indipendentemente dal modello originale adottato da ogni singolo database sorgente, ogni entità verrà descritta dal relativo wrapper utilizzando sempre il concetto di classe.

Le sintassi dei linguaggi ODL ed ODL$_I^3$ sono riportate in BNF rispettivamente in appendice B e C.

3.3 Il linguaggio OQL$_I^3$

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

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

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

Il linguaggio OQL (Object Query Language) è un linguaggio d'interrogazione ad oggetti avente una sintassi eccezionalmente chiara e potente simile all'SQL, da cui peraltro deriva: per molti aspetti è un'estensione ad oggetti di SQL (l'unica vera innovazione rispetto a SQL è l'istruzione "define extent", che definisce una variabile contenente tutte le istanze di una classe).

Questo linguaggio è estremamente versatile ed espressivo perciò se da un lato è necessario uno sforzo maggiore nello sviluppo di moduli per l'interpretazione e gestione delle interrogazioni (implementando le funzionalità proprie di un ODBMS), dall'altro si ha la possibilità di sfruttare al meglio le informazioni rappresentate nello schema globale. Le inevitabili complicazioni a livello implementativo sono, pertanto, ampiamente giustificate da una maggiore versatilità e facilità d'uso per l'utente (si impiega infatti un linguaggio standard e non un formalismo ad-hoc). Le principali caratteristiche di questo linguaggio sono:

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

infatti anche le primitive per gestire, con la stessa efficienza, altri tipi strutturati come array, liste e strutture.

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

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

Capitolo 4

Il wrapper Access/ODL$_I^3$

In questo capitolo vengono descritti la struttura ed il funzionamento del wrapper Access/ODL$_I^3$. Il wrapper Access è un modulo, come già ampiamente accennato nei capitoli precedenti, che si occupa di rendere accessibili al sistema MOMIS le sorgenti dati che, in questo caso, rispondono al formato Access (.mdb). Access è un applicativo Microsoft parte della suite Office ed è stato deciso di svilupparne un wrapper specifico in quanto la diffusione di questo software è sufficientemente ampia da giustificare lo sforzo.

Per poter meglio seguire i passaggi che porteranno dalla sorgente dati .mdb alla corrispondente descrizione ODL$_I^3$ introduciamo nel paragrafo seguente un semplice esempio di db Access: University. Vedremo poi nei paragrafi seguenti la struttura del wrapper e come utilizzarlo sulla sorgente dati per ricavarne la descrizione ODL$_I^3$. Infine verranno accennati i problemi di maggior rilievo incontrati durante lo sviluppo del wrapper e come siano stati risolti.

4.1 Esempio di riferimento

Quest'esempio si riferisce alle definizioni degli schemi delle sorgenti presentati nell'Appendice C in ODL$_I^3$. Per comodità esso è rappresentato in modo schematico in Figura 4.1.

L'esempio si riferisce ad una semplice realtà universitaria: la sorgente da integrare, University, è un database di tipo relazionale, che contiene informazioni sullo staff e sugli studenti di una determinata università. E' costituita da sei tabelle: Research Staff, School Member, Department, Section, Room e Anag.

```
Sorgente University:

Research Staff(First Name, Last Name, Relation, e-mail,
               Dept Code, Course Code)
 primary key(First Name, Last Name),
 foreign key(Dept Code) references Department,
 foreign key(Course Code) references Course,
 foreign key(First Name, Last Name)
   references Anag,

School Member(First Name, Last Name, Faculty, Year)
 primary key(First Name, Last Name),
 foreign key(First Name, Last Name)
   references Anag,

Department(Dept Name, Dept Code, Budget, Dept Area)
 primary key(Dept Code),

Course(Course Name, Course Code, Length, Room Code)
 primary key(Course Code),
 foreign key(Room Code) references Room,

Room(Room Code, Seats Number, Notes)
 primary key(Room Code),

Anag(First Name, Last Name, Address, City, e-mail)
 primary key(First Name, Last Name),
```

Figura 4.1: Esempio di riferimento

Per ogni professore (presente nella tabella Research Staff), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key Dept Code) e sul corso da lui tenuto (Course code). Per gli studenti presenti nella tabella School Member sono archiviati il nome (nella coppia First Name e Last Name), la facoltà di appartenenza (Faculty) e l'anno di corso (Year). Nella tabella Department sono descritti, per ogni dipartimento, oltre al nome (Dept Name) ed al codice (Dept Code) il budget (Budget) che ha a disposizione e l'area (Dept Area) a cui appartiene (scientifica, economica, …). La tabella Course contiene i dati

riguardanti ciascun corso: il nome (Course Name), il codice (Course Code), la durata (Length) ed il numero dell'aula in cui viene tenuto (Room Code), mentre la tabella Room contiene dati riguardanti le aule. La tabella Anag contiene ulteriori dati personali degli studenti e dei professori: indirizzo (Address), città di residenza (City) ed indirizzo di posta elettronica (e-mail). Per ogni tabella del database sono inoltre indicate le chiavi primarie e le eventuali foreign key con le tabelle che queste referenziano. Tabelle e relazioni sono inoltre riassunte in forma grafica in Figura 4.2

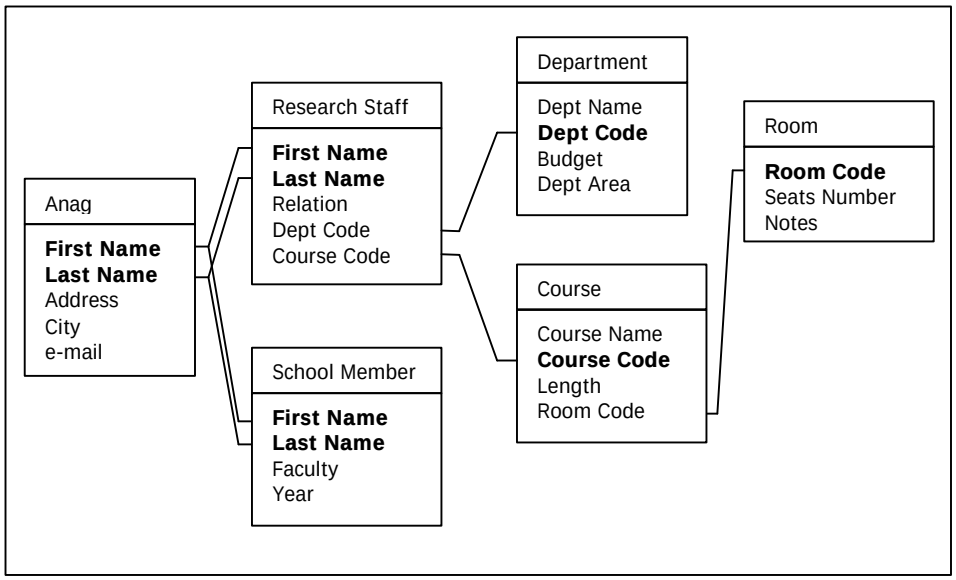


Figura 4.2: Tabelle e relazioni

4.2 Struttura del wrapper

Il wrapper è stato sviluppato utilizzando il linguaggio di programmazione Java (versione 1.2) per mantenere la compatibilità con i diversi moduli MOMIS già sviluppati.

Il wrapper è sostanzialmente costituito da cinque classi Java: Wrapper_Server, Wrapper_Client, TableDesc, TabellaDiMapping e Utility.

