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

SIWeb:
una interfaccia internet per il sistema Momis

Relatore

Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di

Daniele Bianco

Correlatore

Dott. Ing. Maurizio Vincini

Anno Accademico 2000 - 2001

Parole chiave:
Applicazione Web
Integrazione delle Informazioni
Interoperabilità Corba
Java Servlet
Java Server Pages (JSP)

ad Eleonora

RINGRAZIAMENTI

Ringrazio tutti coloro che mi hanno fornito un aiuto alla realizzazione della presente tesi.

Un ringraziamento particolare va alla mia fidanzata Eleonora per la sua gioia di vivere che ha saputo trasmettermi. Senza di lei non sarei mai riuscito a raggiungere i risultati ottenuti.

Indice

Elenco delle figure

Introduzione

Negli ultimi anni internet ha avuto una crescita esponenziale, soprattutto dovuta al mondo imprenditoriale che ha iniziato ad investire in questo settore, sfruttando la rete come veicolo commerciale e pubblicitario.

Questa sovrabbondanza di informazioni disponibili online genera un problema relativo all'estrazione e all'interpretazione dei dati realmente significativi per l'utente. Il problema principale è dato dalla eterogeneità delle sorgenti di dati, che possono essere di svariato tipo: DBMS relazionali, ad oggetti, pagine HTML o dinamiche, documenti di Word Processor, fogli dati di Spreadsheet, documenti XML ecc ...

Sorge l'esigenza di integrare queste sorgenti, possibilmente in maniera automatica, e fornire all'utente una visione globale delle informazioni, in maniera semplice, senza che egli debba essere necessariamente a conoscenza delle tecnologie relative alle singole sorgenti di dati.

Nasce con questo scopo **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources) che permette l'accesso sia a sorgenti di dati eterogenee memorizzate in database di tipo tradizionale (relazionali o ad oggetti) o in file system, che a sorgenti di tipo semistrutturato.

L'obiettivo di questa tesi è quello di progettare una interfaccia utente per MOMIS che sia utilizzabile, via internet, attraverso un qualsiasi browser web. Attualmente esiste una applicazione Java chiamata SI-Designer che, per per la modalità con cui è stata progettata, non è facilmente convertibile in una applicazione web.

Nel corso della presente tesi si sono studiate essenzialmente due alternative: la conversione della preesistente applicazione in una applet, e la completa reingegnerizzazione con una tecnologia server-side, utilizzando Servlet, JavaBeans, JSP e CORBA.

Dopo una descrizione sullo stato dell'arte delle tecnologie a tre livelli, viene descritto nel dettaglio lo sviluppo dell'applicazione SIWeb, motivando la necessità di riprogettare l'interfaccia utilizzando la seconda delle alternative, ovvero l'approccio lato server.

La tesi si conclude con una descrizione dei miglioramenti apportabili al progetto, oltre agli sviluppi possibili con l'implementazione di nuove funzionalità.

Segue una breve descrizione del contenuto dei capitoli:

- **Capitolo 1: L'integrazione intelligente delle informazioni**
  In questo capitolo si descrive la tecnologia su cui si basa MOMIS. Si descrive la tecnologia I^3 e il trattamento dei metadati e delle ontologie.

- **Capitolo 2: L'architettura di MOMIS**
  In questo capitolo viene descritto il funzionamento di MOMIS, descrivendo gli ODB-Tools e le fasi del processo di integrazione delle sorgenti.

- **Capitolo 3: Tecnologie per il Word Wide Web**
  Vengono descritti i principali strumenti per lo sviluppo di sistemi a tre livelli. Particolare enfasi viene data all'ambiente Java ed alle tecnologie correlate. Tra i vari argomenti trattati vengono approfonditi particolarmente i seguenti:

  – Java Servlet
  – Java Server Pages
  – CORBA

- **Capitolo 4: SIWeb: L'interfaccia internet per MOMIS**
  In questo capitolo si affrontano i vari approcci studiati per interfacciare MOMIS con il Wold Wide Web. In particolare si confrontano due soluzioni:

  – Convertire la preesistente applicazione Java SI-Designer in una Applet che possa essere eseguita da un browser web;
  – Reingegnerizzare l'applicazione, sviluppandola direttamente server-side con l'utilizzo di servlet, javabeans e JSP.

  Si giustificherà la scelta dell'implementazione server-side, descrivendo nel dettaglio il funzionamento delle classi svliluppate e la loro interazione, tramite CORBA, con il sistema MOMIS.

- **Capitolo 5: Analisi dei risultati ottenuti**
  Allo scopo di dimostrare la totale compatibilità di SIWeb con SIDesigner, le due applicazioni vengono messe a confronto in una vera sessione di lavoro, mostrando come i due sistemi ottengano esattamente gli stessi risultati. Durante la descrizione di ogni passo del processo di integrazione,

vengono descritte tutte le funzionalità disponibili, così da rendere questo capitolo fruibile come un manuale per l’utente.

Capitolo 1

L'integrazione intelligente delle informazioni

La crescente diffusione delle infrastrutture di rete ha enormemente ampliato la quantità di informazioni a disposizione degli utenti, comportando un parallelo aumento della complessità nelle ricerche. Se aggiungiamo a questo dato anche la componente di eterogeneità che hanno queste sorgenti di dati, non è difficile immaginare come l'utente possa essere in difficoltà di fronte ad una ricerca complessa. Il grande numero di sorgenti eterogenee costringerebbe l'utente a conoscere in dettaglio le tecnologie legate ad ognuna di esse. Egli, invece, è interessato ad effettuare ricerche che portino a risultati accurati ed ottenuti con tempi di risposta accettabili.

Gli standard esistenti (TPC/IP, ODBC, OLE, CORBA, SQL, etc.) risolvono parzialmente i problemi relativi alle diversità hardware e software, dei protocolli di rete e di comunicazione tra i moduli; rimangono però irrisolti quelli relativi alla modellazione delle informazioni. Difatti i modelli e gli schemi dei dati sono differenti e questo crea una eterogeneità semantica (o logica) non risolvibile da questi standard.

Gli approcci all'integrazione presentano diverse metodologie: la *reingegnerizzazione* delle sorgenti mediante standardizzazione degli schemi e la creazione di un database distribuito; i *datawarehouse* che realizzano delle viste presso l'utente finale, replicando fisicamente i dati e utilizzando algoritmi di allineamento per assicurarne la consistenza a fronte di modifiche nelle sorgenti.

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

Nel seguito verrá descritto un tipo di approccio differente, che non ricorre alla duplicazione fisica dei dati, quello che in letteratura viene indicato come Integrazione di Informazioni (I^2) [1].

1.1 I sistemi I^2 e I^3

L'Integrazione delle Informazioni (I^2) va dunque distinta da quella dei dati (e dei DataBase), per ottenere risultati essa richiede *conoscenza* ed *intelligenza* volte all'individuazione delle sorgenti e dei dati, nonché alla loro fusione e sintesi.
Quando l'Integrazione di Informazioni fa uso di tecniche di Intelligenza Artificiale si parla allora di Integrazione Intelligente di Informazioni (***I**ntelligent **I**ntegration of **I**nformation*, I^3).

1.1.1 Il programa I^3

Il programma I^3 è stato ideato e realizzato dall' ARPA (Advanced Research Projects Agency) con lo scopo di produrre un sistema per produrre in maniera automatica l'integrazione di sorgenti di dati eterogenee [2]. Propone l'introduzione di architetture modulari sviluppabili secondo i principi proposti da uno standard che ponga le basi dei servizi da soddisfare dall'integrazione ed abbassi i costi di sviluppo e mantenimento. Questo renderebbe possibile ovviare ai problemi di realizzazione, manutenzione, adattabilità, inoltre la riutilizzazione della tecnologia già sviluppata, rende la costruzione di nuovi sistemi più veloce e meno difficoltosa, con conseguente abbassamento dei costi. Per poter sfruttare un'elevata riusabilità bisogna disporre di interfacce ed architetture standard. Il paradigma suggerito per la suddivisione dei servizi e delle risorse nei diversi moduli si articola su due dimensioni:

- l'orizzontale, divisa in tre livelli:
  - livello utente
  - moduli intermedi che fanno uso di tecniche di IA
  - risorse di dati
- la verticale: molti domini, con un numero limitato di sorgenti, generalmente minore di dieci.

In generale i diversi domini non sono strettamente connessi all'interno di un certo livello ma si scambiano informazioni e dati; la combinazione delle informazioni avviene a livello dell'utilizzatore riducendo la complessità totale del sistema e consentendo lo sviluppo di numerose applicazioni finalizzate a scopi diversi

fra loro. Ad esempio, si supponga di dover integrare informazioni sui trasporti mercantili, ferroviari e stradali: ciò permette all'utente di avere un'idea completa su quale mezzo di trasporto sia maggiormente vantaggioso ai propri fini. Aggiungendo a questo altri domini, quali le situazioni metereologiche ed i costi di immagazzinamento e trasporto, si possono facilitare le scelte di un dirigente che deve decidere come e quando consegnare delle merci.

I^3 si concentra sul livello intermedio della partizione, quello che media tra gli utenti e le sorgenti. Questo livello deve offrire servizi dinamici quali la selezione delle sorgenti, la gestione degli accessi e delle interrogazioni, l'analisi e sintesi dei dati.

Nell'Appendice A è presente un glossario di termini comunemente usato in ambito I^3, questo ha lo scopo di spiegare quei termini che dovessero risultare ambigui o poco chiari, visto il campo recente ed in evoluzione in cui si muove il progetto.

1.1.2 L'architettura di riferimento I^3

Alla base del progetto I^3 bisogna introdurre due ipotesi, citando il sito web [2]

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

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

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

E' importante evidenziare, 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.

1.1.3 I servizi I^3

La figura 1.1 illustra come siano suddivisi i servizi offerti dall'architettura. E' intuitivo evidenziare la suddivisione tra i due assi.

Lungo l'asse orizzontale si nota il rapporto tra i servizi di Coordinamento ed 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.

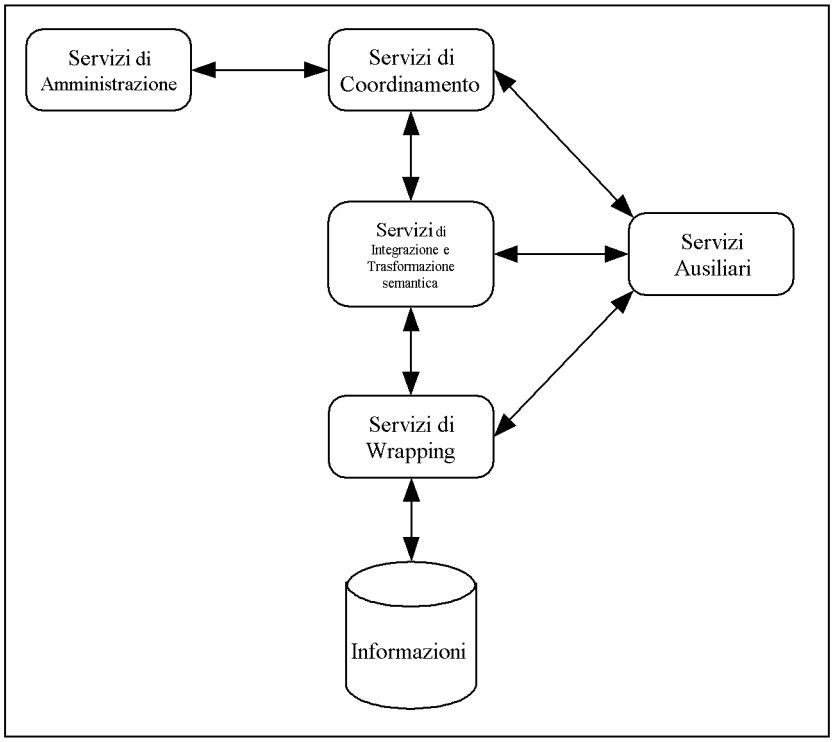

Figura 1.1: Diagramma dei servizi I^3

Se percorriamo l'asse verticale, si può intuire come avviene lo scambio di informazioni nel sistema: in particolare, i serivi di *wrapping* provvedono ad estrarre le informazioni dalle singole sorgenti, che sono poi impacchettate ed integrate dai Servizi di Integrazione e Trasformazioni Semantica, per poi essere passati ai servizi di Coordinamento che ne avevano fatto richiesta.

Analizziamo brevemente la funzione di ogni servizio:

- **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 selezio-

ne dinamica delle sorgenti (o brokering, per Integratori Intelligenti) al semplice *Matchmaking*, in cui il mappaggio tra informazioni integrate e locali è realizzato manualmente ed una volta per tutte. Vediamo alcuni esempi:

1. **Facilitation e Brokering Services**: l'utente manda una richiesta al sistema e questo usa un deposito di metadati per ritrovare il modulo che può trattare la richiesta direttamente. I moduli interessati da questa richiesta potranno essere uno solo alla volta (nel qual caso si parla di Brokering) o più di uno (e in questo secondo caso si tratta di facilitatori e mediatori, attraverso i quali a partire da una richiesta ne viene generata più di una da inviare singolarmente a differenti moduli che gestiscono sorgenti distinte, e reintegrando poi le risposte in modo da presentarle all'utente come se fossero state ricavate da un'unica fonte). In questo ultimo caso, in cui una query può essere decomposta in un insieme di sottoquery, si farà uso di servizi di Query Decomposition e di tecniche di Inferenza (mutuate dall'Intelligenza Artificiale) per una determinazione dinamica delle sorgenti da interrogare, a seconda delle condizioni poste nell'interrogazione.

   I vantaggi che questi servizi di Coordinamento portano stanno nel fatto che non è richiesta all'utente del sistema una conoscenza del contenuto delle diverse sorgenti, dandogli l'illusione di interagire con un sistema omogeneo che gestisce direttamente la sua richiesta. E' quindi esonerato dal conoscere i domini con i quali i vari moduli I^3 hanno a che fare, ottenendone una considerabile diminuzione di complessità di interazione col sistema.

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

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

  In alternativa a questi metodi dei Template, sono utilizzate le *Yellow Pages*, ovvero servizi di directory che mantengono le informazioni sul contenuto delle varie sorgenti e sul loro stato (attive, inattive, occupate). 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 i **servizi il Browsing** che permettono 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 anche i servizi di **Iterative Query Formulation** che aiutano l'utente a rilassare o meglio specificare alcuni vincoli della propria interrogazione per ottenere risposte più precise.

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

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

a fonti differenti, ed i loro risultati devono essere integrati. Comprendono inoltre tecniche di caching, per supportare la materializzazione delle viste (problematica molto comune nei sistemi che vanno sotto il nome di datawarehouse).

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

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

  In pratica, compito di un wrapper è modificare l'interfaccia, i dati ed il comportamento di una sorgente, per facilitarne la comunicazione con il mondo esterno.

  Il vero obiettivo è quindi standardizzare il processo di wrapping delle sorgenti, permettendo la creazione di una libreria di fonti accessibili; inoltre, il processo stesso di realizzazione di un wrapper dovrebbe essere standardizzato, in modo da poter essere riutilizzato per altre fonti.

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

1.2 Il mediatore

Secondo la definizione proposta da Wiederhold in [3] un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare infor-

mazioni per una applicazione di livello superiore... Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno, o al più pochi, esperti.
Un mediatore presenta allora i seguenti compiti:

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

Il progetto MOMIS, di cui questa tesi fa parte, ha come obiettivo la progettazione e realizzazione di un **Mediatore**, come descritto in [4, 5, 6].
L’ipotesi di avere a che fare esclusivamente con sorgenti di dati strutturati e semi-strutturati, ha consentito di restringere il campo applicativo del sistema con una conseguente diminuzione delle problematiche riscontrate in fase di progettazione e realizzazione. L’approccio architetturale scelto é quello classico, che si sviluppa su tre livelli principali:

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.