Wrapper_Server è la classe che costituisce il cuore del wrapper. Questa classe si occupa di creare un'istanza del wrapper e di registrarla con l'ORB e con il servizio di risoluzione dei nomi. Dopo la prima fase di registrazione, il wrapper si pone in attesa delle richieste dei servizi.

Wrapper_Client è la classe che costituisce un client per l'interrogazione del server. Questa classe permette di verificare le varie funzionalità del server.

TableDesc, TabellaDiMapping e Utility sono tre classi ausiliarie a Wrapper_Server. In particolare ad ogni istanza della prima è associata una tabella della sorgente di dati. In ogni istanza, oltre al nome, troviamo la lista delle chiavi della tabella, la lista delle eventuali Foreign Keys e la lista delle definizioni degli attributi.

TabellaDiMapping è una classe creata come conseguenza alla necessità del parsing dei nomi. Nel paragrafo 4.4 riprenderemo la questione.

La classe Utility contiene alcuni metodi di utilità per le classi ora viste.

Vediamo ora in dettaglio le varie classi co i rispettivi metodi.

Classe Wrapper_Server

PROPRIETA'

?? *orb*: quest'oggetto ORB è l'orb attraverso il quale avviene la comunicazione tra il wrapper ed il mediatore;

?? *_connection*: quest'attributo rappresenta la connessione del wrapper con il database di cui deve gestire le informazioni.

?? *_dbmd*: è un insieme di informazioni (MetaData) relative al database; da questo insieme di informazioni possiamo estrarre la lista delle tabelle nel db, i dati relativi ad ogni tabella (indici, chiavi primarie, nomi delle colonne, …). E' la "fonte" principale e più ricca di informazioni riguardanti l db esaminato;

?? *_tdmd*: è un'istanza della classe **TabellaDiMapping** utilizzata per il parsing dei nomi; è una struttura tabellare contenente, per ogni riga,

una coppia nome originale – nome tradotto. Quest'ultimo è ottenuto grazie al metodo parseString della classe **Utility**.

?? *_tableList*: è un vettore che contenente la lista delle tabelle del database ed è compilato dal metodo *getTableDesc*. E' una variabile membro privata.

METODI

?? *getSourceName*: è il metodo che restituisce il nome della sorgente interrogata; non richiede parametri e restituisce una stringa contenente il nome;

?? *getType*: restituisce il tipo di sorgente; restituisce il valore costante "JDBC";

?? *getDescription*: questo metodo restituisce la descrizione in ODLi3 della sorgente; non richiede parametri e restituisce una stringa contenente la descrizione;

?? *runQuery*: è il metodo che permette di eseguire le query sul db; come unico parametro richiede una stringa contenente la query. Restituisce un insieme di dati sotto forma di MomisResultSet contenente i risultati della query;

?? *getTableDesc*: è un metodo privato. E' importante citarlo in quanto permette di ottenere la descrizione della tabella il cui nome gli è passato come parametro (stringa). Ad ogni invocazione, questo metodo crea una nuova istanza della classe **TableDesc**, la compila con tutti i dati rappresentativi della tabella (nome, chiavi primarie e foreign key, attributi e loro tipo) ed infine aggiorna la lista *_tableList.*

Classe Wrapper_Client

PROPRIETA'

?? *myRef*: è un oggetto di tipo Wrapper che rappresenta il server cui vogliamo connetterci;

METODI

?? *main*: è l'unico metodo della classe e si occupa di interpretare pochi semplici comandi che danno la possibilità di interrogare la base di dati "simulando" quelle che potrebbero essere le richieste del server: richieste di descrizione, del nome e del tipo di sorgente ed esecuzione di query. Costituisce l'interfaccia tra l'utente e il **Wrapper_Server**

Classe TabellaDiMapping

METODI

?? *setName*: è il metodo che permette di aggiungere un nome alla tabella dei nomi. Prevede, come parametro, una stringa contenente il nome da inserire: dopo aver verificato che nella tabella non esista già, questo viene "normalizzato" e quindi inserito. Il metodo restituisce il nome normalizzato.

?? *getName*: è il metodo che estrae dalla tabella dei nomi il nome originale corrispondente al nome "normalizzato" passatogli come parametro. Esegue sostanzialmente l'operazione inversa di ciò che sortisce il metodo *setName* senza tuttavia modificare la tabella. Restituisce la stringa nome originale.

?? *translQuery*: è il metodo che si occupa di tradurre le query. Prevede, come unico parametro in ingresso, una stringa contenente la query proveniente dal Query Manager; restituisce, dopo aver effettuato un confronto dei termini che la costituiscono con i nomi in tabella, la stessa query espressa in termini non normalizzati.

?? *view*: stampa il contenuto della tabella dei nomi;

?? *clear*: cancella il contenuto della tabella dei nomi.

Classe TableDesc

PROPRIETA'

Ogni istanza della classe è rappresentativa di una tabella e contiene quattro variabili membro private:

?? *tableName*: contiene il nome della tabella;

?? *keyList*: contiene la descrizione della lista delle chiavi della tabella;

?? *fkeyList*: contiene la descrizione della lista delle foreign key della tabella;

?? *attrDef*: contiene la descrizione delle definizioni degli attributi della tabella.

METODI

?? *TableDesc*: è il metodo costruttore che ad ogni invocazione istanzia un oggetto TableDesc e gli assegna come nome il nome della tabella passatogli come parametro;

?? *addkeyList*: il metodo aggiunge la descrizione della lista delle chiavi della tabella passatagli come parametro;

?? *addfkeyList*: questo metodo aggiunge la descrizione della lista delle foreign key della tabella passatagli come parametro;

?? *addattrDef*: il metodo aggiunge la descrizione delle definizioni degli attributi della tabella passatagli come parametro;

?? *getName*: il metodo restituisce il nome della tabella;

?? *getkeyList*: il metodo restituisce la descrizione della lista delle chiavi primarie;

?? *getfkeyList*: il metodo restituisce la descrizione della lista delle foreign key;

?? *getattrDef*: il metodo restituisce la descrizione della definizione degli attributi;

Classe Utility

METODI

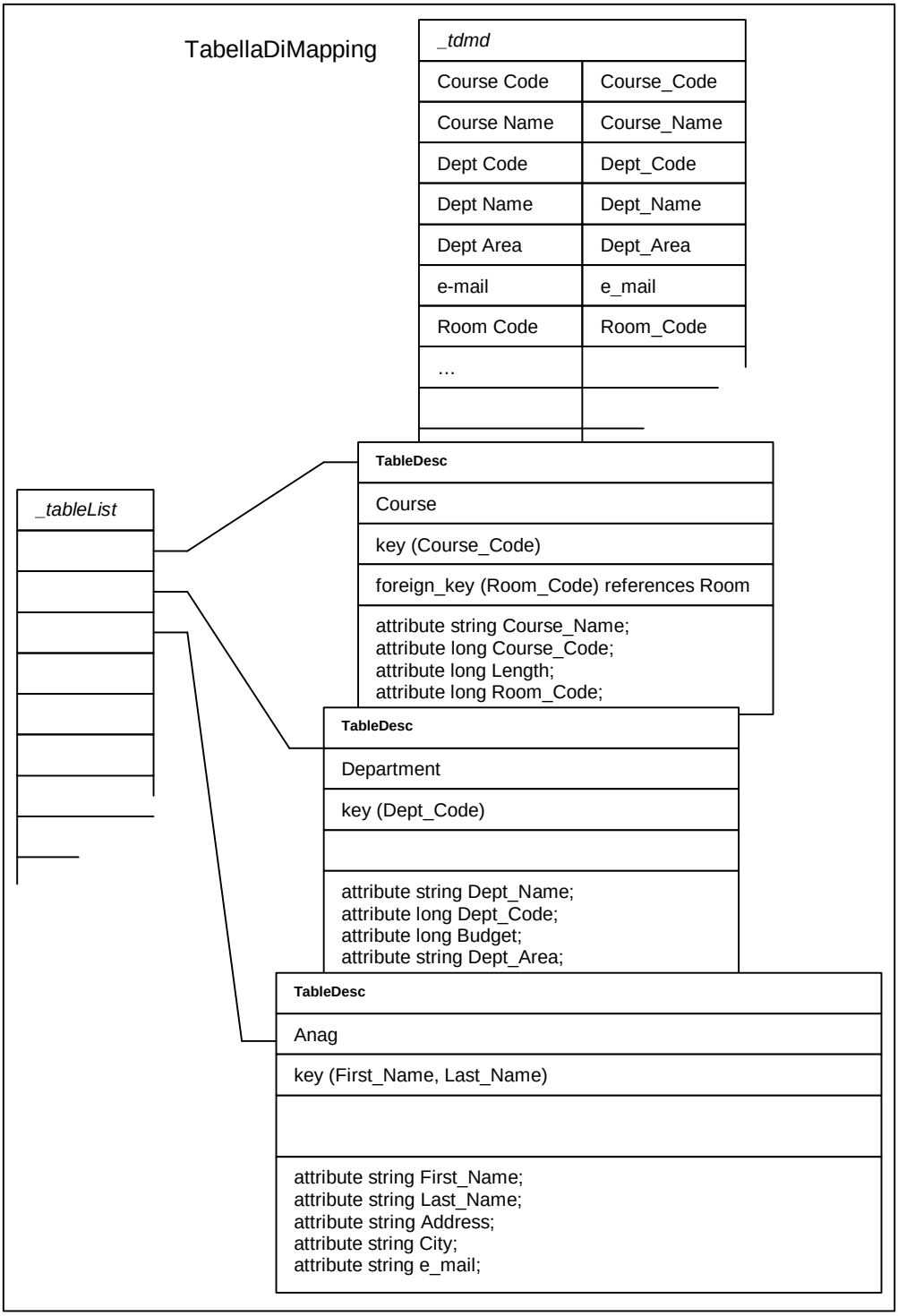
?? *getRelation*: è il metodo che effettua l'estrazione di una relazione dalla tabella di sistema di Access MSysRelationships; questa operazione è necessaria quando dalle tabelle del db non è possibile ricavare tutti i dati necessari a ricostruire le relazioni esistenti nel database. Riprenderemo la questione nella sezione 4.4.

?? *parseString*: è il metodo che effettua il parsing di una stringa; questo metodo traduce tutto ciò che non è carattere maiuscolo, minuscolo e/o numerico nella stringa passatagli come parametro d'ingresso in '_' (underscore). Restituisce il nome normalizzato sotto forma di stringa.

A titolo di esempio, vediamo come il wrapper popola queste strutture in funzione degli oggetti che individua durante l'analisi della sorgente dati University. In particolare osserviamo in Figura 4.3 come il vettore *_tableList* contenga una lista di istanze della classe *TableDesc*, ciascuna rappresentativa di una tabella del database. All'interno di ciascuna di queste troviamo la descrizione di tutti gli elementi caratteristici della tabella. In figura compare anche una porzione della tabella di mapping dei nomi TabellaDiMapping: all'interno abbiamo le coppie nome originale-nome tradotto.

4.3 Uso (wrapper server e client)

Il primo passo che occorre necessariamente compiere per poter effettuare qualunque operazione sul db d'esempio, o su qualunque altro db scelto come origine dati, è la registrazione dello stesso come origine dati nel sistema in cui è situato. Occorre cioè impostare le *Origini dati ODBC (32 bit)* nel *Pannello di controllo*. Richiamando la *Scheda DSN utente* è possibile aggiungere un'origini dati utente. Queste origini dati sono locali ad un computer e possono essere utilizzate soltanto dall'utente


TabellaDiMapping
_tdmd
Course Code
Course_Code
Course Name
Course_Name
Dept Code
Dept_Code
Dept Name
Dept_Name
Dept Area
Dept_Area
e-mail
e_mail
Room Code
Room_Code
…
_tableList
TableDesc
Course
key (Course_Code)
foreign_key (Room_Code) references Room
attribute string Course_Name;
attribute long Course_Code;
attribute long Length;
attribute long Room_Code;
TableDesc
Department
key (Dept_Code)
attribute string Dept_Name;
attribute long Dept_Code;
attribute long Budget;
attribute string Dept_Area;
TableDesc
Anag
key (First_Name, Last_Name)
attribute string First_Name;
attribute string Last_Name;
attribute string Address;
attribute string City;
attribute string e_mail;

corrente. Selezionando il pulsante *Aggiungi…* e successivamente *Microsoft Access Driver (*.mdb)* ci appare una finestra in cui ci viene richiesto il *Nome origine dati*, il nome mediante il quale il nostro db sarà visibile nel sistema (nel nostro esempio 'University') ed una breve *Descrizione* facoltativa. Selezioniamo infine University.mdb come origine dei dati (*Seleziona…* in *Database*) e siamo pronti ad utilizzare il wrapper.

L'utilizzo del wrapper si rivela piuttosto semplice. E' infatti sufficiente lanciare il wrapper da una finestra DOS mediante il comando

```
C:\java\wrapper_access>java Wrapper_Server wrapper.conf
```

dove `C:\java\wrapper_access` è la directory contenente la classe Wrapper_Server.class mentre `wrapper.conf` è il file di configurazione del wrapper. In questo file troviamo i parametri per la configurazione della connessione CORBA (orbServer, l'URL del server ORB e orbPort, la porta di comunicazione tra il wrapper ed il server ORB), il nome con cui il wrapper server si registra (wrapper_serverName=wrapperAccess) ed il nome della sorgente che il server gestisce (wrapper_dataSourceName=University).

A questo punto il wrapper è pronto a rispondere ad ogni richiesta che gli perviene. In particolare l'interfaccia MomisInterface prevede nome della sorgente, tipo, descrizione e possibilità di rivolgere query.

Per poter verificare le funzionalità del server è stato sviluppato un client il cui compito è quello di inviare al server richieste di servizi simulando così l'interazione del server con il mediatore. In una seconda finestra DOS è sufficiente mandare in esecuzione

```
C:\java\wrapper_access>java Wrapper_Client wrapperAccess
```

per avere il prompt del client. `wrapperAccess` rappresenta il nome con cui il Wrapper_Server si registra al mediatore: è comunque possibile modificare tale nome intervenendo sul file di configurazione del server `wrapper.conf`. A questo punto sono disponbili diversi comandi per l'interrogazione del server (h for help).

Richiedendo il nome, il tipo e la descrizione della nostra sorgente di esempio otterremo rispettivamente `University`, `JDBC` e la descrizione riportata in appendice D.