L’architettura del mediatore che si è progettato è riportata in Figura 1.2.In particolare, in questa sono state esaminate le seguenti funzionalità:

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

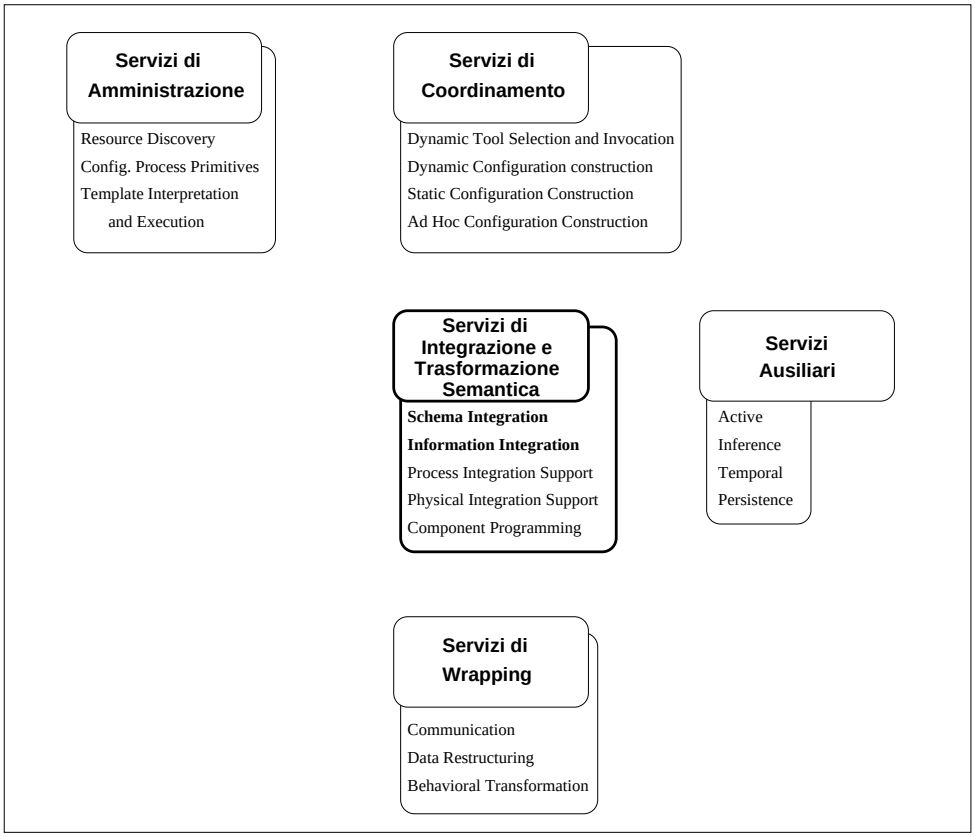

Figura 1.2: Servizi I^3 presenti nel mediatore

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

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

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, né tantomeno è banale realizzare una loro coerente integrazione. Mettendo da parte per un attimo le differenze dei sistemi fisici (alle quali dovrebbero pensare i moduli wrapper) i problemi che si è dovuto risolvere, o con

i quali occorre giungere a compromessi, sono (a livello di mediazione, ovvero di integrazione delle informazioni) essenzialmente di due tipi:

1. problemi ontologici.
2. problemi semantici.

1.2.2 Problemi ontologici

Per ontologia si intende, in questo ambito, "l'insieme dei termini e delle relazioni usate in un dominio, che denotano concetti ed oggetti" . Con ontologia quindi ci si riferisce a quell'insieme di termini che, in un particolare dominio applicativo, denotano in modo univoco una particolare conoscenza e fra i quali non esiste ambiguità poiché sono condivisi dall'intera comunità di utenti del dominio applicativo stesso. Non è certamente l'obiettivo né di questo paragrafo, né della tesi in generale, dare una descrizione esaustiva di cosa si intenda per ontologia e dei problemi che essa comporta (ancorché ristretti al campo dell'integrazione delle informazioni), ma mi limito a riportare una semplice classificazione delle ontologie (mutuata da Guarino [7, 8]), per inquadrare l'ambiente in cui ci si muove. I livelli di ontologia (e dunque le problematiche ad essi associate) sono essenzialmente tre:

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 dice che

i diversi sistemi usino esattamente gli stessi vocaboli per rappresentare questi concetti, né tantomeno le stesse strutture dati. Poiché infatti le diverse sorgenti sono state progettate e modellate da persone differenti, è molto improbabile che queste persone condividano la stessa concettualizzazione del mondo esterno, ovvero non esiste nella realtà una semantica univoca a cui chiunque possa riferirsi.

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

Come riportato in [9] la causa principale delle differenze semantiche si può identificare nelle diverse concettualizzazioni del mondo esterno che persone distinte possono avere, ma non è l'unica. Le differenze nei sistemi di DBMS possono portare all'uso di differenti modelli per la rappresentazione della porzione di mondo in questione: partendo così dalla stessa concettualizzazione, determinate relazioni tra concetti avranno strutture diverse a seconda che siano realizzate attraverso un modello relazionale, o ad oggetti.

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

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

2. eterogeneità tra le strutture delle classi: comprendono le differenze nei criteri di specializzazione, nelle strutture per realizzare una aggregazione, ed anche le discrepanze schematiche, quando 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 cosidetto arricchimento semantico, ovvero all’aggiungere esplicitamente ai dati tutte quelle informazioni che erano originariamente presenti solo come metadati negli schemi, dunque in un formato non interrogabile.

Capitolo 2

L'architettura di MOMIS

Tenendo presente tutte le problematiche relative al mediatore esposte nel capitolo 1, e un insieme di progetti preesistenti quali TSIMMISS [10, 11, 12, 13], GARLIC [14, 15] e SIMS [16, 17], si è giunti alla progettazione di un sistema intelligente di Integrazione delle Informazioni.

Recependo l'esigenza di avere ambienti software "$intelligenti$" e funzionali, in grado non solo di fornire accesso a grosse moli di dati ma soprattutto capaci di aumentare la qualità ed il valore delle informazioni ottenibili, le Università di Modena e Reggio e di Milano hanno avviato la realizzazione del sistema **MOMIS** (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources)

MOMIS è iniziato all'interno del progetto **MURST 40%** ***INTERDATA*** (97/98) e continua ad essere sviluppato nell'ambito del progetto **MURST D2I**, nel biennio 2000/2001. L'obiettivo del progetto è la realizzazione di uno strumento che, seguendo le linee guida tracciate nell'ambito I^3, permetta la reale integrazione di sorgenti distribuite, eterogenee sia strutturate che semistrutturate.

2.1 Scelte implementative

In letteratura sono stati presentati diversi "$approcci$" al problema dell'integrazione di database convenzionali, anche applicabili all'integrazione di dati semistrutturati. Per comprendere come MOMIS possa essere collocato rispetto ad altri sistemi esistenti, o in fase di sviluppo, è opportuno fare un'analisi di tali approcci.

2.1.1 Le proposte di integrazione

Di tali approcci, seguendo quanto esposto in [18], si può realizzare una prima categorizzazione sulla base del diverso approccio utilizzato per la risoluzione dei conflitti semantici.

Ad un'estremo dello spettro di soluzioni troviamo una proposta di standardizzazione dei database e della rappresentazioni dei dati, come ad esempio vien fatto in SAP [19]. Tale proposta si basa quindi sulla definizione di un *modello globale dei dati* mediante il quale deve essere fatta un reingegnerizzazione dei sistemi locali. Questa strada comporta ingenti investimenti ovviamente ed un forte grado di collaborazione e interazione, presupposti questi che possono essere trovati solo in contesti caratterizzati da un forte accentramento amministrativo.

All'estremo opposto troviamo sistemi che risolvono le eterogeneità presenti nelle sorgenti fornendo strumenti in grado di fornire all'utente esterno una visione omogenea degli schemi e delle informazioni. Questa soluzione preserva quindi l'autonomia delle sorgenti ed è l'unica strada percorribile quando è necessario un elevato grado di indipendenza dei singoli database.

Concentrando l'attenzione sul secondo tipo di approccio è possibile raffinare la classificazione sulla base del modo in cui vengono descritte le sorgenti ed i dati in esse contenuti.

Come descritto in [20] è possibile distinguere due approcci:

- *semantici*
- *strutturali*

Per quanto riguarda **l'approccio strutturale** (e tra questi l'esempio piú importante è senza dubbio costituito dal progetto TSIMMIS [5]) possiamo sottolineare l'impiego di 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. I punti caratterizzanti di tale approccio sono quindi:

- utilizzo di un modello autodescrittivo per trattare tutti i singoli oggetti presenti nel sistema, sopperendo all'eventuale mancanza degli schemi concettuali delle diverse sorgenti;
- inserimento delle informazioni semantiche in modo esplicito attraverso l'impiego di regole dichiarative (ed in particolare, in TSIMMIS, attraverso le MSL rule);
- utilizzo di un linguaggio self-describing che facilita l'integrazione anche e soprattutto di dati semi-strutturati;

Come è facile intuire, in questo modo si ha la possibilità di integrare in modo completamente trasparente al mediatore basi di dati fortemente eterogenee e magari mutevoli nel tempo. Oggetti simili provenienti da una stessa sorgente possono avere strutture differenti, semplificando quindi la trattazione di dati semistrutturati.

D'altro canto, però, l'assenza di schemi concettuali vincola le possibili interrogazioni ad un insieme predefinito dall'operatore (per queste viene preventivamente memorizzato un piano di accesso), limitando in questo modo la libertà di richieste all'utente del sistema ed inoltre non permette, in caso di database di grandi dimensioni, la realizzazione di una ottimizzazione semantica.

L'approccio semantico è invece caratterizzato dai seguenti aspetti:

- il mediatore dispone, per ogni sorgente, di uno schema concettuale;
- nello schema concettuale sono presenti oltre ai metadati anche informazioni semantiche che possono essere sfruttate sia nella fase di integrazione delle sorgenti, sia in quella di ottimizzazione delle interrogazioni;
- deve essere disponibile un modello comune per descrivere le informazioni da condividere (e dunque per descrivere anche i metadati);
- viene realizzata un'unificazione (parziale o totale) degli schemi concettuali per arrivare alla definizione di uno schema globale.

Lo *schema globale* sopra citato rappresenta una *vista integrata* delle sorgenti e tale vista può essere realizzata seguendo due approcci distinti, quello *materializzato* e quello *virtuale* [18].
Nella prima soluzione, adottata nei *datawarehouse* [21], le informazioni vengono raccolte in un database centralizzato: le interrogazioni possono quindi essere eseguite senza dover accedere alle sorgenti. Sebbene i tempi di risposta siano decisamente contenuti, la necessità di mantenere l'allineamento tra vista globale e sorgenti impone la definizione e l'impiego di complesse procedure per l'aggiornamento dei dati.

La seconda strategia si basa invece su un modello di *decomposizione* delle query che, analizzando le richieste, porti all'individuazione delle sorgenti "$interessanti$" e alla generazioni di sotto-interrogazioni che possano essere eseguite localmente. Lo schema globale deve poi disporre di tutte le informazioni necessarie alla ricombinazione, o *fusione*, dei dati ricevuti, in modo da ottenere informazioni significative, cioè al contempo complete e corrette.

2.1.2 L'approccio adottato

In base alla classificazione fatta possiamo dire che MOMIS segue un approccio *semantico* e *virtuale*. Partendo dagli schemi concettuali locali, con una metodologia *bottom up*, si arriva infatti a definire uno schema globale in grado di fornire un accesso integrato alle sorgenti. Tale schema è quindi arricchito di tutte quelle informazioni che permettono l'individuazione dei dati ed il loro reperimento direttamente dalle fonti di informazione.

Diverse sono le motivazioni che hanno spinto all'adozione di un approccio di questo tipo:

1. la presenza di una schema globale permette all'utente di formulare qualsiasi interrogazione che sia consistente con lo schema;

2. le informazioni semantiche che esso comprende possono contribuire ad una eventuale ottimizzazione delle interrogazioni;

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

4. la vista virtuale rende il sistema estremamente flessibile, in grado cioè di sopportare frequenti cambiamenti sia nel numero e tipo di sorgenti, sia nei loro contenuti (non occorre prevedere onerose politiche di allineamento);

Parallelamente a questa impostazione si è deciso di adottare, sia per la rappresentazione degli schemi che per la formulazione delle interrogazioni, un unico modello dei dati basato sul paradigma ad oggetti. Questa scelta è stata fatta per diverse ragioni:

1. la necessità di disporre di un linguaggio di interrogazione espressivo, in grado cioè di rappresentare quei concetti di alto livello fondamentali per l'estrazione di conoscenza da un insieme di dati;

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

3. ampi sforzi sono sono già stati realizzati per lo sviluppo di standard rivolti agli oggetti: CORBA [22] per lo scambio di oggetti attraverso sistemi diversi; ODMG-93 [23] (e con esso i modelli ODM e ODL per la descrizione degli schemi, e OQL come linguaggio di interrogazione) per gli object-oriented database;

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

5. la possibilità di tradurre, in modo automatico, i modelli ad oggetti in logiche descrittive (ad esempio OLCD) permette l'introduzione di comportamenti intelligenti di supporto all'operatore sia nella fase di integrazione sia in quella di interrogazione.

Queste scelte sono poi state tradotte in un modello dei dati ed in un'architettura.

2.2 Il Modello dei dati

Come si è detto all'interno del sistema è stato adottato un modello comune dei dati (ODM$_{I^3}$) di alto livello in modo da facilitare la comunicazione tra i Wrapper ed il Mediatore.

La base di partenza per la definizione di questo modello è rappresentata dalle raccomandazioni relative alla proposta di standardizzazione per linguaggi di mediazione, risultato del lavoro svolto in ambito I^3. Tali raccomandazioni sottolineano la necessità per un mediatore di poter gestire sorgenti con modelli complessi, come quello ad oggetti, e sorgenti molto più semplici come file di strutture: l'impiego di un formalismo il più possibile completo, e quindi in grado di rappresentare in modo adeguato tutte le possibili situazioni, risulta una possibile soluzione.

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

Le principali caratteristiche del linguaggio ODL$_{I^3}$sono:

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

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

- per ogni classe, il wrapper può indicare nome e tipo del sorgente di appartenenza;

- per le classi appartenenti ai sorgenti relazionali è possibile definire le chiavi candidate ed eventuali foreign key;

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

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

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

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

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

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

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

- dichiarazione di relazioni estensionali: Il sistema MOMIS integra gli schemi locali secondo criteri sia intensionali che estensionali. I conflitti intensionali rappresentano quelle incompatibilità derivanti dall'avere porzioni di schemi sovrapposte, ossia gli stessi aspetti del dominio applicativo rappresentati usando strutture differenti (vedi Capitolo 1, Sezione 1.2.1). Ciò che occorre fare è quindi fornire una rappresentazione unificata ed omogenea dei medesimi concetti descritti in sorgenti differenti.
  L'integrazione degli schemi non è però l'unico aspetto che occorre gestire

per ottenere un'effettiva integrazione di sorgenti eterogenee: è necessario risolvere anche i conflitti derivanti dalla sovrapposizione delle estensioni, cioè dalla presenza, in sorgenti diverse, di informazione relativa alla stessa entità del "mondo reale". Per arrivare ad un'integrazione corretta può non essere sufficiente fare una semplice unione delle estensioni delle classi, in quanto dati relativi alla stessa entità potrebbero essere presenti in più classi. Questi inconvenienti possono essere risolti soltanto gestendo in modo adeguato le relazioni fra le estensioni. Si introducono, a tale scopo, gli *assiomi estensionali* [24], i quali descrivono le relazioni insiemistiche esistenti fra le estensioni delle sorgenti. Ogni assioma estensionale definito *vincola* le classi coinvolte ad avere anche un legame intensionale. In particolare, facendo riferimento alla traduzione ODL_{I^3} degli assiomi estensionali, abbiamo:

- **C1** SYN_{Ext} **C2**: le istanze della classe C1 sono le stesse della classe C2; è semanticamente equivalente alle relazioni C1 ISA C2, C2 ISA C1 più una relazione intensionale C1 SYN C2;
- **C1** NT_{Ext} **C2**: le istanze della classe C1 sono un sottoinsieme di quelle della classe C2; è semanticamente equivalente alla relazione C1 ISA C2 più la relazione intensionale C1 NT C2;
- **C1** BT_{Ext} **C2**: le istanze della classe C1 sono un sovrainsieme di quelle della classe C2; è equivalente alla relazione C2 ISA C1 più la relazione intensionale C1 BT C2;
- **C1** $DISJ_{Ext}$ **C2**: le istanze della classe C1 sono diverse da quelle della classe C2; questo assioma non implica nessun tipo di relazione intensionale.