Sottolineiamo come sia possibile utilizzare il servered il client senza necessariamente connettersi ad una rete. Sono infatti sufficienti poche operazioni per poter verificare il server off-line.

Come primo passo lanciamo il gestore del servizio di risoluzione dei nomi da una prima finestra DOS:

```
C:\tnameserv
```

Configuriamo il server mediante i seguenti parametri in wrapper.conf:

```
orbServer=127.0.0.1
orbPort=900

wrapper_serverName=wrapperAccess
wrapper_dataSourceName=University
```

e attiviamo quindi il server ed il client rispettivamente in una seconda ed una terza finestra DOS come precedentemente spiegato.

4.4 Problemi risolti e problemi da risolvere

Un primo problema, peraltro minore, che ci si è trovati a dover risolvere è quello già accennato nel paragrafo 4.2: il parsing dei nomi. Access ammette gli 'spazi' nei nomi delle tabelle e/o delle relazioni, mentre in MOMIS si è deciso di normalizzare tutti i nomi, ossia di filtrare tutto ciò che non è alfanumerico. In particolare gli spazi ed altri caratteri di punteggiatura vengono sostituiti da '_' (underscore). Questo ha comportato la necessità di creare una classe dedicata alla mappatura dei nomi (**TabellaDiMapping**) e la creazione di metodi per la gestione della stessa (**parseString**).

Il problema principale però è rimasto il seguente: creato il nostro database di esempio, configurato ed avviato Wrapper_Server, verifichiamo, grazie a Wrapper_Client, che non vi sono problemi nell'ottenerne la descrizione.

Se ora creiamo un nuovo database a partire da uno più semplice ottenuto mediante lo strumento di autocomposizione di Access e ne chiediamo la descrizione, notiamo che non sono presenti le definizioni delle Foreign Key. Questo 'bug' è dovuto al fatto che dalle tabelle di proprietà dell'Amministratore del database non è possibile ottenere tali informazioni. Inoltre il driver ODBC non supporta le funzioni API di Java

che permettono di ottenere le stesse informazioni. La soluzione proposta, funzionante ma non eccessivamente elegante, consiste nel modificare le autorizzazioni di accesso ad una particolare tabella di sistema di Access (MSysRelationships) in cui sono contenute tutte le informazioni riguardanti le relazioni definite nel db.

Nel menu *Strumenti/Opzioni…* selezioniamo, nel pannello *Visualizzazione*, Mostra*/Oggetti di sistema*: noteremo che tra le tabelle sono apparse sei nuove tabelle di sistema (distinguibili dall'icona___) tra cui MSysRelationships.

Richiamando dal menu *Strumenti/Sicurezza/Autorizzazioni utenti e gruppi…* il pannello *Cambia proprietario* possiamo impostare come nuovo proprietario della tabella l'Amministratore invece di Engine, il modulo di gestione di database Microsoft Jet. Questo è il sistema di gestione di database che recupera e memorizza dati nei database degli utenti e di sistema. Il modulo di gestione di database Microsoft Jet può essere considerato come un componente di gestione di dati con cui vengono generati altri sistemi di accesso ai dati come Microsoft Access. Come ultimo passo diamo all'amministratore, nuovo proprietario della tabella, *l'Autorizzazione* alla *Lettura dati* nel pannello *Autorizzazioni.*

Questa procedura permette al wrapper di ricavare tutte le informazioni necessarie a ricostruire le relazioni ma si può definire poco elegante in quanto impone una serie di operazioni che sarebbe più logico demandare al software.

Una possibile soluzione toerica potrebbero essere le funzioni di gestione dei database contenute nella libreria di sistema Access *odbcjt32.dll*: Java permette infatti di chiamare funzioni contenute all'interno di librerie native grazie alla Java Native Interface (JNI). Questo permetterebbe di modificare gli attributi delle tabelle senza dover impegnare l'utente.

Conclusioni

Il sistema MOMIS intende fornire all'utente una vista globale di un insieme di sorgenti eterogenee senza che l'utente debba avere un'effettiva conoscenza delle stesse. MOMIS, ed in particolar modo il componente Mediatore, deve quindi essere in grado di acquisire la descrizione di ciascuna sorgente che intende integrare. Queste descrizioni vengono fornite al mediatore dai wrapper mediante un linguaggio di definizione comune: l'ODL_I^3.

Obiettivo di questa tesi è stato lo sviluppo del componente che realizza la traduzione di una sorgente dati di tipo Access in ODL_I^3, il wrapper Access/ ODL_I^3. Questo componente si inserisce quindi al livello wrapper, ossia dei "traduttori" che interfacciano il mediatore alle diverse sorgenti.

Il lavoro svolto si è articolato in diverse fasi prima di giungere al progetto vero e proprio. Un primo passo è stato quello di studiare il sistema MOMIS al fine di capirne il funzionamento ma soprattutto l'ambiente in cui il progetto si inserisce. Successivamente è stato studiato il linguaggio Java con cui si è realizzato il software, un linguaggio ad oggetti sviluppato da Sun Microsystems. A questo punto è stato possibile iniziare la fase di realizzazione del modulo wrapper che costituisce la parte principale del lavoro svolto.

Essendo Access un applicativo in continuo sviluppo, in futuro potrà rendersi necessario intervenire sul wrapper qualora le prossime versioni di Access non mantengano una compatibilità "all'indietro" a livello di organizzazione dei dati nelle tabelle di sistema. Rimane inoltre aperto il problema delle modifiche degli attributi delle tabelle eseguito da software, ossia della necessità dell'intervento dell'utente sul db sorgente.

Appendice A

Glossario I^3

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

?? Sezione 1: Architettura

?? Sezione 2: Servizi

?? Sezione 3: Risorse

?? Sezione 4: Ontologie

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

A.1 Architettura

?? Architettura = insieme di componenti.

?? architettura di riferimento = linea guida ed insieme di regole da seguire per l'architettura.

?? componente = uno dei blocchi sui quali si basa un'applicazione o una configurazione. Incorpora strumenti e conoscenza specifica del dominio.

?? applicazione = configurazione persistente o transitoria dei componenti, rivolta a risolvere un problema del cliente, e che può coprire diversi domini.

?? configurazione = istanza particolare di un'architettura per un'applicazione o un cliente.

?? collante (glue) = software o regole che servono per collegare i componenti o per interoperare attraverso i domini.

?? strato = grossolana categorizzazione dei componenti e degli strumenti in una configurazione. L'architettura I3 distingue tre strati, ognuno dei quali fornisce una diversa categoria di servizi:

?? Servizi di Coordinamento = coprono le fasi di scoperta delle risorse, distribuzione delle risorse, invocazione, scheduling…

?? Servizi di Mediazione = coprono la fase di query processing e di trattamento dei risultati, nonchè il filtraggio dei dati, la generazione di nuove informazioni, etc.

?? Servizi di Wrapping = servono per l'utilizzo dei wrappers e degli altri strumenti simili utilizzati per adattarsi a standards di accesso ai dati e alle convenzioni adoperate per la mediazione e per il coordinamento.

?? agente = strumento che realizza un servizio, sia per il suo proprietario, sia per un cliente del suo proprietario.

?? facilitatore = componente che fornisce i servizi di coordinamento, come pure l'instradamento delle interrogazioni del cliente.

?? mediatore = componente che fornisce i servizi di mediazione e che provvede a dare valore aggiunto alle informazioni che sono trasmesse al cliente in risposta ad una interrogazione.

?? cliente (customer) = proprietario dell'applicazione che gestisce le interrogazioni o utente finale, che usufruisce dei servizi.

?? risorsa = base di dati accessibile, server ad oggetti, base di conoscenze…

?? contenuto = risultato informativo ricavato da una sorgente.

?? servizio = funzione fornita da uno strumento in un componente e diretta ad un cliente, direttamente od indirettamente.

?? strumento (tool) = programma software che realizza un servizio, tipicamente indipendentemente dal dominio.

?? wrapper = strumento utilizzato per accedere alle risorse conosciute, e per tradurre i suoi oggetti.

?? regole limitative (constraint rules) = definizione di regole per l'assegnamento di componenti o di protocolli a determinati strati.

?? interoperare = combinare sorgenti e domini multipli.

?? informazione = dato utile ad un cliente.

?? informazione azionabile = informazione che forza il cliente ad iniziare un evento.

?? dato = registrazione di un fatto.

?? testo = dato, informazione o conoscenza in un formato relativamente non strutturato, basato sui caratteri.

?? conoscenza = metadata, relazione tra termini, paradigmi…, utili per trasformare i dati in informazioni.

?? dominio = area, argomento, caratterizzato da una semantica interna, per esempio la finanza, o i componenti elettronici…

?? metadata = informazione descrittiva relativa ai dati di una risorsa, compresi il dominio, proprietà, le restrizioni, il modello di dati, …

?? metaconoscenza = informazione descrittiva relativa alla conoscenza in una risorsa, includendo l'ontologia, la rappresentazione…

?? metainformazioni = informazione descrittiva sui servizi, sulle capacità, sui costi.

A.2 Servizi

?? Servizio = funzionalità fornita da uno o più componenti, diretta ad un cliente.

?? instradamento (routing) = servizio di coordinamento per localizzare ed invocare una risorsa o un servizio di mediazione, o per creare una configurazione. Fa uso di un direttorio.

?? scheduling = servizio di coordinamento per determinare l'ordine di invocazione degli accessi e di altri servizi; fa spesso uso dei costi stimati.

?? accoppiamento (matchmaking) = servizio che accoppia i sottoscrittori di un servizio ai fornitori.

?? intermediazione (brokering) = servizio di coordinamento per localizzare le risorse migliori.

?? strumento di configurazione = programma usato nel coordinamento per aiutare a selezionare ed organizzare i componenti in una istanza particolare di una configurazione architetturale.

?? servizi di descrizione = metaservizi che informano i clienti sui servizi, risorse…

?? direttorio = servizio per localizzare e contattare le risorse disponibili, come le pagine gialle, pagine bianche…

?? decomposizione dell'interrogazione (query decomposition) = determina le interrogazioni da spedire alle risorse o ai servizi disponibili.

?? riformulazione dell'interrogazione (query reformulation) = programma per ottimizzare o rilassare le interrogazioni, tipicamente fa uso dello scheduling.

?? contenuto = risultato prodotto da una risorsa in risposta ad interrogazioni.

?? trattamento del contenuto (content processing) = servizio di mediazione che manipola i risultati ottenuti, tipicamente per incrementare il valore delle informazioni.

?? trattamento del testo = servizio di mediazione che opera sul testo per ricerca, correzione…

?? filtraggio = servizio di mediazione per aumentare la pertinenza delle informazioni ricevute in risposta ad interrogazioni.

?? classificazione (ranking) = servizio di mediazione per assegnare dei valori agli oggetti ritrovati.

?? spiegazione = servizio di mediazione per presentare i modelli ai clienti.

?? amministrazione del modello = servizio di mediazione per permettere al cliente ed al proprietario del mediatore di aggiornare il modello.

?? integrazione = servizio di mediazione che combina i contenuti ricevuti da una molteplicità di risorse, spesso eteorgenee.

?? accoppiamento temporale = servizio di mediazione per riconoscere e risolvere differenze nelle unità di misura temporali utilizzate dalle risorse.

?? accoppiamento spaziale = servizio di mediazione per riconoscere e risolvere differenze nelle unità di misura spaziali utilizzate dalle risorse.

?? ragionamento (reasoning) = metodologia usata da alcuni componenti o servizi per realizzare inferenze logiche.

?? browsing = servizio per permettere al cliente di spostarsi attraverso le risorse.

?? scoperta delle risorse = servizio che ricerca le risorse.

?? indicizzazione = creazione di una lista di oggetti (indice) per aumentare la velocità dei servizi di accesso.

?? analisi del contenuto = trattamento degli oggetti testuali per creare informazioni.

?? accesso = collegamento agli oggetti nelle risorse per realizzare interrogazioni, analisi o aggiornamenti.

?? ottimizzazione = processo di manipolazione o di riorganizzazione delle interrogazioni per ridurne il costo o il tempo di risposta.

?? rilassamento = servizio che fornisce un insieme di risposta maggiore rispetto a quello che l'interrogazione voleva selezionare.

?? astrazione = servizio per ridurre le dimensioni del contenuto portandolo ad un livello superiore.

?? pubblicità (advertising) = presentazione del modello di una risorsa o del mediatore ad un componente o ad un cliente.

?? sottoscrizione = richiesta di un componente o di un cliente di essere informato su un evento.

?? controllo (monitoring) = osservazione delle risorse o dei dati virtuali e creazione di impulsi da azionare ogniqualvolta avvenga un cambiamento di stato.

?? aggiornamento = trasmissione dei cambiamenti dei dati alle risorse.

?? instanziazione del mediatore = popolamento di uno strumento indipendente dal dominio con conoscenze dipendenti da un dominio.

?? attivo (activeness) = abilità di un impulso di reagire ad un evento.

?? servizio di transazione = servizio che assicura la consistenza temporale dei contenuti, realizzato attraverso l'amministrazione delle transzioni.

?? accertamento dell'impatto = servizio che riporta quali risorse saranno interessate dalle interrogazioni o dagli aggiornamenti.

?? stimatore = servizo di basso livello che stima i costi previsti e le prestazioni basandosi su un modello, o su statistiche.

?? caching = mantenere le informazioni memorizzate in un livello intermedio per migliorare le prestazioni.

?? traduzione = trasformazione dei dati nella forma e nella sintassi richiesta dal ricevente.

?? controllo della concorrenza = assicurazione del sincronismo degli aggiornamenti delle risorse, tipicamente assegnato al sistema che amministra le transazioni.

A.3 Risorse

?? Risorsa = base di dati accessibile, simulazione, base di conoscenza, … comprese le risorse "legacy".

?? risorse “legacy” = risorse preesistenti o autonome, non disegnate per interoperare con una architettura generale e flessibile.

?? evento = ragione per il cambiamento di stato all’interno di un componente o di una risorsa.

?? oggetto = istanza particolare appartenente ad una risorsa, al modello del cliente, o ad un certo strumento.

?? valore = contenuto metrico presente nel modello del cliente, come qualità, rilevanza, costo.

?? proprietario = individuo o organizzazione che ha creato, o ha i diritti di un oggetto, e lo può sfruttare.

?? proprietario di un servizio = individuo o organizzazione responsabile di un servizio.

?? database = risorsa che comprende un insieme di dati con uno schema descrittivo.