Utilizzando questo linguaggio (vedi Appendice B), il wrapper compie la traduzione delle classi da integrare e le fornisce al mediatore: da sottolineare che le descrizioni ricevute rappresentano tutte e sole le classi che una determinata sorgente vuole mettere a disposizione del sistema, e quindi interrogabili. Non è quindi detto che lo schema locale ricevuto dal mediatore rappresenti l'intera sorgente, bensì ne descrive il sottoinsieme di informazioni visibili da un utente del mediatore, esterno quindi alla sorgente stessa.

2.3 L'architettura di MOMIS

Nel sistema MOMIS, che ricordo adotta l'architettura di riferimento I^3 [2], i componenti sono disposti su tre livelli (come si puó vedere da Fig. 2.1):

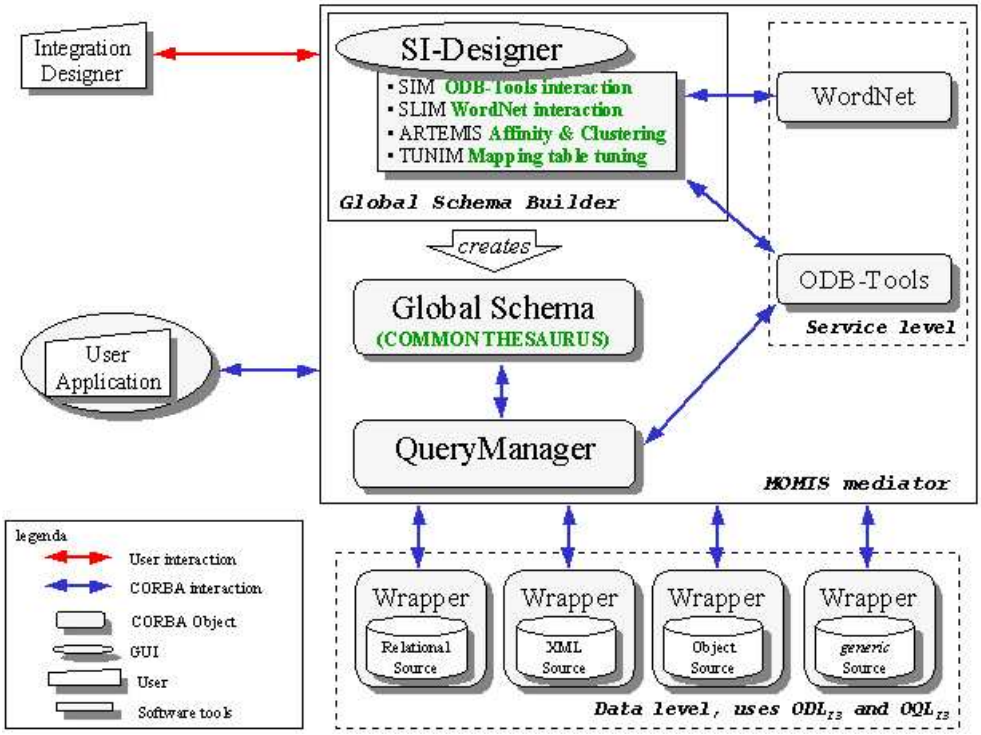

Figura 2.1: Architettura del sistema MOMIS

1. **Livello Dati.** Qui si trovano i **Wrapper**. Posti al di sopra di ciascuna sorgente, sono i moduli che fungono da interfaccia tra mediatore vero e proprio e le sorgenti locali di dati. La loro funzione è duplice:

   - in fase di integrazione, forniscono la descrizione delle informazioni contenute nelle sorgenti. Questa descrizione viene fornita attraverso il linguaggio ODL$_{I^3}$ (descritto in Sezione 2.2);
   - in fase di query processing, traducono la query ricevuta dal mediatore (espressa quindi nel linguaggio comune di interrogazione OQL$_{I^3}$, che è definito in analogia al linguaggio OQL) in una interrogazione comprensibile (e realizzabile) dalla sorgente stessa. Devono inoltre esportare i dati ricevuti in risposta all'interrogazione, presentandoli al mediatore attraverso il modello comune di dati utilizzato dal sistema;

2. **Livello Mediatore.** Il mediatore è il cuore del sistema, in grado di fornire una rappresentazione omogenea delle informazioni ed un accesso integrato alle sorgenti. Esso è composto da due moduli distinti:

- Global Schema Builder (GSB): è il modulo di integrazione degli schemi locali che, partendo dalle descrizioni delle sorgenti espresse attraverso il linguaggio ODL_{I^3}, genera un unico schema globale da presentare all'utente. Questa fase di integrazione, realizzata in modo semi-automatico con l'interazione del progettista del sistema, fa uso di ODB-Tools (componente esterno sfruttato dal modulo software **SIM**) e del DataBase lessicale WordNet (componente esterno utilizzato di SLIM).

  L'interfaccia grafica del Global Schema Builder è SI_Designer.

  Oggetto di questa tesi è lo studio di una nuova interfaccia grafica, che sostituisca SI_Designer e che possa essere utilizzata dalla rete con un comune browser web. Come vedremo in dettaglio nel capitolo 4 l'applicazione in questione si chiama **SIWeb** e fornisce all'utente le stesse funzionalità offerte dal vecchio SI_Designer.

- Query Manager (QM): è il modulo di gestione delle interrogazioni. Provvede a gestire la query dell'utente, deducendone da essa un insieme di *sottoquery* da spedire alle fonti locali, ed a ricomporre le informazioni da esse ricevute. Tra i suoi compiti vi è anche l'ottimizzazione semantica delle interrogazioni, realizzata utilizzando ODB-Tools.

3. **Livello Utente.** Il progettista interagisce col Global Schema Builder e crea la vista integrata delle sorgenti: l'utente formula le interrogazioni sullo schema globale passandole come input al Query Manager, che interrogherà le sorgenti e fornirà all'utente la risposta cercata.

Mediante questa struttura e queste funzionalità MOMIS è quindi in grado di fornire un accesso integrato ad informazioni eterogenee memorizzate sia in database di tipo tradizionale (e.g. relazionali, object-oriented) o file system, sia in sorgenti di tipo semistrutturato.

2.3.1 Il Global Schema Builder

L'integrazione delle sorgenti realizzata dal *Global Schema Builder*(GSB) si basa sull'individuazione di un'ontologia, comune alle diverse sorgenti, sotto forma di thesaurus: un insieme di relazioni terminologiche chiamato *Common Thesaurus*. Come mostrato in Figura 2.2, GSB è composto da quattro moduli:

- SIM (Sources Integrator Module): estrae automaticamente le relazioni intra-schema deducibili dalla struttura delle classi ODL_{I^3}; inoltre si occupa della "validazione semantica" e dell'inferenza di nuove relazioni attraverso il componente esterno ODB-Tools.

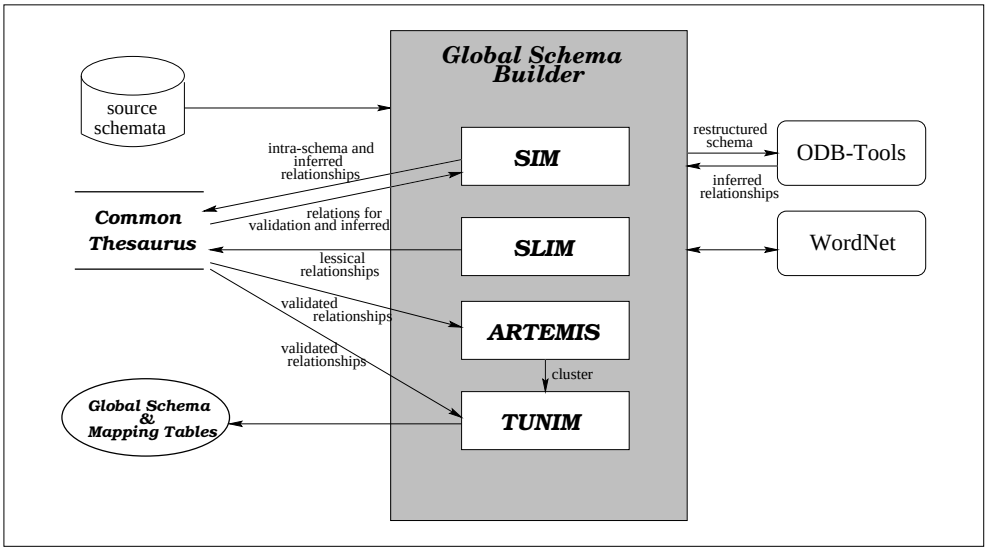

Figura 2.2: Architettura del Global Schema Builder

- SLIM (Schemata Lessical Integrator Module): estrae relazioni intensionali inter-schema tra nomi di attributi e classi ODL$_{I^3}$ utilizzando la conoscenza espressa in WordNet;
- ARTEMIS (Analisys and Reconciliation Tool Environment for Multiple Infomation Sources): è un tool basato su tecniche di affinità e di clustering che esegue l'analisi semantica ed il clustering delle classi ODL$_{I^3}$. L'analisi semantica ha come obiettivo l'identificazione degli elementi legati da relazioni semantiche nei diversi schemi. I coefficienti di affinità sono calcolati tra coppie di elementi per esprimere la loro similarità nel rappresentare la stessa informazione in schemi differenti;
- TUNIM (Tuning of the Mapping-Table): gestisce l'ultima fase di integrazione degli schemi per giungere allo schema globale. Partendo dalle relazioni del Common Thesaurus, per ciascuno dei cluster creati da ARTEMIS viene generata una classe globale. Ciascuna classe è caratterizzata da un insieme di attributi globali e da una mapping-table: l'insieme di attributi ne definisce la struttura mentre la mapping-table indica quali informazioni rappresenta ogni attributo globale.

Per completezza, è bene ricordare che il sistema MOMIS affronta problematiche relative anche alla conoscenza estensionale [24]. A tale proposito è stato progettato il modulo EHB (Extensional Hierarchy Builder), il quale individua ed elabora le relazioni estensionali tra classi locali. Tale modulo è composto da due

Sorgente UNIVERSITY (UNI)

```
Research_Staff(name,e_mail,dept_code,section_code)
School_Member(name,faculty,year)
Department(dept_name,dept_code,budget)
Section(section_name,section_code,length,room_code)
Room(room_code,seats_number,notes)
```

Sorgente COMPUTER_SCIENCE (CS)

```
CS_Person(first_name,last_name)
Professor:CS_Person(belongs_to:Office,rank)
Student:CS_Person(year,takes:set⟨Course⟩,rank)
Office(description,address:Location)
Location(city,street,number,county)
Course(course_name,taught_by:Professor)
```

Sorgente TAX_POSITION (TP)

```
Student(name,student_code,faculty_name,tax_fee)
ListOfStudent(Student:set⟨Student⟩)
```

Figura 2.3: Esempio di riferimento

parti ben distinte: la prima permette al progettista di specificare degli *assiomi estensionali* tra tutte le classi delle sorgenti, la seconda gestisce il calcolo delle *base extension* e della gerarchia estensionale, considerando una classe globale per volta.

Malgrado l'integrazione del suddetto modulo all'interno di SIWeb sia ancora stata implementata solo per quanto riguarda la prima parte, è comunque evidente come essa dovrà essere inserita nel Global Schema Builder. Ogni assioma estensionale di equivalenza ed inclusione implica, infatti, un legame intensionale molto forte tra le classi coinvolte (vedi Sezione 2.2); per tale motivo, due classi tra le quali è stato definito un assioma di questo tipo verranno *forzate* ad appartenere allo stesso cluster. Risulta così evidente la necessità di posizionare questo sottomodulo prima delle fasi di clustering e generazione delle mapping-table (TUNIM).

2.3.2 Esempio di riferimento

In Appendice C è riportata la descrizione ODL$_{I^3}$ delle sorgenti che costituiscono l'esempio di riferimento utilizzato per illustare le fasi del processo di integrazione. In Figura 2.3 lo stesso viene invece presentato in modo schematico.
Il contesto preso a modello è una realtà universitaria: le sorgenti da integrare sono tre.

La prima sorgente, `University` (UNI), è un DataBase di tipo relazionale, che contiene informazioni sullo staff e sugli studenti di una determinata università; le tabelle che lo compongono sono: `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Per ogni professore (presente nella tabella `Research_Staff`), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key `dept_code`), sul suo indirizzo di posta elettronica (`e_mail`), e sul corso da lui tenuto (`section_code`). Per il corso inoltre viene memorizzata l'aula (`Room`) dove questo si svolge, mentre del dipartimento è descritto, oltre al nome (`dept_name`) ed al codice (`dept_code`), il budget (`budget`) che ha a disposizione. Per gli studenti presenti nella tabella `School_Member` sono invece mantenuti il nome (`name`), la facoltà di appartenenza (`faculty`) e l'anno di corso (`year`).

La sorgente `Computer_Science` (CS) contiene invece informazioni sulle persone afferenti a questa facoltà, è un DataBase ad oggetti. Sono presenti sei classi: `CS_Person`, `Professor`, `Student`, `Office`, `Location` e `Course`. I dati mantenuti sono comunque abbastanza simili a quelli della sorgente UNI: per quanto riguarda i professori, sono memorizzati il livello(`rank`), e l'ufficio di appartenenza (`belongs_to`), che a sua volta fa parte di un dipartimento (e ne può quindi essere considerata una specializzazione); per gli studenti sono memorizzati i corsi seguiti (`takes`), l'anno di corso (`year`) ed il livello raggiunto (`rank`). Il corso ha poi un attributo complesso che lo lega al professore che ne è titolare (`taught_by`), mentre per l'ufficio si tiene l'indirizzo (`address`) e la descrizione (`description`).

È presente infine una terza sorgente, `Tax_Position` (TP), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (`tax_fee`). A tale sorgente appartengono `Student` e `ListOfStudent`. `Tax_Position` è una sorgente semistrutturata.

Capitolo 3

Tecnologie per il Word Wide Web

Uno dei settori sul quale maggiormente si focalizza l'attenzione dello scenario della IT è quello della programmazione web oriented, ovvero quella in cui la parte client è costituita da un semplice browser che interagisce con la parte server attraverso il protocollo HTTP.

In questo capitolo viene esposta una panoramica su quelle che sono le tecnologie disponibili focalizzando particolarmente l'attenzione su quelle legate all'ambiente Java.

3.1 Introduzione

Il linguaggio HTML, grazie alla sua semplicità e al largo supporto ricevuto dai produttori di browser web, si è rapidamente affermato come uno standard per la pubblicazione di documenti su web.

Nelle prime versioni del linguaggio i documenti HTML erano essenzialmente documenti statici. Pur avendo le notevoli capacità di un ipertesto di incorporare contenuti di varia natura e di stabilire collegamenti con altri documenti (link), le possibilità di interazione con l'utente restavano limitate. Per venire incontro all'esigenza di una maggiore interazione tra client e server e per fornire all'utente contenuti più vari e meno statici sono state via via introdotte svariate tecnologie, che introducono elementi dinamici nei siti web.

Anche se generalmente queste tecnologie comportano qualche supporto sia da parte del client che del server e del linguaggio, è possibile operare una classificazione sulla base del luogo in cui viene eseguito il codice che compie le operazioni e le elaborazioni che rendono il funzionamento dinamico.

In base a tale criterio si hanno le seguenti categorie:

- **Tecnologie "lato client"** nelle quali la dinamicità e l'interazione con l'utente sono gestite direttamente da codice eseguito dal client o da un suo plugin.

È il caso degli script supportati dai browser (JavaScript, VBScript, ecc.) e delle applet Java, che sono diventati elementi standard di una pagina web, con uno specifico supporto nel linguaggio HTML (interazione tra oggetti HTML e script, tag APPLET).

- **Tecnologie "lato server"** dove il client ha un ruolo essenzialmente passivo, ed è il server, o un suo plugin, a gestire la parte dinamica. Il caso più tipico è la comunicazione client-server per mezzo del protocollo HTTP, supportata in HTML dai form e dai suoi componenti. Attraverso questi componenti l'utente può inserire dei dati, inviarli al server e ottenerne una risposta. Il browser invia questi dati sotto forma di una richiesta HTTP, utilizzando uno dei metodi previsti dal protocollo (in genere get o post) a cui il server fa seguire una appropriata risposta. Naturalmente, perché il server sia in grado di elaborare la risposta, è necessario che supporti tecnologie adeguate. Poiché l'interazione avviene con richieste e risposte non predefinite, ma stabilite invece dagli sviluppatori di volta in volta, queste tecnologie devono innanzitutto fornire al programmatore i mezzi per scrivere sul lato server il codice che elabora le risposte (sul lato client il supporto per l'invio delle richieste è fornito dai form HTML). In queste categoria troviamo CGI, le Active Server Pages (ASP) di Microsoft e, in Java, le servlet e le Java Server Pages.