?? warehouse = database che contiene o dà accesso a dati selezionati, astratti e integrati da una molteplicità di sorgenti. Tipicamente ridondante rispetto alle sorgenti di dati.

?? base di conoscenza = risorsa comprendente un insieme di conoscenze trattabili in modo automatico, spesso nella forma di regole e di metadata; permettono l’accesso alle risorse.

?? simulazione = risorsa in grado di fare proiezioni future sui dati e generare nuove informazioni, basata su un modello.

?? amministrazione della transazione = assicurare che la consistenza temporale del database non sia compromessa dagli aggiornamenti

?? impatto della transazione = riporta le risorse che sono state coinvolte in un aggiornamento.

?? schema = lista delle relazioni, degli attributi e, quando possibile, degli oggetti, delle regole, e dei metadata di un database. Costituisce la base dell’ontologia della risorsa.

?? dizionario = lista dei termini, fa parte dell’ontologia.

?? modello del database = descrizione formalizzata della risorsa database, che include lo schema.

?? interoperabilità = capacità di interoperare.

?? eterogeneità = incompatibilità trovate tra risorse e servizi sviluppati autonamamente, che vanno dalla piattaforma utilizzata, sistema operativo, modello dei dati, alla semantica, ontologia, …

?? costo = prezzo per fornire un servizio o un accesso ad un oggetto.

?? database deduttivo = database in grado di utilizzare regole logiche per trattare i dati.

?? regola = affermazione logica, unità della conoscenza trattabile in modo automatico.

?? sistema di amministrazione delle regole = software indipendente dal dominio che raccoglie, seleziona ed agisce sulle regole.

?? database attivo = database in grado di reagire a determinati eventi.

?? dato virtuale = dato rappresentato attraverso referenze e procedure.

?? stato = istanza o versione di una base di dati o informazioni.

?? cambiamento di stato = stato successivo ad una azione di aggiornamento, inserimento o cancellazione.

?? vista = sottoinsieme di un database, sottoposto a limiti, e ristrutturato.

?? server di oggetti = fornisce dati oggetto.

?? gerarchia = struttura di un modello che assegna ogni oggetto ad un livello, e definisceper ogni oggetto l'oggetto da cui deriva.

?? network = struttura di un modello che fa uso di relazioni relativamente libere tra oggetti.

?? ristrutturare = dare una struttura diversa ai dati seguendo un modello differente dall'originale.

?? livello = categorizzazione concettuale , dove gli oggetti di un livello inferiore dipendono da un antenato di livello superiore.

?? antenato (ancestor) = oggetto di livello superiore, dal quale derivano attributi ereditabili.

?? oggetto root = oggetto da cui tutti gli altri derivano, all'interno di una gerarchia.

?? datawarehouse = deposito di dati integrati provenienti da una molteplicità di risorse.

?? deposito di metadata = database che contiene metadata o metainformazioni.

A.4 Ontologia

?? Ontologia = descrizione particolareggiata di una concettualizzazione, p.e. l'insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti, spesso ambigui tra domini diversi.

?? concetto = definisce una astrazione o una aggregazione di oggetti per il cliente.

?? semantico = che si riferisce al significato di un termine, espresso come un insieme di relazioni.

?? sintattico = che si riferisce al formato di un termine, espresso come un insieme di limitazioni.

?? classe = definisce metaconoscenze come metodi, attributi, ereditarietà, per gli oggetti in essa istanziati.

?? relazione = collegamento tra termini, come is-a, part-of, …

?? ontologia unita (merged) = ontologia creata combinando diverse ontologie, ottenuta mettendole in relazione tra loro (mapping).

?? ontologia condivisa = sottoinsieme di diverse ontologie condiviso da una molteplicità di utenti.

?? comparatore di ontologie = strumento per determinare relazioni tra ontologie, utilizzato per determinare le regole necessarie per la loro integrazione.

?? mapping tra ontologie = trasformazione dei termini tra le ontologie, attraverso regole di accoppiamento, utilizzato per collegare utenti e risorse.

?? regole di accoppiamento (matching rules) = dichiarazioni per definire l'equivalenza tra termini di domini diversi.

?? trasformazione dello schema = adattamento dello schema ad un'altra ontologia.

?? editing = trattamento di un testo per assicurarne la conformità ad una ontologia.

?? algebra dell'ontologia = insieme delle operazioni per definire relazioni tra ontologie.

?? consistenza temporale = è raggiunta se tutti i dati si riferiscono alla stessa istanza temporale ed utilizzano la stessa granularità temporale.

?? specifico ad un dominio = relativo ad un singolo dominio, presuppone l'assenza di incompatibilità semantiche.

?? indipendente dal dominio = software, strumento o conoscenza globale applicabile ad una molteplicità di domini.

Appendice B

Il linguaggio ODL dello standard ODMG-93

L'appendice riporta la descrizione in BNF del linguaggio ODL dello standard ODMG-93 descritta in [28].

<specification> :: = <definition>
| <definition> <specification>
<definition> :: = <type_dcl> ;
| <const_dcl> ;
| <except_dcl> ;
| <interface_dcl> ;
| <module> ;
<module> :: = **module** <identifier> { <specification> }
<interface> :: = <interface_dcl>
| <forward_dcl>
<interface dcl> :: = <interface_header>
[:<persistence_dcl>] {[<interface body>]}
<persistence_dcl> :: = **persistent** | **transient**
<forward_dcl> :: = **interface** <identifier>
<interface header> :: = **interface** <identifier>
[<inheritance_spec>]
[<type_property_list>]
<type_property_list> :: = ([<extent_spec>] [<key_spec>])
<extent_spec> :: = **extent** <string>
<key_spec> :: = **key**[**s**] <key_list>
<key_list> :: = <key> | <key> , <key_list>
<key> :: = <property_name> | (<property_list>)
<property_list> :: = <property_name>
| <property_name> , <property_list>

```
<property_name>        :: =  <identifier>
<interface_body>       :: =  <export> | <export> <interface_body>
<export>               :: =  <type_dcl> ;
                             | <const_dcl> ;
                             | <except_dcl> ;
                             | <attr_dcl> ;
                             | <rel_dcl> ;
                             | <op_dcl> ;
<inheritance_spec>     :: =  :<scoped_name> [,<inheritance_spec>]
<scoped_name>          :: =  <identifier>
                             | ::<identifier>
                             | <scoped_name>::<identifier>
<const_dcl>            :: =  const <const_type> <identifier> = <const_exp>
<const_type>           :: =  <integer_type>
                             | <char_type>
                             | <boolean_type>
                             | <floating_pt_type>
                             | <string_type>
                             | <scoped_name>
<const_exp>            :: =  <or_expr>
<or_expr>              :: =  <xor_expr>
                             | <or_expr> | <xor_expr>
<xor_expr>             :: =  <and_expr>
                             | <xor_expr> ^ <and_expr>
<and_expr>             :: =  <shift_expr>
                             | <and_expr> & <shift_expr>
<shift_expr>           :: =  <add_expr>
                             | <shift_expr> >> <add_expr>
                             | <shift_expr> << <add_expr>
<add_expr>             :: =  <mult_expr>
                             | <add_expr> + <mult_expr>
                             | <add_expr> - <mult_expr>
<mult_expr>            :: =  <unary_expr>
                             | <mult_expr> * <unary_expr>
                             | <mult_expr> / <unary_expr>
                             | <mult_expr> % <unary_expr>
<unary_expr>           :: =  <primary_expr>
                             | <unary_operator> | <primary_expr>
<unary_operator>       :: =  - | + | ?
```