Noi ci dedicheremo essenzialmente alla descrizione delle tecnologie lato server.

3.2 CGI

In questo scenario, il lato server è composto tipicamente da una serie di applicazioni invocate dal client grazie all'interazione del server HTTP. La configurazione tipica fino a qualche tempo fa vedeva applicazioni installate nel server, scritte in C o ancor più frequentemente in PERL.

Questa tipologia di programmi, il cui modello di funzionamento viene tipicamente denominato **C**ommon **G**ateway **I**nterface (**CGI**), [25] ha come obiettivo quello di permettere l'interfacciamento da parte di un client web con le risorse e i servizi residenti sul server.

Dal punto di vista della logica di esecuzione, questa è la sequenza delle operazioni:

- il client tramite browser effettua una chiamata al web server;
- il server web provvede a eseguire l'applicazione (il programma scritto in PERL o in altro linguaggio);

- l'applicazione, dopo aver eseguito tutte le operazioni del caso, produce un output che viene passato al web server che lo invierà poi al client.

A questo punto è bene tener presente che sebbene CGI sia una definizione generica, tipicamente con tale sigla si tende a fare riferimento a programmi scritti utilizzando linguaggi come il C o il PERL, ovvero secondo una filosofia di progettazione ormai superata.

3.3 ASP

Active **S**erver **P**ages (**ASP**) è una tecnologia sviluppata da Microsoft [26] a supporto di altre sue tecnologie proprietarie, principalmente gli oggetti ActiveX e il linguaggio VBScript, simile al VisualBasic (sebbene ASP supporta anche Jscript, simile a JavaScript).

L'idea base di ASP è quella di dare la possibilità di inserire contenuti dinamici in una pagina web inframmezzando il codice HTML con codice in linguaggio script che viene eseguito dal server. Questa esecuzione ha come effetto sia il compimento di operazioni particolari nell'ambiente server (ad esempio la modifica di un database), sia eventualmente l'invio di dati al client (in genere in formato HTML), che vengono inseriti nella pagina ASP al posto degli script.

Gli script, da parte loro, possono sia eseguire direttamente delle operazioni, sia utilizzare oggetti ActiveX residenti sul server.

3.4 Servlet

Per la realizzazione di applicazioni CGI, Java mette a disposizione la servlet API che di fatto è diventata in poco tempo una delle più importanti di tutto il JDK.

La definizione generale di *Servlet* è quella di "componenti lato server per l'estensione di server Java enabled", cioè un qualcosa di più generale del solo WWW; in questo caso ci si concentra però solo sulle cosiddette servlet HTML.

Una servlet è quindi un programma Java in esecuzione sul lato server e in grado di colloquiare con il client per mezzo del protocollo HTTP.

Tipicamente questo si traduce nella possibilità di generare dinamicamente contenuti web da visualizzare nella finestra del browser del client.

I packages che contengono tutte le classi necessarie per la programmazione dele servlet sono il `javax.servlet` ed il `javax.servlet.http`: come si può intuire dal loro nome si tratta di Standard Extension API, ovvero non facenti parte formalmente del "JDK core".

Sebbene per quanto riguarda l'invocazione il meccanismo sia piuttosto simile al CGI, dal punto di vista del funzionamento si ha una situazione radicalmente

diversa. Infatti ogni volta che il client esegue una richiesta di un servizio basato su CGI, il web server deve mandare in esecuzione un processo dedicato per quel client. Se `n` client effettuano la stessa richiesta, allora `n` processi devono essere prima istanziati e poi eseguiti. Questo comporta un notevole dispendio di tempo e di risorse macchina.

Una servlet invece, una volta mandata in esecuzione (cioè inizializzata), può servire un numero `n` di richieste senza la necessità di ulteriori esecuzioni. In questo caso infatti il server manda in esecuzione una sola JVM, la quale effettua il caricamento dinamico delle classi necessarie ad eseguire le varie servlet invocate.

Il server da questo punto di vista offre una gestione ottimale dei processi poiché per ogni client che effettui una richiesta di servizio, server manderà in esecuzione un thread che eseguirà il metodo service (o, in maniera equivalente, il doGet() o doPost() a seconda dell'implemtenzione). Il tutto in modo automatico e trasparente agli occhi del programmatore che dovrà solo preoccuparsi di scrivere il codice relativo alle operazioni da effettuare in funzione della richiesta di un client.

Una servlet una volta correttamente compilata e installata nel server, segue un suo ciclo di vita ben preciso, composto dalla inizializzazione, dalla gestione delle invocazioni da parte dei client, e dalla conclusiva disintallazione. Ognuna di queste fasi è particolarmente importante, poiché condiziona sia la logica di funzionamento complessiva, sia le performance dell'applicazione nel suo complesso.

3.4.1 Inizializzazione

Il metodo `init()` ha lo scopo di effettuare tutte quelle operazioni necessarie per l'inizializzazione del servlet stesso, e per il corretto funzionamento successivo. La signature del metodo è la seguente

```
public void init(ServletConfig config) throws ServletException
```

L'inizializzazione avviene soltanto all'avvio dell'applicazione, che può coincidere con l'avvio del web server o alla prima richiesta di un client.

Le successive richieste verranno servite dal metodo `service()` senza eseguire nuovamente `init()`.

3.4.2 Gestione delle richieste

Dopo l'inzializzazione, una servlet si mette in attesa di una eventuale chiamata da parte del client, che potrà essere indistintamente una GET o una POST HTTP.

L'interfaccia Servlet mette a disposizione a questo scopo il metodo

```
public void service(HttpServletRequest req,
                    HttpServletResponse res) throws IOException
```

il quale come si è avuto modo di osservare in precedenza viene invocato direttamente dal server secondo una possibilità multithread.

Il metodo `service` viene invocato indistintamente sia nel caso di una invocazione tipo GET che di una POST. La ridefinizione del metodo `service` permette di modificare il comportamento della servlet stessa.

I due parametri passati sono di `ServletRequest` e `ServletResponse`, che permettono di interagire con la richiesta effettuata dal client e di inviare risposta tramite pacchetti HTTP.

Nel caso in cui invece si desideri implementare un controllo a grana più fine si possono utilizzare i due metodi `doGet()` e `doPost()`, che risponderanno rispettivamente ad una chiamata di tipo GET e a una di tipo POST.

Le signature dei metodi, molto simili a quella del metodo `service()`, sono:

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                  throws ServletException, IOException
```

Una servlet può comunicare con il client in maniera bidirezionale per mezzo delle due interfacce `HttpServletRequest` e `HttpServletResponse`: la prima rappresenta la richiesta e contiene i dati provenienti dal client, mentre la seconda rappresenta la risposta e permette di incapsulare tutto ciò che deve essere inviato indietro al client stesso.

Ecco di seguito un semplice esempio

```
public class MyServlet extends HttpServlet {
    public void service (HttpServletRequest req,
                         HttpServletResponse res)
                              throws ServletException {
        res.setContentType("text/html");
        res.setHeader("Pragma", "no-cache");
        res.writeHeaders();
        String param = req.getParameter("parametro");
        // esegue programma di servizio
        OutputStream out = res.getOutputStream();
        // usa out per scrivere sulla pagina di risposta
    }
}
```

3.4.3 Distruzione

A completamento del ciclo di vita, si trova la fase di distruzione della servlet, legata al metodo `destroy()`, il quale permette inoltre la terminazione del processo ed il log dello status.

La ridefinizione di tale metodo, derivato dalla interfaccia `Servlet` permette di specificare tutte le operazioni simmetriche alla inizializzazione, oltre a sincronizzare e rendere persistente lo stato della memoria.

Il codice che segue mostra quale sia il modo corretto di operare

```
public class myServlet extends HttpServlet {
    public void init(ServletConfig config) throws ServletException {
        // effettua operazioni di inizializzazione
        // per esempio l'apertura della connessione verso il db
    }
    public void destroy() {
        // effettua la chiusura della connessione
    }
}
```

È importante notare che la chiamata della `destroy()`, così come quella della `init()` è a carico del server che ha in gestione il servlet.

3.4.4 La gestione delle sessioni

Il concetto di sessione, introdotto da Java con le servlet permette di memorizzare le informazioni su client in modo piuttosto simile a quanto avviene ad esempio per i cookie: la differenza fondamentale è che le informazioni importanti sono memorizzate all'interno di una sessione salvata sul server e non sul client, al quale invece viene affidato un identificativo di sessione.

Poiché non si inviano dati al client come invece avviene con i cookie, si ha una drastica riduzione del traffico di dati in rete con benefici sia per le prestazioni che per la sicurezza.

Per associare il client con la sessione salvata, il server deve "marcare" il browser dell'utente: questo viene fatto inviando un piccolo e leggero cookie sul browser con il numero di identi ficazione della sessione, il **Session Id**.

Le interfacce che permettono la gestione delle sessioni, detto session tracking, sono la `HttpSession`, la `HttpSessionBindingListener` e la `HttpSessionContext`.

Per creare una session, si può utilizzare il metodo `getSession()` della classe `HttpServletRequest`: come ad esempio:

```
public class SessionServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response) throws ServletException,
                                                           IOException {
        // ricava o crea la sessione del client
        HttpSession session = request.getSession();
        out = response.getWriter();
    }
}
```

Una volta ottenuto l'oggetto `Session`, è possibile usarlo per memorizzare qualsiasi tipo di informazione relativo alla sessione logica che essa rappresenta, per mezzo degli opportuni metodi della `HttpSession`.

Ad esempio, per contare il numero di volte che una servlet viene invocata da un certo determinato client, sarà sufficiente scrivere

```
// si prova a ricavare dalla sessione
// il valore di sessiontest.counter
Integer ival = (Integer) session.getValue("sessiontest.counter");

// se la sessione non esiste il valore restituito sarà null
if (ival==null)
    // ival non è definito
    ival = new Integer(1);
else {
    // il valore di ival è definito
    // si incrementa di 1
    ival = new Integer(ival.intValue() + 1);
}

// si memorizza nella sessione il nuovo valore
session.putValue("sessiontest.counter", ival);
```

Dunque `session` è una specie di dizionario di valori a cui possiamo accedere da una qualsiasi servlet lanciata dallo stesso client.

3.5 Java Server Pages

JSP si ispira direttamente ad ASP, con somiglianze notevoli anche nella sintassi e nell'impostazione generale.

Rispetto alle tecnologie Microsoft, oltre ai vantaggi generici legati all'uso di Java che abbiamo già menzionato per i servlet, si ha quello di una reale portabilità in tutte le piattaforme su cui esista un'implementazione della macchina virtuale Java (ossia tutte quelle comunemente usate).

Ciò che accomuna le diverse tecnologie server side è che in tutti i casi si ha la generazione di pagine web (cioè, ora come ora, pagine HTML) come risultato di una determinata richiesta da parte del client. Cioè la pagina non esiste come risorsa statica, ma viene generata dinamicamente.

3.5.1 Il funzionamento delle JSP

Da un punto di vista funzionale una pagina JSP si può considerare come un file di testo scritto secondo le regole di un markup language in base al quale il contenuto del file viene elaborato da un **JSP container** (così viene chiamato un software per l'elaborazione di pagine JSP), per restituire il risultato di una trasformazione del testo originale, secondo le istruzioni inserite nel testo. Si tratta quindi di pagine

web a contenuto dinamico che viene generato al momento in cui la pagina viene richiesta dal client.

A differenza di interpreti di linguaggi come XML e HTML, un JSP container si comporta più come un template processor (o preprocessore, sul modello di quello del linguaggio C) che come un semplice markup processor. Infatti una pagina JSP è un file in formato testo che comprende essenzialmente due tipi di testo:

- template text ovvero testo "letterale" destinato a rimanere tale e quale dopo l'elaborazione della pagina;
- JSP text porzioni di testo che vengono interpretate ed elaborate dal JSP container.

Quindi in JSP solo alcune parti del testo vengono interpretate ed elaborate, mentre non ci sono restrizioni sul contenuto del template text.

In pratica, nella maggior parte dei casi, il template text è in formato HTML (o a volte XML), poiché le JSP sono pagine web.

Dal punto di vista del programmatore Java, invece, una Java Server Page può essere usata come una modalità per interfacciarsi a oggetti Java, in particolare a servlet e bean.

Infatti il JSP container converte la pagina JSP in un servlet, generando prima il codice sorgente, poi compilandolo. Il servlet così generato potrà interagire con componenti JavaBeans, secondo le istruzioni espressamente inserite nella pagina JSP. Per capire meglio come ciò avvenga, consideriamo il seguente esempio.

Questo è probabilmente l'esempio più semplice che si possa fare di una pagina JSP.

```
<html>
<head>
<title>Data e ora</title>
</head>
<body>
<p>
Data e ora corrente:<br>
<b><%= new java.util.Date() %></b>
</p>
</body>
</html>
```

Si tratta, come si vede, di una normale pagina HTML, con un solo elemento estraneo a questo linguaggio:

```
<%= new java.util.Date() %>
```

Si tratta di una espressione JSP, in linguaggio Java, che verrà interpretata e valutata dal JSP container, restituendone il risultato. Se eseguiamo la pagina JSP, vedremo che questa espressione è stata sostituita dalla data e ora corrente, anche

se in un formato standard che non è probabilmente quello che normalmente ci serve.

Questo significa che l'espressione `new java.util.Date()` è stata eseguita e convertita in stringa.

Quindi, prescindendo dai dettagli implementativi, l'esecuzione di una pagina JSP prevede due fasi distinte:

- fase di traduzione-compilazione, durante la quale viene costruito un oggetto eseguibile dalla VM (in genere una servlet) in grado di elaborare una risposta alle richieste implicite o esplicite contenute nella pagina;
- fase di processing, in cui viene mandato in esecuzione il codice generato e viene effettivamente elaborata e restituita la risposta.

Finora si sono visti solo alcuni degli elementi di una pagina JSP: il template text e le espressioni. Di seguito si fornisce un elenco completo di tali elementi, con una breve descrizione di ciascuno.

- **Template text** Tutte le parti di testo che non sono definite come elementi JSP; vengono copiate tale e quali nella pagina di risposta.
- **Comment** Con sintassi: `<%-- comment --%>`; sono commenti che riguardano la pagina JSP in quanto tale, e pertanto vengono eliminati dal JSP container nella fase di traduzione–compilazione; da non confondere con i commenti HTML e XML, che vengono inclusi nella risposta come normale template text.
- **Directive** Con sintassi: `<%@ directive ... %>`; si tratta di direttive di carattere generale, indipendenti dal contenuto specifico della pagina, relative alla fase di traduzione-compilazione.
- **Action** Seguono la sintassi dei tag XML ossia `<tag attributes> body </tag>` oppure `<tag attributes/>` dove attributes sono nella forma `attr1 = value1 attr2 = value2 ...`; le azioni sono eseguite nella fase di processing, e danno origine a codice Java specifico per la loro esecuzione.
- **Scripting element** Sono porzioni di codice in uno scripting language specificato nelle direttive; il linguaggio di default è lo stesso Java. Si dividono in tre sottotipi:
  - *Scriptlet* Con sintassi `<% code %>`; sono porzioni di codice nello scripting language che danno origine a porzioni di codice Java; generalmente inserite nel metodo `service()` della servlet; se contengono dichiarazioni di variabili queste saranno variabili locali valide solo

nell'ambito di una singola esecuzione della servlet; se il codice contiene istruzioni che scrivono sullo stream di output, il contenuto inviato in output sarà inserito nella pagina di risposta nella stessa poszione in cui si trova lo scriptlet.

– *Declaration* Con sintassi `<%! declaration [declaration] ... %>`; sono dichiarazioni che vengono inserite nella servlet come elementi della classe, al di fuori di qualunque metodo. Possono essere sia variabili di classe che metodi. Se si tratta di variabili, la loro durata è quella della servlet stessa; di conseguenza sopravvivono e conservano il loro valore nel corso di tutte le esecuzioni dello stesso oggetto servlet.

– *Expression* Con sintassi `<%= expression %>`; contengono un'espressione che segue le regole delle espressioni dello scripting language; l'espressione viene valutata e scritta nella pagina di risposta nella posizione corrispondente a quella dell'espressione JSP.

3.5.2 Un piccolo esempio

Quello che segue è solo un piccolo esempio per mostrare l'uso degli elementi JSP in una pagina.