<primary_expr> :: = <scoped_name>
| <literal>
| **(** | <const_exp>**)**
<literal> :: = <integer_literal>
| <string_literal>
| <character_literal>
| <floating_pt_literal>
| <boolean_literal>
<boolean_literal> :: = **TRUE** | **FALSE**
<positive_int_const> :: = <const_exp>
<type_dcl> :: = **typedef** <type_declarator>
| <struct_type>
| <union_type>
| <enum_type>
<type_declarator> :: = <type_spec> <declarators>
<type_spec> :: = <simple_type_spec>
| <constr_type_spec>
<simple_type_spec> :: = <base_type_spec>
| <template_type_spec>
| <scoped_name>
<base_type_spec> :: = <floating_pt_type>
| <integer_type>
| <char_type>
| <boolean_type>
| <octet_type>
| <any_type>
<template_type_spec> :: = <array_type>
| <string_type>
| <coll_type>
<coll_type> :: = <coll_spec> <simple_type_spec>
<coll_spec> :: = **set** | **list** | **bag**
<constr_type_spec> :: = <struct_type>
| <union_type>
| <enum_type>

<declarators> :: = <declarator>
| <declarator> , <declarators>
<declarator> :: = <simple_declarator>
| <complex_declarator>

```
<simple_declarator>      :: = <identifier>
<complex_declarator>     :: = <array_declarator>
<floating_pt_type>       :: = float | double
<integer_type>           :: = <signed_int>
                              <unsigned_int>
<signed_int>             :: = <signed_long_int>
                              <signed_short_int>
<signed_long_int>        :: = long
<signed_short_int>       :: = short
<unsigned_int>           :: = <unsigned_long_int>
                              <unsigned_short_int>
<unsigned_long_int>      :: = unsigned long
<unsigned_short_int>     :: = unsigned short
<char_type>              :: = char
<boolean_type>           :: = boolean
<octet_type>             :: = octet
<any_type>               :: = any
<struct_type>            :: = struct <identifier> {<member_list>}
<member_list>            :: = <member>
                              | <member> <member_list>
<member>                 :: = <type_spec> <declarators> ;
<union_type>             :: = union <identifier> switch
                              (<switch_type_spec>) {<switch_body>}
<switch_type_spec>       :: = <integer_type>
                              | <char type>
                              | <boolean type>
                              | <enum type>
                              | <scoped name>
<switch_body>            :: = <case> | <case> <switch_body>
<case>                   :: = <case_label_list> <element_spec> ;
<case_label list>        :: = <case_label>
                              | <case_label> <case_label_list>
<case_label>             :: = case <const_exp>:
                              | default:
<element_spec>           :: = <type_spec> <declarator>
<enum_type>              :: = enum <identifier> {<enumerator_list>}
<enumerator_list>        :: = <enumerator>
                              | <enumerator> , <enumerator_list>
<enumerator>             :: = <identifier>
```

<array_type> :: = <array_spec>
?<simple_type_spec> , <positive_int_const>?
| <array_spec> ?<simple type spec>?
<array_spec> :: = **array** | **sequence**
<string_type> :: = **string** | **string** <positive_int_const>
<array_declarator> :: = <identifier> <array_size_list>
<array_size_list> :: = <fixed_array_size>
| <fixed_array_size> <array_size_list>
<fixed_array_size> :: = [<positive_int_const>]
<attr_dcl> :: = [**readonly**] **attribute** <domain_type>
<attribute_name> [<fixed_array_size>]
<domain_type> :: = <simple_type_spec>
| <struct_type>
| <enum_type>
<rel_dcl> :: = **relationship** <target_of_path> <identifier>
inverse <inverse_trasversal_path>
<target_of_path> :: = <identifier>
| <rel_collection_type> ?<identifier>?
<inverse_trasversal_path> :: = <identifier>**::** <identifier>
<attribute_list> :: = <scoped_name>
| <scoped_name> , <attribute_list>
<rel_collection_type> :: = **set** | **list** | **bag** | **array**
<except_dcl> :: = **exception** <identifier> {[<member_list>]}
<op_dcl> :: = [<op_attribute>] <op_type_spec> <identifier>
<parameter_dcls> [<raises_expr>]
[<context_expr>]
<op_attribute> :: = **oneway**
<op_type_spec> :: = <simple_type_spec> |**void**
<parameter_dcls> :: = ([<param_dcl_list>])
<param_dcl_list> :: = <param_dcl>
| <param_dcl> , <param_dcl_list>
<param_dcl> :: = <param_attribute> <simple_type_spec>
<declarator>
<param_attribute> :: = **in** | **out** | **inout**
<raises_expr> :: = **raises** (<scoped_name_list>)
<scoped_name_list> :: = <scoped_name>
| <scoped_name> , <scoped_name_list>
<context_expr> :: = **context** (<string_literal_list>)

<string_literal_list> :: = <string_literal>
| <string_literal> , <string_literal_list>

Appendice C

Il linguaggio ODL_I^3

L'appendice riporta la descrizione in BNF del linguaggio ODL_I^3. Viene descritta solo la parte di sintassi relativa alle estensioni compiute rispetto alla grammatica ODL dello standard ODMG-93, già presentata in appendice B.

```
<specification>          :: =  <definition>
                               | <definition> <specification>
<definition>                   <type_dcl> ;
                               | <const_dcl> ;
                               | <except_dcl> ;
                               | <interface_dcl> ;
                               | <relationships_dcl> ;
                               | <rule_dcl> ;
                               | <module> ;
```

Parte relativa alla definizione del tipo-valore range.

```
<base_type_spec>               <floating_pt_type>
                               | <integer_type>
                               | <char_type>
                               | <boolean_type>
                               | <octet_type>
                               | <range_type>
                               | <any_type>
<range_type>                   range [ <range_specifier> ]
<range_specifier>              <const_exp> , <const_exp>
                               | <const_exp> , +inf
                               | -inf , <const_exp>
```

Parte relativa alla specificazione delle classi, in cui è possibile definire più di un interface_body.

<interface_dcl>	<interface_header> {[<interface body>] [**union** <interface_body>]}
<interface_header>	**interface** <identifier> [<inheritance_spec>] [<type_property_list>]
<inheritance_spec>	**:** <scoped_name> [**,** <inheritance_spec>]

Parte relativa alla specificazione delle classi, in cui occorre specificare il tipo e il nome della sorgente informativa.

<type_property_list>	([<source_spec>] [<extent_spec>] [<key_spec>] [<f_key_list>])
<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_list>	<f_key_spec> | <f_key_spec> , <f_key_list>
<f_key_spec>	**foreign key** (<key_list>) **references** <identifier> [**,**(<key_list>)]

…

Regole di definizione del mapping tra attributi della classe globale dello schema del mediatore ed 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>

…

Terminological relationships usate per la definizione del Common Thesaurus.

<relationships_dcl>	<local_entity> <relationship_type> <local_entity>
<local_entity>	<local_class_name> | <local_attr_name>
<local_class_name>	<source_name>**.**<class_name>
<relationship_type>	**SYN | BT** | **NT** | **RT**

…

Definizione degli **OLCD** integrity constraint: le regole sono definite tramite la logica di *if then* e sono valide per ogni istanza del database. Definizione delle regole di *or* e *union.*

<rule_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> [<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 D

Esempio di riferimento in ODL_I^3

Di seguito è riportata la descrizione, mediante il linguaggio ODL_I^3, dell'esempio di riferimento presentato nella Sezione 4.1

```
Source name:  University

interface Anag (
  source relational University
  key (First_Name, Last_Name)
  )
  {
    attribute string First_Name;
    attribute string Last_Name;
    attribute string Address;
    attribute string City;
    attribute string e_mail;
  };

interface Course (
  source relational University
  key (Course_Code)
  foreign_key (Room_Code) references Room
  )
  {
    attribute string Course_Name;
    attribute long Course_Code;
    attribute long Length;
    attribute long Room_Code;
  };

interface Department (
  source relational University
  key (Dept_Code)
  )
```

```
  {
    attribute string Dept_Name;
    attribute long Dept_Code;
    attribute long Budget;
    attribute string Dept_Area;
  };

interface Research_Staff (
  source relational University
  key (First_Name, Last_Name)
  foreign_key (First_Name, Last_Name) refer ences Anag
  foreign_key (Course_Code) references Course
  foreign_key (Dept_Code) references Department
  )
  {
    attribute string First_Name;
    attribute string Last_Name;
    attribute string Relation;
    attribute long Dept_Code;
    attribute long Course_Code;
  };

interface Room (
  source relational University
  key (Room_Code)
  )
  {
    attribute long Room_Code;
    attribute long Seats_Number;
    attribute string Notes;
  };

interface School_Member (
  source relational University
  key (First_Name, Last_Name)
  foreign_key (First_Name, Last_Name) references Anag
  )
  {
    attribute string First_Name;
    attribute string Last_Name;
    attribute string Faculty;
    attribute long Year;
  };
```

Appendice E

Restrizione del OQL per le BasicQuery

Le “base extension” costituiscono di fatto una restrizione del linguaggio OQL. Di seguito viene riportata la descrizione in forma BNF di tale restrizione.

<Query> :: = (<Query>)
| <SelectExpr>

<SelectPreamble> :: = **select distinct**
| **select**

<SelectExpr> :: = <SelectPreamble> <ProjectionList>
<FromClause> <WhereClause>

<ProjectionList> :: = <ProjectionAttributes>
| ***

<ProjectionAttributes> :: = <Attribute>
| <ProjectionAttributes> **,** <Attribute>

<Attribute> :: = <Projection>
| <Property>

<Projection> :: = (<Projection>)
| <Identifier> **,** <Property>
| <Property> **as** <Identifier>

<Property> :: = (<Property>)
| <Basic>
| <Identifier>
| <Accesor>

<Basic> :: = **nil**
| **true**
| **false**
| <FloatLiteral>
| <IntegerLiteral>
| <StringLiteral>

```
<StringLiteral>          :: = “<String>”
<IntegerLiteral>         :: = <UnsignedLong>
                              | <Sign> <UnsignedLong>
<FloatLiteral>           :: = . <UnsignedLong>
                              | <Sign> . <UnsignedLong>
                              | <IntegerLiteral> . <UnsignedLong>
<Sign>                   :: = + | -
<Digit>                  :: = 0 | 1 | … | 9
<Char>                   :: = a | b | … | z | A | B | … | Z
<UnsignedLong>           :: = <Digit>
                              | <Digit> <UnsignedLong>
<String>                 :: = <Char> | <Digit> | _
                              | <Char> <String>
                              | <Digit> <String>
                              | _<String>
<Identifier>             :: = <Char>
                              | <Char> <String>
<Accessor>               :: = <Identifier> <Path> <Accessor>
                              | <Identifier> <Path> <Identifier>
<Path>                   :: = . | ?
<FromClause>             :: = from <VariableDeclaration>
<VariableDeclaration>    :: = <Identifier>
                              | <Identifier> as <Identifier>
                              | <Identifier> <Identifier>
<WhereClause>            :: = ?
                              | where <Predicates>
<Predicates>             :: = ( <Predicates> )
                              | <Comparison>
                              | <BooleanExpr>
                              | <CollectionExpr>
<Comparison>             :: = <Operand> <ComparisonOperator>
                               <Quantifier> <Operand>
                              | <Property> like <StringLiteral>
<ComparisonOperator>     :: = > | >= | < | <= | = | ! =
<Quantifier>             :: = ?
                              | some
                              | any
                              | all
```

<Operand> :: = <Basic>
| <SimpleExpr>
| <Property>

<SimpleExpr> :: = <Property> + <Property>
| <Property> - <Property>
| <Property> / <Property>
| <Property>* <Property>
| - <Property>
| + <Property>
| <Property> **mod** <Property>
| **abs(** <Property> **)**
| <Property> || <Property>

<BooleanExpr> :: = **not** <Predicates>
| <Predicates> **and** <Predicates>
| <Predicates> **or** <Predicates>

<CollectionExpr> :: = **for all** <Identifier> **in** <Property>
: <Condition>
| **exists** <Identifier> **in** <Property>
: <Condition>
| **exists(** <Accessor> **)**
| **unique(** <Accessor> **)**
| <Property> **in** <Condition>
| **count(** <Property> **)**
| **sum(** <Property> **)**
| **min(** <Property> **)**
| **max(** <Property> **)**
| **avg(** <Property> **)**

Bibliografia

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

[2] Arpa reference architecture. Available at http://dc.isx.com/I3/.

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

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

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

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

[7] M.T. Roth and P. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In Proc. of the 23rd Int. Conf. on Very Large Databases, Athens, Greece, March 1995.

[8] Daniel P.Miranker and Vasilis Samoladas. Alamo: An Architecture for Integrating Heterogenous Data Sources. In Proceedings of the 4th KRDB Workshop, Athens, Greece, August 1997.

[9] 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, 1996-1997.

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

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

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

[13] G.P. Grifa. Analisi di affinità strutturali fra classi ODLI3 nel sistema MOMIS. Tesi di laurea, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di Laurea in Ingegneria Informatica, 1997-1998.

[14] S. Castano and V. De Antonellis. Semantic dictionary design for database interoperability. In Proc. of Int. Conf. on Data Engineering, ICDE'97, Birmingham, UK, April 1997.

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

[16] S. Castano and V. De Antonellis. Semantic dictionary design for database interoperability. In Proc. of Int. Conf. on Data Engineering, ICDE'97, Birmingham, UK, 1997.

[17] 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.

[18] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. IEEE Transactions on Knowledge and Data Engineering, 10:576-598, July/August 1998.

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

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

[21] D. Beneventano, 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.

[22] Odb-tools Project. Available at http://sparc20.dsi.unimo.it/index.html.

[23] R.G.G. Cattell, editor. The Object Database Standard: ODMG-93. Morgan Kaufmann Publishers, San Mateo, CA, 1994. Available at http://www.odmg.org/.

[24] R.G.G. Cattell and others., editors. The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publishers, San Francisco, CA, 2000. Available at http://www.odmg.org/.

[25] Available at http://www.omg.org/

[26] P. Buneman, L. Raschid, and J. Ullman. Mediator languages - a proposal for a standard, April 1996. Available at ftp://ftp.umiacs.umd.edu/pub/ONRrept/medmodel96.ps.