```
<html>

<head>
<title>Data e ora</title>
</head>

<%@ page import = "java.util.*, java.text.*" %>

<%!
String DateFormat = "EEEE d MMMM yyyy";
DateFormat dateFormat = new SimpleDateFormat(DateFormat);
DateFormat timeFormat = new SimpleDateFormat("H:mm");
%>

<% Date dateTime = new Date(); %>

<body style = "fontsize: 16pt;">
Oggi &egrave; <%= dateFormat.format(dateTime) %><br>
e sono le ore <%= timeFormat.format(dateTime) %>
</body>

</html>
```

Il primo elemento JSP è una direttiva page di cui si specifica l'attributo `import` per importare dei package Java.

Poi c'è una JSP declaration con cui si creano due oggetti di tipo `DateFormat`. Si usano le dichiarazioni JSP perché si desidera creare questi

oggetti solo all'inizio e non a ogni esecuzione, dato che il formato della data e dell'ora restano costanti.

Si usa invece uno scriptlet di una sola riga per creare un oggetto di tipo `Date`. Poiché il valore della data e dell'ora corrente vengono impostate al momento della creazione, è necessario creare un'istanza a ogni esecuzione.

Gli ultimi due elementi sono delle espressioni JSP con le quali visualizziamo la data e l'ora.

3.5.3 JSP e JavaBeans

I componenti JavaBeans sono probabilmente il mezzo migliore, più pulito ed elegante per inserire contenuti dinamici in una pagina JSP. Mentre gli script finiscono con lo "sporcare" una pagina JSP mischiando tag HTML o XML e codice del linguaggio script, i bean, interfacciati per mezzo delle action, si integrano benissimo nel contesto della pagina che in tal modo contiene solo tag in un formato familiare a tutti i creatori di pagine web. In questo modo si ha la massima separazione del codice dalla presentazione e di conseguenza la massima manutenibilità.

I tag per la manipolazione di bean sono tre:

- `jsp:useBean` serve per utilizzare un bean già esistente o creare una nuova istanza di un bean.
- `jsp:getProperty` inserisce nella pagina il valore di una proprietà del bean.
- `jsp:setProperty` assegna il valore di una o più proprietà del bean.

Si riporta di seguito un semplice esempio. Si tratta della versione "bean" dell'esempio "Data e ora" precedentemente realizzato con gli script element.

```
FILE: datetimebean.jsp

<html>
<head>
<title>Data e ora</title>
</head>

<jsp:useBean id = "dateTime" class = "dates.DateTime" />

<body style = "fontsize: 16pt;">

Oggi &egrave; <jsp:getProperty name = "dateTime" property = "date"/><br>
e sono le ore <jsp:getProperty name = "dateTime" property = "time"/>

</body>

</html>
```

Nell'azione `jsp:useBean` si specifica il nome dell'oggetto `(dateTime)` e quello della classe (la classe `DateTime` del package `dates`), mentre nelle azioni `jsp:getProperty` specifichiamo sempre il nome dell'oggetto e quello della proprietà (`date` e `time` rispettivamente)

Anche in un esempio così semplice si può apprezzare la differenza tra la versione script e quella bean: la pagina si presenta molto più pulita e compatta e il contenuto risulta di facile e intuitiva comprensione.

Il JavaBean invocato è il seguente:

```
FILE: DateTime.java

import java.util.*;
import java.text.*;

public class DateTime {
    DateFormat dateFormat = new SimpleDateFormat("EEEE d MMMM yyyy");
    DateFormat timeFormat = new SimpleDateFormat("H:mm");

    public String getDate() {
        Date date = new Date();
        return dateFormat.format(date);
    }
    public String getTime() {
        Date date = new Date();
        return timeFormat.format(date);
    }
}
```

Tutto il codice che prima era sparso per la pagina è ora incapsulato in una classe Java che segue le specifiche dei JavaBeans.

3.6 Web Applications

Introdotte con la versione 2.2 della servlet API [27] le Web Application sono forse una delle novità più interessanti introdotte da Sun, anche se non sono esclusivamente legate ai servlet dato che comprendono anche immagini, suoni o quanto altro possa essere utilizzato per comporre pagine web ed infine anche applicazioni, applet, componenti in esecuzione sul client.

Una Web Application, come definita nella Servlet Specification, è una collezione di servlet, Java Server Pages, Enterprise Java Beans, classi Java di utilità, documenti statici come ad esempio pagine HTML, raggruppati all'interno di un archivio JAR in modo da semplificare al massimo le operazioni di installazione e configurazione. Infatti, se in passato per installare una applicazione basata su servlet era necessario copiare una serie di file ed editarne altri per poter configurare il corretto funzionamento della applicazione, adesso con le Web Application è sufficiente copiare un unico file JAR in una directory opportuna del server: all'interno di tale file (che ha estensione .war) sono contenute tutte le risorse necessarie

alla applicazione (.class, .html, immagini) ma anche tutte le informazioni per la configurazione dell'applicazione stessa. Ad esempio per poter mappare un servlet con un nome logico e poterlo invocare da remoto, adesso è sufficiente editare un opportuno file XML dove sono indicati sia il nome del servlet che i vari parametri di configurazione.

Chiunque abbia provato a utilizzare la vecchia modalità di installazione e configurazione si renderà conto di quanto questo possa essere comodo e vantaggioso.

3.6.1 I file .WAR

Il file JAR che contiene una Web Application in questo caso prende il nome di Web Application Archive (WAR file) e ha estensione .war. Una tipica struttura di un file di questo tipo potrebbe essere ad esempio

```
index.jsp
images/logo.gif
images/SIWeb_logo.gif
WEB-INF/web.xml
WEB-INF/lib/jspbean.jar
WEB-INF/classes/MiaServlet.class
WEB-INF/classes/sam/SourcesAcquisition.class
WEB-INF/classes/myjavabeans/UnJavaBean.class
```

Al momento della installazione, il file .war deve essere posizionato in una directory mappata poi dal server HTTP con un URI particolare. Tutte le richieste inoltrate a tale URI saranno poi gestite dalla applicazione contenuta in tale file WAR.

3.6.2 La directory WEB-INF

La directory WEB-INF ha un compito piuttosto particolare all'interno di una Web Application: i file qui contenuti infatti non sono accessibili direttamente dai client, ma sono utilizzati dal server per configurare l'applicazione stessa. Il suo funzionamento è quindi molto simile a quello della directory META-INF di un normale file JAR. La sottodirectory `WEB-INF/classes` contiene tutti i file compilati dei vari servlet e delle classi di supporto; la sottodirectory `WEB-INF/lib` invece contiene altre classi che sono però memorizzate in archivi JAR. Tutte le classi contenute in queste due directory sono caricate automaticamente dal server al momento del load della applicazione stessa. I servlet presenti nella directory WEB-INF possono essere invocati tramite un'invocazione che, nel caso dell'esempio appena visto, potrebbe essere

```
/SIWeb/servlet/MiaServlet
/SIWeb/servlet/sam/SourcesAcquisition
```

Il file `web.xml` contenuto nella dir WEB-INF è noto come **deployment descriptor file** e difatti contiene tutte le informazioni relative all'applicazione in cui risiede. Si tratta di un file XML il cui DTD è fissato da Sun con una specifica

che indica oltre 50 tag con i quali poter specificare una qualsiasi delle seguenti informazioni:

- le icone per la rappresentazione grafica dell'applicazione;
- la descrizione dell'applicazione;
- un flag che indica se la applicazione è distributed oppure no, ovvero se tale applicazione può essere condivisa fra diversi server remoti. I motivi per cui possa essere utile realizzare un'applicazione distribuita sono molti, dei quali forse il più intuitivo è legato a una maggiore flessibilità e alla possibilità di implementare tecniche di balancing e carico computazionale ripartito dinamicamente. Le regole per progettare e scrivere un'applicazione distribuita sono molteplici, ed esulano dallo scopo di questa tesi, per cui si rimanda per maggiori approfondimenti alla bibliografia ufficiale [28] [29];
- i parametri di inizializzazione della applicazione complessiva o delle singole servlet;
- la registrazione del nome o dei nomi di una servlet in esecuzione: possibilità questa che semplifica e soprattutto standardizza le operazioni;
- l'ordine di caricamento delle servlet;
- le regole di mapping fra le varie servlet ed i relativi URL;
- il timeout delle varie sessioni, opzione resa possibile grazie ai nuovi metodi di gestione delle sessioni introdotti con la API 2.2;
- il welcome file list, ovvero la sequenza dei file da utilizzare per la risposta da inviare al client (p.e.: index.htm, index.html o welcome.htm);
- le regole per la gestione degli errori, tramite le quali specificare fra l'altro anche quali pagine HTML (statiche o dinamiche) debbano essere visualizzate in corrispondenza del verificarsi di un errore del server (una pagina non trovata) o di una eccezione prodotta dal servlet engine;
- i riferimenti aggiuntivi alle tabelle di lookup utilizzate durante i riferimenti con JNDI a risorse remote;
- le regole di policy, tramite le quali specificare vincoli aggiuntivi di sicurezza.

La struttura del file XML non è importante di per sé, ma è importante il fatto che la sua presenza permette finalmente la possibilità di automatizzare e standardizzare il processo di installazione e deploy della applicazione così come di un gruppo di servlet.

Grazie al meccanismo di installazione e deploy messo in atto con l'utilizzo dei file `.war`, si parla sempre più spesso di **Pluggable Web Components** [29], ovvero di componenti con funzionalità specifiche installabili in un application server in modo molto semplice e veloce.

Si pensi ad esempio alla possibilità di attivare un motore di ricerca semplicemente copiando un file in una directory opportuna, indipendentemente dal sistema operativo e dal server. Parallelamente, una azienda che desideri attivare servizi di hosting a pagamento, potrà utilizzare il sistema delle Web Applications e dei file .war per semplificare la gestione degli utenti e dei vari domini governati da servlet.

3.7 Corba

CORBA [30] (*Common Object Request Broker Architecture*) è un'architettura standard, distribuita e ad oggetti sviluppata dall'Object Management Group (OMG). Dal 1989 l'obiettivo del gruppo OMG è stato la progettazione di una architettura per un *software bus* aperto, chiamato *Object Request Broker* (ORB), sul quale oggetti diversi potessero interagire via rete, indipendentemente dal sistema operativo in cui sono stati implementati. Questo standard permette a più oggetti di invocarne altri senza conoscerne l'esatta locazione o in quale linguaggio sono stati implementati.

Il linguaggio utilizzato per definire un oggetto CORBA è l'IDL (*Interface Definition Language*) mentre gli ORB comunicano attraverso il protocollo standard IIOP (*Internet InterORB Protocol*), definito sempre dall'OMG.

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

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

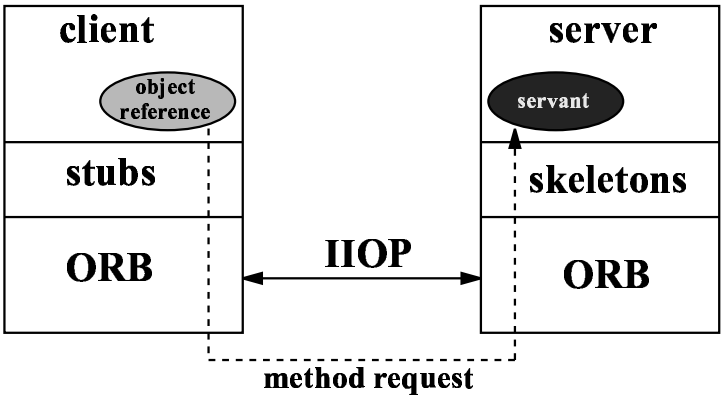

Figura 3.1: La invocazione di un metodo di un oggetto CORBA remoto

3.7.1 Come funziona

Il diagramma di Figura 3.1 mostra come un client manda un *messaggio* (inteso come esecuzione di un metodo di un altro oggetto) ad un oggetto CORBA implementato in un server, chiamato *servant-object*. Un client può essere un qualsiasi programma (anche un oggetto CORBA) che invoca un metodo di un servant-object.

Per invocare il metodo, il client utilizza un *object reference* dell'oggetto CORBA che vuole invocare. Se l'oggetto CORBA è locale, l'object reference è un puntatore ad un oggetto altrimenti, se l'oggetto CORBA è remoto, l'object reference punta ad una *stub function* senza che il client se ne accorga: per il client l'object reference è *sempre* un puntatore ad un oggetto. É l'apparato basato sugli ORB che rende possibile tutto questo.

L'invocazione di un metodo di un oggetto CORBA remoto da parte di un client avviene in questo modo:

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

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

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

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

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

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

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

3.7.2 Il Naming Service

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

In particolare, quello che occorre fare è attivare un *naming server* (un'applicazione fornita assieme alle librerie che permettono di creare oggetti CORBA in uno specifico linguaggio) sulla macchina in cui si vogliono creare oggetti CORBA accessibili in remoto: ogni oggetto CORBA creato dovrà poi registrarsi nel naming server, che gestisce il registro degli oggetti CORBA su quella macchina.

I nomi degli oggetti possono essere organizzati in una struttura ad albero proprio come i file sono organizzati in directory. Per accedere ad un determinato oggetto CORBA , il client esegue due sole operazioni:

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

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

Per esempio, nella Figura 3.2 è rappresentata la struttura ad albero memorizzata presso un naming server: si notano i *naming context* (equivalenti alle directory per i file system) *Initial Naming Context* (sempre presente) e *Personal*, mentre gli oggetti CORBA registrati sono `plans`, `calendar` e `schedule`. Per accedere all'oggetto CORBA `calendar` il client dovrà prima chiedere al naming server di accedere al naming context *Personal* e poi l'oggetto di nome `calendar`.

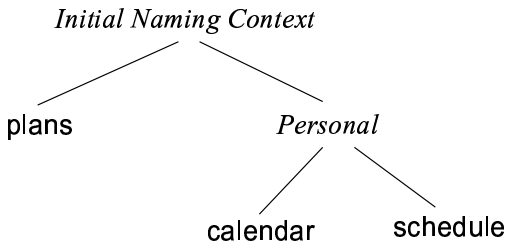

Figura 3.2: Esempio di albero creato dal naming server

3.7.3 La creazione di oggetti CORBA in Java

Per creare un oggetto CORBA occorre definire quali sono le loro interfaccie. Per fare questo si utilizza il linguaggio IDL, **I**nterface **D**efinition **L**anguage. Con un semplice tool messo a disposizione dalla Sun (`idltojava`) le definizioni delle interfaccie IDL vengono tradotte nelle corrispondenti espresse in linguaggio Java, assieme ad una serie di classi che permetteranno l'implementazione dell'oggetto CORBA desiderato in modo semplice. La Figura 3.3 mostra la dichiarazione IDL di un oggetto CORBA **Wrapper** e la corrispondente traduzione in Java. Definendo poi una classe Java che implementa l'interfaccia creata si può creare l'oggetto CORBA: naturalmente, occorrerà scrivere il codice dei metodi dichiarati nell'interfaccia.

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

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

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

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

Figura 3.3: Traduzione in Java di una interfaccia IDL di un oggetto CORBA

Capitolo 4

SIWeb: L'interfaccia internet per MOMIS

4.1 Introduzione alle applicazioni web

La realizzazione di applicazioni Web presenta alcune problematiche sconosciute alle applicazioni tradizionali. Queste problematiche derivano dalla natura stessa del Web, pensato originariamente non per la realizzazione di applicazioni ma per la distribuzione di documenti. Un classico problema è quello del cosiddetto mantenimento dello stato dell'applicazione. Il modello di interazione di base del Web è privo del concetto di connessione (poiché il protocollo HTTP è *stateless*), per cui l'esecuzione di un'applicazione Web consiste in una serie di interazioni disconnesse.

Lo sviluppatore deve utilizzare informazioni codificate nella richiesta HTTP e nelle risorse condivise durante la sequenza di interazioni per stabilire la continuità dell'applicazione.

La progettazione di un'applicazione Web deve tenere conto anche dell'ampiezza di banda di trasmissione disponibile e del carico di lavoro stimato per il server.

Dalle considerazioni sull'ampiezza di banda derivano le scelte relative alla quantità di dati da inviare ad un client in risposta ad una richiesta.

Ciò può coinvolgere anche l'interfaccia utente dell'applicazione stessa, portando in certi casi a rinunciare ad una veste grafica accattivante pur di fornire tempi di risposta accettabili.

Tuttavia, all'ottimizzazione dell'uso dei mezzi trasmissivi può contribuire, e in certi casi essere determinante, un'attenta ripartizione del carico di elaborazione tra client e server. Infatti, consentire al client di effettuare alcune elaborazioni senza coinvolgere il server, come ad esempio la convalida dei dati inseriti in una

form, permette di ridurre il traffico sulla rete sia di snellire il carico di lavoro del server.

Inoltre, la presenza di applicazioni esterne particolarmente complesse sulla stessa macchina del server Web può rendere meno efficiente l'applicazione Web sottraendole risorse preziose. In questi casi è opportuno distribuire il carico di lavoro su macchine diverse prevedendo, ad esempio, una macchina diversa per un DBMS o per un'applicazione particolarmente esosa di risorse.

Le applicazioni Web si pongono come valida alternativa alle tradizionali applicazioni Client/Server per vari motivi:

- **facilità di distribuzione e aggiornamento**: un'applicazione Web si trova interamente sul server, per cui la pubblicazione sul server coincide con la distribuzione e l'aggiornamento effettuato sul server è automaticamente reso disponibile a tutti gli utenti;
- **accesso multipiattaforma**: l'accesso all'applicazione è indipendente dall'hardware e dal sistema operativo utilizzato dagli utenti;
- **riduzione del costo di gestione**: l'uso di Internet come infrastruttura per un'applicazione Web riduce notevolmente sia i costi di connettività che i costi di gestione dei client;
- **scalabilità**: un'applicazione Web ben progettata può crescere insieme alle esigenze senza particolari problemi.

4.2 Sistemi a tre livelli

Un'applicazione Web, nella maggior parte dei casi, si sviluppa su tre livelli logico-funzionali (applicazioni Three-Tier) ma che possono essere distribuiti anche su più livelli (applicazioni Multi-Tier):

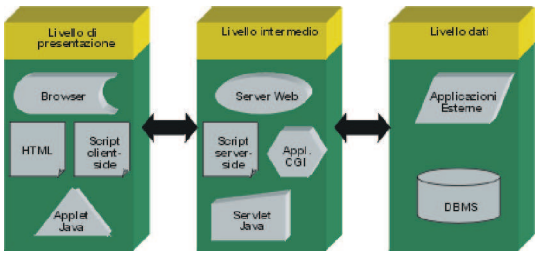

Figura 4.1: Sistemi a tre livelli

- **livello di presentazione** - rappresenta l'interfaccia utente dell'applicazione e si occupa di acquisire dati e visualizzare risultati;
- **livello intermedio** - si occupa delle elaborazioni dei dati in base alla cosiddetta busines logic, cioè all'insieme delle regole per cui i dati sono considerati significativi e le loro relazioni consistenti; le elaborazioni del livello intermedio generano i risultati richiesti dall'utente;
- **livello dati** - rappresenta l'insieme dei servizi offerti da applicazioni indipendenti dal Web, come ad esempio un gestore di database, un sistema di gestione di posta elettronica, ecc.

4.3 Conversione dell'applicazione

Avendo come punto di partenza una applicazione già realizzata e perfettamente funzionante (SI-Designer), un grande sforzo è stato fatto per riutilizzare gran parte del codice.

Dopo una prima analisi del codice sorgente del programma è risultato evidente come l'interfaccia utente non fosse separata dalla logica applicativa.

Il progetto era organizzato per separare i singoli moduli uno dall'altro, rendendo del resto il programma agilmente ampliabile con altre fasi del processo di integrazione delle sorgenti. Non era però separata la presentazione dalla business logic all'interno dei singoli moduli. Questo ha reso problematica la conversione dell'applicazione.

Sono state analizzate essenzialmente due alternative:

- convertire l'applicazione in una applet, in modo da lasciare intatta la quasi totalità del codice, fornendo quindi sul web una interfaccia identica alla preesistente;
- reingegnerizzare l'intera applicazione sviluppandola tenendo separati i tre livelli: dati, logica e presentazione.

Analizziamo più in dettaglio queste due scelte, anticipando fin d'ora che l'opzione migliore si è rivelata la seconda.

4.3.1 Realizzazione di una Applet

SI-Designer è stato scritto utilizzando le classi facenti parte del package grafico *Swing* che, purtroppo, non è compatibile (al momento) con le applet. Sun ha rilasciato un *plugin* da installare nelle macchine client per abilitare la grafica Swing nelle Applet. Purtroppo questo tipo di plugin non è abbastanza diffuso e non viene

installato di default nelle configurazioni standard dei più diffusi sistemi operativi e dei browser.

L'ipotesi di convertire la grafica utilizzando un package compatibile (ovvero AWT) è stata scartata a priori in quanto avrebbe comportato una completa riscrittura del codice, vanificando gli sforzi mirati alla riusabilità dei sorgenti disponibili.

Un ulteriore punto a sfavore dello sviluppo della Applet è la gestione della sicurezza.

Le applet vengono scaricate sul client al momento della richiesta, e vengono eseguite visualizzando l'interfaccia nel browser. Il modello di sicurezza delle Applet prevede che l'esecuzione dell'applicazione avvenga in un contesto protetto, detto *sandbox* dove ci sono una serie di regole da seguire. In particolare tra i numerosi vincoli che il modello prevede (e che non elenchiamo in questa sede) ci interessano i seguenti:

- la applet non può leggere e scrivere nel filesystem della macchina client;
- la applet può comunicare soltanto con il server dalla quale l'applet stessa è stata scaricata

La prima delle due regole può essere scavalcata chiedendo l'autorizzazione al client di dare i permessi sul controllo della macchina, quindi anche in lettura e scrittura sul filesystem. Questo tipo di autorizzazione prevede una serie di *alert* ovvero schermate di avvertimento all'utente, al momento del primo caricamento dell'applicazione. Essendo tra l'altro avvertimenti generici nei quali viene richiesto di autorizzare l'applet ad un controllo totale sulla macchina, potrebbe anche "spaventare" l'utente che non conosce esattamente il programma. Egli dovrebbe quindi "fidarsi" ed accettare incondizionatamente tutte le autorizzazioni. Questa soluzione ci sembra del resto poco pratica in previsione di una diffusione dell'applicazione ad un pubblico non necessariamente esperto.

La seconda regola implicherebbe invece l'installazione dei wrapper e del mediatore sulla stessa macchina sulla quale risiede il web server che ospita le applet. Questo vincolo ostacolerebbe notevolmente la scalabilità del sistema.

Ultimo punto, ma non meno importante dei precedenti riguarda le prestazioni del sistema. Ipotizzando di aver risolto tutti gli altri ostacoli, si produrrebbe una applet di notevoli dimensioni, che comporterebbe notevoli disagi per gli utenti. La larghezza di banda delle connessioni può variare molto, da alcuni kilobit al secondo per i collegamenti via modem fino ai megabit per secondo delle connessioni dedicate (ADSL, HDSL, CDN ecc...). Coloro i quali hanno una connessione troppo lenta si troverebbero a dover scaricare una applicazione probabilmente di diverse centinaia di kilobyte. Inoltre se la macchina client non è sufficientemente prestante, l'esecuzione potrebbe risentirne.

Riassumiamo quindi quanto detto nei seguenti punti:

- applicazione realizzata senza separazione tra logica e presentazione;
- interfaccia grafica realizzata con Swing, non compatibile con le applet;
- problemi sulla gestione della sicurezza;
- problemi di scalabilità;
- scarse prestazioni.

Queste motivazioni sono sufficienti per accantonare l'ipotesi della conversione di SI-Designer in una Applet.

4.3.2 Reingegnerizzazione dell'applicazione

I vantaggi di una progettazione a tre livelli sono notevoli:

- alta scalabilità del sitema, agevolata dalla possibilità di ripartire il carico tra più server;
- buone prestazioni con qualsiasi tipo di client, essendo gran parte dell'elaborazione delegata al server;
- indipendenza dalla piattaforma client, in quanto il lato dell'interfaccia utente può essere sviluppato in più versioni, supportando oltre all'HTML anche WML per i terminali WAP ed altre tecnologie;
- semplicità nell'aggiornamento, essendo necessario aggiornare solo il server.

La tecnologia utilizzata per lo sviluppo della nuova applicazione è Java ed in particolare è stato utilizzato CORBA per la comunicazione tra l'applicativo ed i tools di supporto come Wordnet o gli ODB-Tools.

Nei prossimi paragrafi entreremo nel dettaglio dell'implementazione.

4.4 Architettura di SIWeb

Nell'architettura di SIWeb, visibile in figura 4.2, sono stati tenuti separati i tre strati. In particolare i tre livelli non devono necessariamente risiedere sulla stessa macchina.

Il lato di presentazione consiste esclusivamente in codice HTML (generato dinamicamente) e in codice javascript che svolge sul client delle funzioni elementari come la validazione dell'input dell'utente.

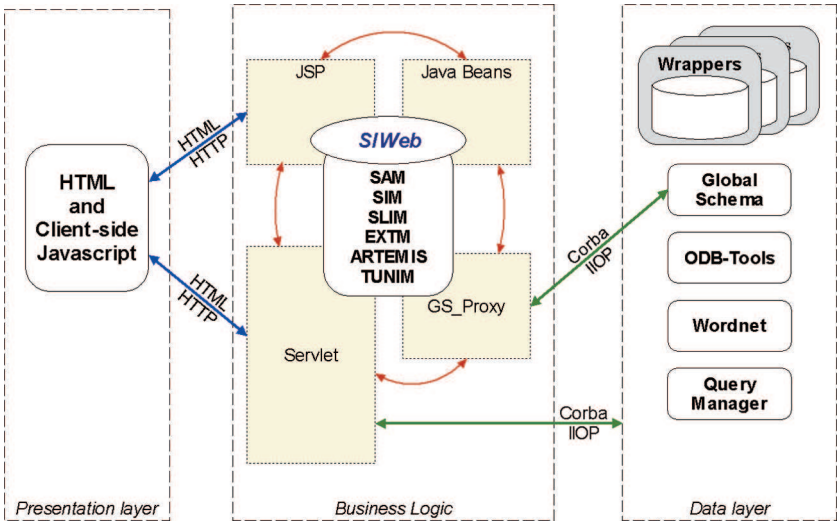

Figura 4.2: SIWeb: Architettura a tre livelli

La business logic è l'applicazione vera e propria che produce verso i client il codice HTML e comunica con MOMIS attraverso CORBA.

Lo strato dei dati non è costituito da un DBMS, ma dai tools che costituiscono il cuore di MOMIS, ovvero i wrapper, gli ODB-Tools, il Wordnet e il GlobalSchema.

Vediamo nello specifico il funzionamento dei tre strati.

4.4.1 Il livello di presentazione

Per capire in quale modo fosse più opportuno procedere per rendere SI-Designer fruibile da un web browser è necessario analizzarne il funzionamento dal punto di vista dell'interazione con l'utente.

Prendiamo una fase qualsiasi del processo di integrazione delle sorgenti, ad esempio quella di figura 4.3 ed analizziamone l'interfaccia.

Si può notare immediatamente quanti siano gli elementi di interattività di questa schermata. Sono presenti dei bottoni, un albero cliccabile, le fasi organizzate in pannelli cliccabili, ecc ...

Ottenere lo stesso tipo di interattività su una pagina web costringe a fare dei compromessi:

- data la struttura sequenziale del processo di integrazione delle sorgenti, si è deciso di sostituire i pannelli delle singole fasi in semplici link che collegano alle fasi precendente e successiva, oltre ad un link che torna sempre alla fase iniziale di acquisizione delle sorgenti;

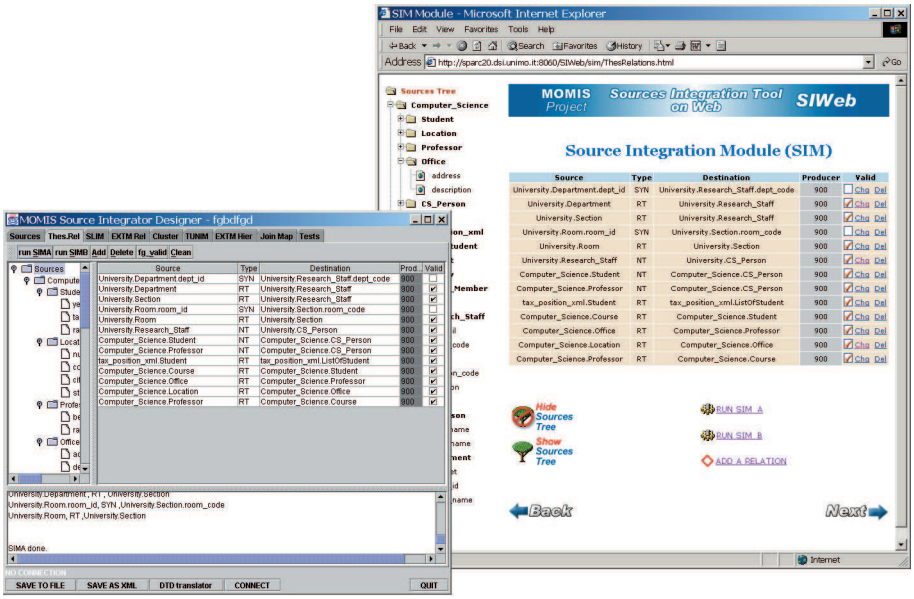

Figura 4.3: Le interfacce utente a confronto

- l'albero è stato creato utilizzando un Javascript che viene eseguito direttamente sulla macchina client. Il Javascript permette una buona interattività consentendo di cliccare sui nodi dell'albero per espanderne i rami. Non consente di rendere cliccabili le foglie e nemmeno funzioni di *drag and drop*. Essendo l'albero la rappresentazione delle sorgenti acquisite dal sistema, è intuitivo immaginare che il codice javascript che viene inviato al client, viene creato dinamicamente da un Javabean che risiede sul server. Negli ultimi paragrafi del capitolo entreremo nel dettaglio anche negli aspetti implementativi del codice;

- i bottoni sono dei semplici link o bottoni di un form. L'esecuzione dell'azione associata viene generalmente prodotta dalla chiamata ad una Servlet.

Gli snapshot dell'interfaccia grafica realizzata sono ampiamente illustrati nel capitolo 5.

4.4.2 La logica applicativa

In questo strato risiede l'applicazione vera e propria. La logica applicativa è stata implementata utilizzando congiuntamente Servlet, Java Server Pages e JavaBeans.

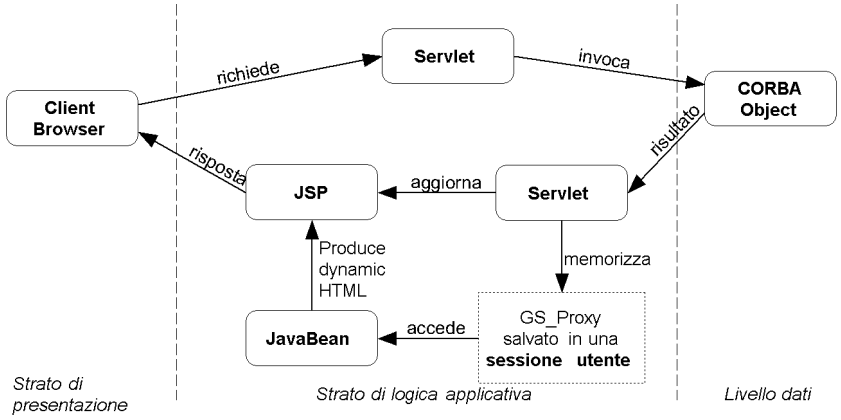

Figura 4.4: Flusso dei dati durante l'esecuzione di una operazione.

Il flusso dei dati di una tipica chiamata ad una funzione del processo di integrazione (illustrato in figura 4.4) può essere schematizzato nel seguente modo:

- l'utente clicca nel browser su un bottone (che può essere un link o un bottone *submit* di un form) per richiedere l'esecuzione di un comando;
- viene aperta una finestra separata da quella principale nella quale viene eseguita una Servlet;
- la servlet, esegue una funzione o richiama un oggetto CORBA a seconda dei casi;
- il risultato dell'operazione (sia essa una funzione interna o un oggetto CORBA remoto) viene salvato nel `GlobalSchemaProxy`. Il `GlobalSchemaProxy` è una copia locale dell'oggetto CORBA `GlobalSchema`. Esso è stato creato per evitare che ogni operazione effettuata dall'utente provochi una chiamata ad un oggetto remoto, penalizzando fortemente le prestazioni;
- l'oggetto `GlobalSchemaProxy` viene salvato nella *sessione utente* in modo da mantenere lo stato del sistema durante il passaggio tra le varie fasi. Ogni utente ha una propria sessione, questo permette l'utilizzazione dell'applicazione da più utenti contemporaneamente, in maniera assolutamente trasparente;
- la servlet, dopo aver salvato il `GlobalSchemaProxy`, restituisce all'utente l'esito dell'operazione e aggiorna la pagina principale del browser;

- la pagina principale si avvale di un JavaBean per leggere il contenuto del GlobalSchemaProxy e produrre dinamicamente l'HTML da restituire al client.

4.4.3 Il livelo dei dati

Il livello dei dati è composto da un insieme di oggetti CORBA che costituiscono il nucleo di MOMIS. Questi oggetti offrono una serie di servizi come gli ODB-Tools, Wordnet, i Wrappers e alcuni metodi di supporto all'integrazione delle sorgenti.

Tali oggetti sono stati utilizzati come "server" senza modificare nulla nei servizi offerti. CORBA è stato utilizzato nelle Servlet come un client per connettersi ai servizi disponibili.

Ciò permette di migliorare e ampliare MOMIS intervenendo direttamente sugli algoritmi dei singoli tools, provocando un automatico aggiornamento del sistema.

4.5 Implementazione del prototipo

Le difficoltà maggiori che sono state affrontate e superate possono essere riassunte nel seguente modo:

- mantenere la piena compatibilità con SI-Designer;
- mantenere lo stato della connessione, superando il problema della natura stateless di HTTP;
- minimizzare il traffico di rete sia verso il client che verso il livello dati costituito dai tools MOMIS;
- gestire la concorrenza delle connessioni, in quanto più utenti dovranno avere la possibilità di connettersi a SIWeb ed ognuno di essi dovrà poter operare indipendentemente dagli altri.

Nei paragrafi che seguono verranno illustrati i metodi utilizzati per realizzare un sistema che rispetti le specifiche sopra elencate.

4.5.1 Modulo SAM

Il modulo SAM (Source Acquisition Module) si occupa del collegamento con i wrapper per l'acquisizione delle sorgenti secondo la seguente procedura:

- Viene presentata all'utente la pagina `AddNewSource.jsp` nella quale compare un form nel quale vengono richiesti il nome del wrapper, il nome di dominio (o indirizzo IP) del server e la porta alla quale collegarsi.
- il form viene inviato alla servlet `AcquireSource` che provvede al collegamento via CORBA al server specificato. Il collegamento avviene nel seguente modo:

```
//
// Create and initialize the ORB
System.out.println(" - initializing the orb");
orb = ORB.init(connArray, null);
//
// Get the root naming context
System.out.println(" - get the root naming context");
org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
System.out.println(" - get the root naming context.....");
NamingContext ncRef = NamingContextHelper.narrow(objRef);
//
// Bind the object reference in naming service
System.out.println(" - resolving wrapper in naming service");
NameComponent nc = new NameComponent(wrapperName, "");
NameComponent path[] = {nc};
wrRef = WrapperHelper.narrow(ncRef.resolve(path));
```

  Il funzionamento dell'invocazione di un oggetto CORBA remoto è stato descritto ampiamente nel capitolo 3. In questo frammento di codice è possibile vedere come venga effettuata l'inizializzazione dell'ORB e il binding dell'object reference tramite il naming service.

  Al termine del collegamento con il wrapper, la servlet reindirizza il browser sulla pagina che visualizza l'elenco delle sorgenti acquisite. Il reindirizzamento viene eseguito dalla servlet utilizzando il seguente metodo:

```
response.sendRedirect("../sam/sources.html");
```

- la pagina che visualizza l'elenco delle sorgenti acquisite è composta da due frame. Il frame nella parte sinistra visualizza la struttura ad albero degli schemi acquisiti. Il seguente frammento di codice mostra come il JavaBean generi dinamicamente il Javascript necessario alla visualizzazione dell'albero nel client:

```
//
// Generating ALL NODES getting Interfaces from schema.
//
Map nodeMap = new HashMap();
//
_interfaceArray = src.getInterfaces();
for (int j=0;j<_interfaceArray.length;j++) {
    Interface infc = (Interface)_interfaceArray[j];
    System.out.println("  -         ITERF [" + infc.getName() + "]");
    _treeOut = _treeOut + "level_" + i + "_" + j
                    + " = insFld(level_" + i + ", gFld(\"<b>"
                    + infc.getName() + "</b>\", \"\")) \n ";

    HashSet attributeSet = new HashSet();
    _intBodies = infc.getIntBodies();
    for (int k = 0; k < _intBodies.length; k++) {
        //
        // Vector of attribute handling
        java.lang.Object[] attributes =
                    ((IntBody) _intBodies[k]).getAttributes();
        for (int l = 0; l < attributes.length; l++){
            //
            // single attribute handling
            System.out.println("  -              ATTRIBUTE ["
                        + ((Attribute) attributes[l]).getName() + "]");
            _treeOut = _treeOut + "insDoc(level_" + i + "_" + j
                     + ", gLnk(0, \"" + ((Attribute) attributes[l]).getName()
                     + "\", \"\")) \n ";

        }
    }
}
```

Questo frame viene visualizzato in tutte le fasi del processo di integrazione delle sorgenti, per comodità del progettista che può in ogni momento visualizzare lo schema delle sorgenti.

Il frame sulla destra visualizza semplicemente l'elenco dei wrappers ai quali si è collegati. Nella stessa pagina è contenuto il collegamento alla fase successiva, ovvero il modulo SIM.

4.5.2 Modulo SIM

La pagina JSP che visualizza il modulo SIM contiene due link per l'esecuzione degli algoritmi SIM_Ae SIM_B. La servlet viene invocata tramite il metodo HTTP GET, ovvero l'invocazione della seguente URL:

```
http://sparc20.ing.unimo.it:8060/SIWeb/servlet/runSIM?action=simA
```

Analizzandone la struttura si può vedere che la servlet invocata si chiama `runSIM` e risiede sul web server installato su `sparc20.ing.unimo.it` in ascolto sulla porta 8060. La URL contiene anche il parametro `action` impostato sul valore `simA`. La servlet `runSIM` preleva il valore del parametro `action` ed esegue il metodo appropriato. Il seguente codice illustra quanto descritto:

```
String action = request.getParameter("action");

if (action.equalsIgnoreCase("simA")) {
    _output = runSIM_A(_gsProxy);
    _tableThesRel.reloadStatus(_gsProxy.getLocalSchemata());
    session.setAttribute("tableThesRel", _tableThesRel);
} else {
    if (action.equalsIgnoreCase("simB")) {
        _output = runSIM_B(_gsProxy);
        _tableThesRel.reloadStatus(_gsProxy.getLocalSchemata());
        session.setAttribute("tableThesRel", _tableThesRel);
    } else {
        response.sendRedirect("../error.jsp");
        return;
    }
}
```

Si può notare inoltre come oltre ad essere invocato il metodo `runSIM_A` venga salvato l'output che viene mostrato all'utente, l'aggiornamento della tabella che viene visualizzata nella pagina JSP e il salvataggio nella sessione utente della tabella stessa. Il metodo runSIM_A esegue una chiamata ad un oggetto remoto CORBA in una maniera simile al precedente modulo, quindi ometteremo di illustrarne il codice.

4.5.3 Modulo EXTM

Il modulo EXTM permette di inserire nuove relazioni estensionali. Le classi che possono essere inserite nella relazione sono inserite in un menù a tendina che viene generato dinamicamente tramite un JavaBean. L'invocazione di un JavaBean permette di evitare l'iserimento di codice Java direttamente nella pagina JSP, rendendo il codice più pulito e leggibile. L'invocazione del JavaBean avviene tramite l'inserimento di un tag "XML style":

```
<jsp:useBean id="comboRules"
             scope="session"
             class="momisbeans.ExtmComboRulesBean" />
...
<jsp:getProperty name="comboRules"
                 property="showComboInterfListFirst" />
```

Il frammento di codice del JavaBean che genera le classi da visualizzare è il seguente:

```
//
// Generating ALL Interfaces from each source.
//
_interfaceArray = src.getInterfaces();
for (int j=0;j<_interfaceArray.length;j++) {
    Interface infc = (Interface)_interfaceArray[j];
    _left = _left + "<option value=\""
            + infc.getDottedName()
            + "\">" + infc.getDottedName()
            + "</option> \n";
    _right = _right + "<option value=\""
            + infc.getDottedName()
            + "\">" + infc.getDottedName()
            + "</option> \n";
}
```

L'inserimento della relazione avviene sempre invocando un metodo presente in una servlet, con gli stessi schemi seguiti nei precedenti moduli.

4.5.4 Modulo Artemis

L'ultimo modulo, Artemis ha la necessità di visualizzare l'albero degli schemi delle sorgenti in una modalità cliccabile. E'stato raggiunto lo scopo aggiungendo un bottone radio ai nodi e alle foglie dell'albero. Anche in questo caso l'albero viene visualizzato dinamicamente con l'ausilio di un JavaBean.

L'algoritmo Artemis viene invece eseguito invocando un metodo di una servlet. Non viene eseguito tramite un oggetto remoto CORBA perché ci sarebbe stato un degrado delle prestazioni notevole, in quanto l'algoritmo comunica con il `GlobalSchema`, generando un notevole traffico di rete. Agendo invece come metodo locale, e comunicando con il `GlobalSchemaProxy` si risolve il problema diminuendo notevolmente i tempi di risposta.

Nel capitolo successivo verrà confrontato SIWeb con il suo predecessore SI-Designer, mostrando come la nuova applicazione produca i medesimi risultati. Dal punto di vista funzionale e dell'interfaccia grafica le due applicazioni hanno moltissimi punti in comune anche se, come si è potuto constatare in questo capitolo, l'architettura interna è completamente diversa.

Capitolo 5

Analisi dei risultati ottenuti

Abbiamo descritto dettagliatamente nel capitolo 4 tutte le fasi della progettazione di SIWeb e la sua implementazione. Concludiamo ora l'analisi di SIWeb con l'esame di una sessione di lavoro reale. Da notare come l'interfaccia utente di SIWeb sia facilmente utilizzabile da chi già conosce il suo predecessore SI-Designer.

L'interazione di SIWeb e' stata studiata tenendo sempre presente l'importanza dell'usabilità [31] del sistema.

SIWeb organizza il processo di integrazione delle sorgenti in quattro fasi:

- Acquisizione delle sorgenti
- Integrazione
- Modulo per le relazioni estensionali
- Generazione dei cluster

Essendo un processo essenzialmente sequenziale, è stato intuitivo organizzarlo come una sorta di *Wizard* dove ogni fase contiene un collegamento alla precedente e alla successiva. Da ogni punto, inoltre, si può tornare alla fase di acquisizione delle sorgenti per poterne sempre acquisire di nuove.

Vediamo nei paragrafi che seguono ognuna delle quattro fasi, mentre le figure evidenziano la similitudine tra l'applicazione SI-Designer e la sua nuova versione via web.

Risulta evidente che le due applicazioni producono esattamente gli stessi risultati.

5.1 Acquisizione delle sorgenti

La prima fase che si presenta all'utente è quella dell'acquisizione delle sorgenti.

SI-Designer presenta un pannello nel quale bisogna inserire:

- Nome della sorgente
- Indirizzo del server sul quale è in esecuzione il wrapper
- Porta a cui collegarsi sul server

Come evidenziato nella figura 5.1 all'utente viene quindi presentato l'elenco delle sorgenti collegate, corredate da indirizzo e porta del relativo server. Viene inoltre esposta sulla sinistra una rappresentazione ad albero degli schemi delle sorgenti stesse.

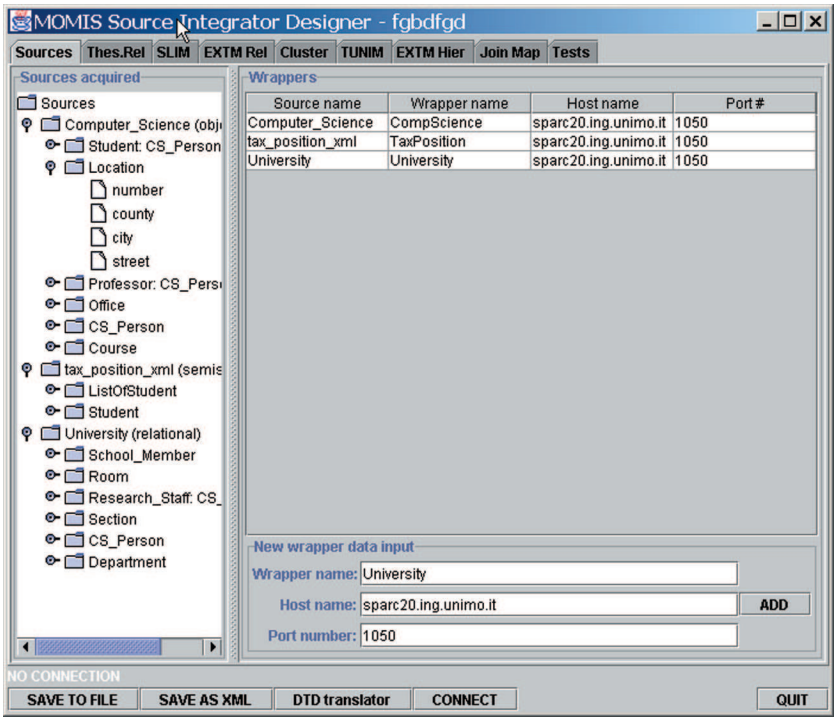

Figura 5.1: SI-Designer: Modulo SAM

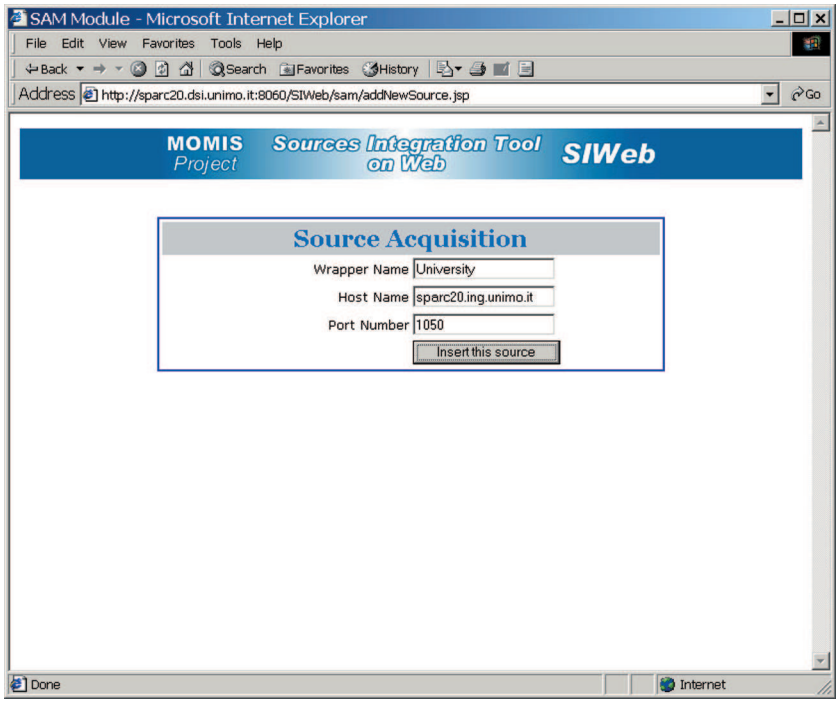

Figura 5.2: SIWeb: Acquisizione sorgenti

SIWeb divide questa fase in due, proponendo inizialmente un modulo per l'inserimento dei dati necessari al collegamento con la sorgente (Figura 5.2) e una seconda dove si vedono quelle già collegate (Figura 5.3) corredata dalla rappresentazione ad albero generata dinamicamente con un Javascript, come descritto nel Capitolo 4.

5.2 Modulo di integrazione delle sorgenti

5.2.1 Il modulo SIM_A

SI-Designer viene mostrato nella figura 5.4 con il risultato dell'esecuzione di SIM_A. E' possibile vedere nella parte sinistra la rappresentazione delle sorgenti, mentre nella parte bassa viene restituito l'output dell'algoritmo.

Avendo l'HTML degli strumenti per l'interazione più limitati, si è reso necessario creare i bottoni *Change* e *Delete* in ogni riga della tabella, in quanto sarebbe stato difficoltoso e poco intuitivo selezionare una riga e premere un bottone separato, non essendo cliccabili le tabelle HTML (Figura 5.5).

SIWeb presenta una schermata molto simile, nella quale l'unica differenza

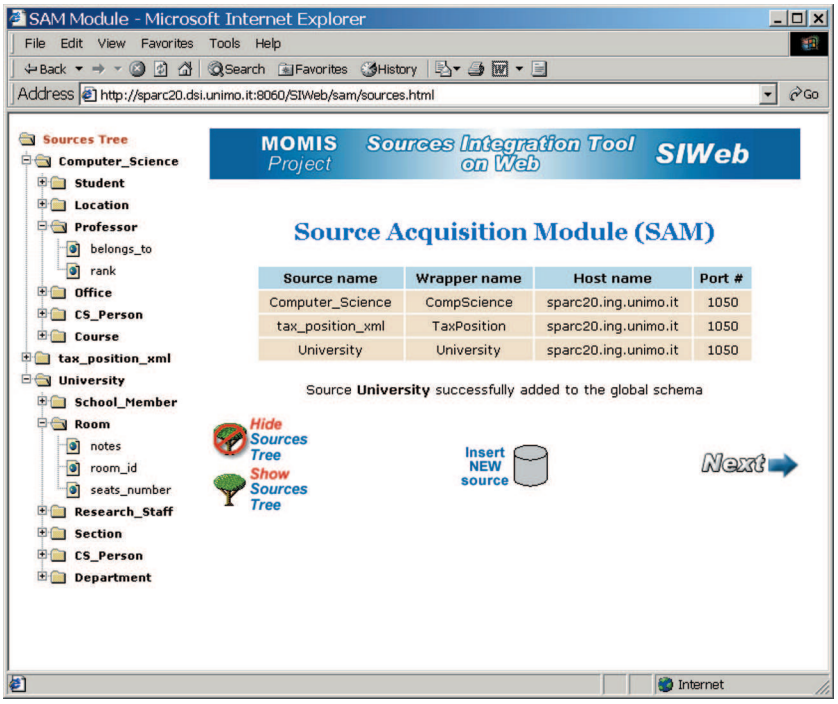

Figura 5.3: SIWeb: Elenco sorgenti acquisite

consiste nell'output dell'algoritmo che viene visualizzato in una finestra separata (Figura 5.6).

L'algoritmo viene invocato via CORBA, quindi è esattamente lo stesso eseguito da SI-Designer.

5.2.2 Il modulo SIM_B

Per il modulo SIM_B valgono le stesse considerazioni fatte per il modulo SIM_A.

E' possibile confrontare anche in questo caso le similitudini tra SI-Designer e SIWeb nelle figure 5.7, 5.8 e 5.9

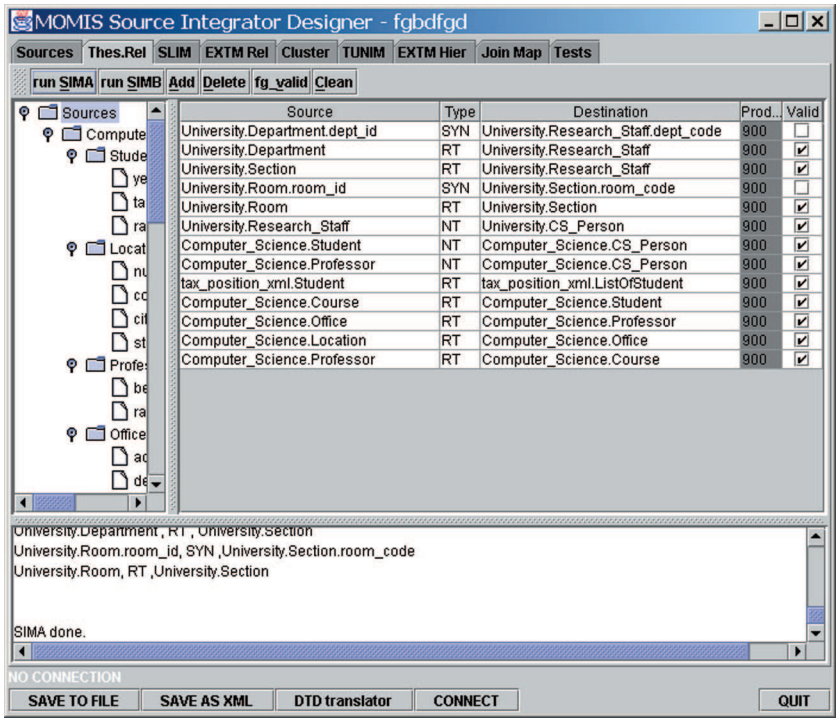

Figura 5.4: SI-Designer: Modulo SIM_A

5.3 Inserimento di una relazione

Da evidenziare in questa fase (Figure 5.10, 5.11 e 5.12) come SIWeb permetta l'inserimento delle relazioni cliccando semplicemente su una voce dell'elenco invece di doverne digitare il nome, come accade per SI-Designer.

5.4 Modulo per le relazioni estensionali

L'interfaccia grafica relativa al modulo per l'inserimento delle relazioni estensionali è esattamente uguale in entrambe le applicazioni SI-Designer e SIWeb, come è possibile vedere nelle figure 5.13 e 5.14

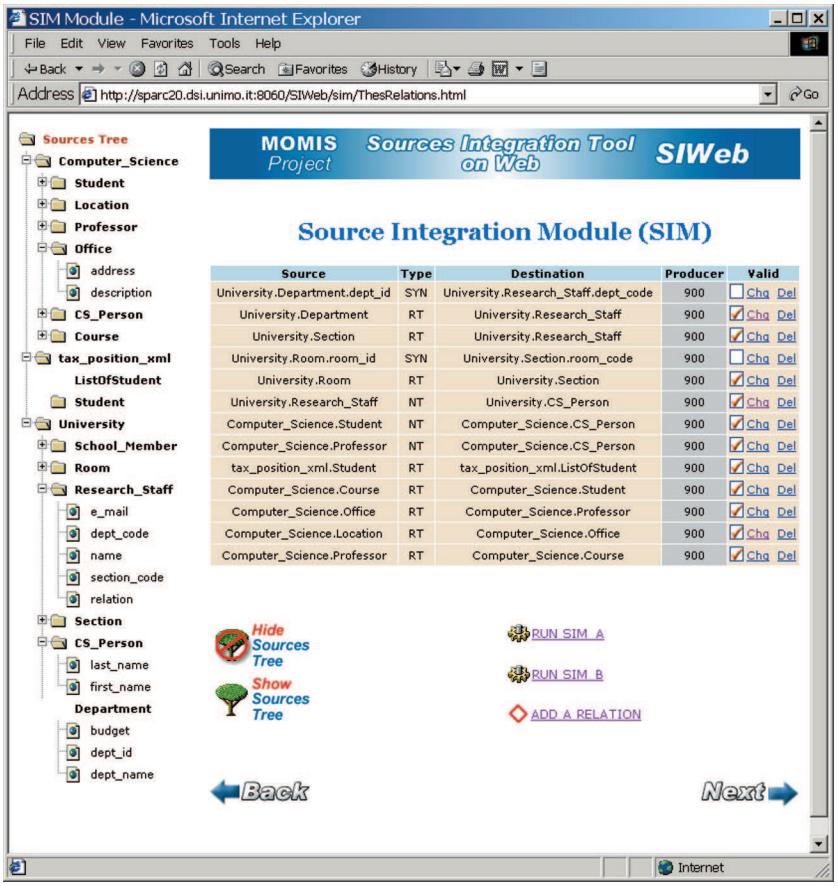

Figura 5.5: SIWeb: Risultato dell'esecuzione di SIM_A

5.5 La generazione dei cluster

5.5.1 Il modulo Artemis

Il modulo Artemis si presenta in SI-Designer (Figura 5.15) con l'albero degli schemi sulla sinistra. Tramite i comandi MAP e UNMAP si possono creare e personalizzare i cluster, mentre con CREATE GLOBAL CLASSES viene lanciato l'algoritmo Artemis per la generazione automatica dei clusters.

SIWeb (Figura 5.16) pur mantenendo le stesse funzionalità, propone l'albero delle sorgenti come un elenco indentato, cliccabile tramite dei "*bottoni radio*". Sono accessibili via web tutte le funzioni esattamente come in SI-Designer.

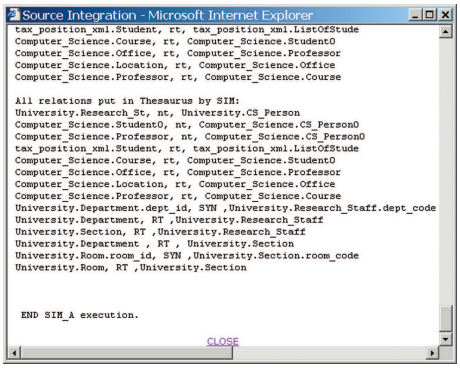

Figura 5.6: SIWeb: Output del metodo SIM_A

5.5.2 La configurazione di Artemis

Anche la schermata relativa alla configurazione di Artemis, con riferimento alle figure 5.17 e 5.18, si presenta assolutamente identica in entrambe le applicazioni.

5.5.3 I cluster

L'ultima fase della procedura corrisponde al risultato dell'applicazione del tool Artemis ed è visibile nelle figure 5.19 e 5.20. L'unico elemento che distingue le due aplicazioni è la posizione dei pulsanti *rename* e *delete* che in SIWeb sono presenti accanto al nome di ogni cluster.

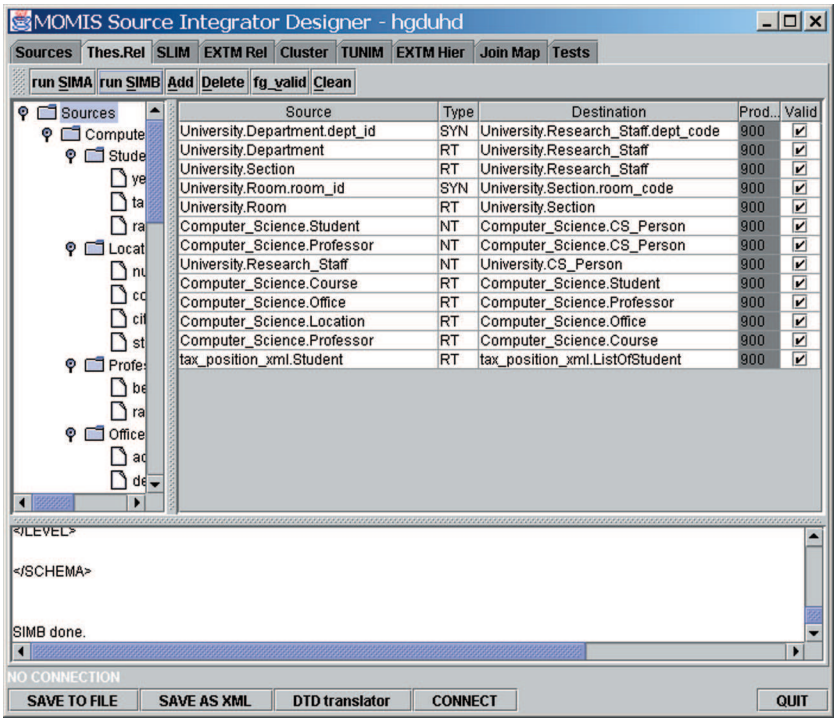

Figura 5.7: SI-Designer: Modulo SIM_B

5.6 Tools utilizzati

Per la realizzazione di SIWeb è stato utilizzato

- **Forte for Java Internet Edition 2.0** con il quale sono state realizzate le servlet e i javabean.
- **Macromedia Dreamweaver Ultradev 4** per la realizzazione delle pagine JSP
- **Macromedia Fireworks 4** facente parte del pacchetto Ultradev, è stato utilizzato per la realizzazione della grafica (icone, bottoni, ecc...)
- **Tomcat 3.2** è stato usato come web server e come application server per l'esecuzione delle servlet e delle pagine JSP

L'applicazione è stata eseguita sia su macchine Sun con sistema operativo Solaris, sia in ambiante Microsoft Windows 2000. In entrambe le situazioni l'applicazione ha funzionato senza problemi e senza alcuna modifica al codice.

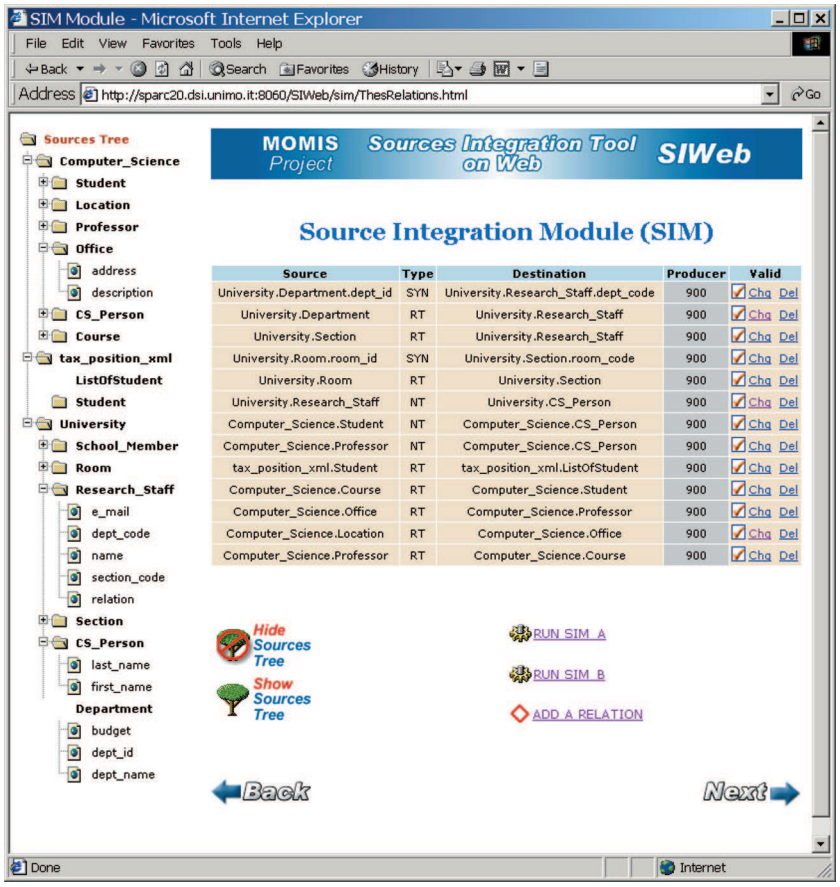

Figura 5.8: SIWeb: Risultato dell'esecuzione di SIM_B

Per il trasferimento dell'applicazione da un ambiente all'altro è stato necessario semplicemente copiare il file SIWeb.war nell'apposita directory del server Tomcat della macchina ospite. Questo è stato possibile grazie alla notevole portabilità di Java, come del resto già descritto nel capitolo 3.

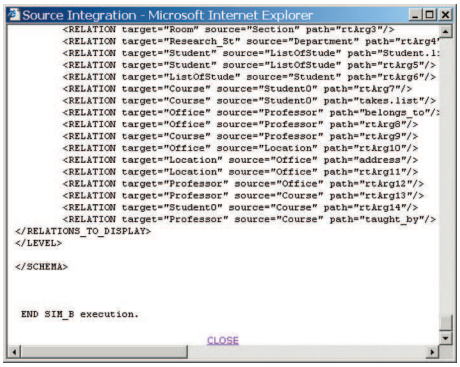

Figura 5.9: SIWeb: Output del metodo SIM_B

Questo documento è stato scritto con il linguaggio Latex, in ambiente Windows con i seguenti tools:

- **Miktex 2.1** è stata la versione del Latex utilizzata per la compilazione del documento
- **Crimson Editor** è un piccolo programma **freeware** molto versatile utilizzato per redigere il presente testo.
- **SnagIt 6.0** è un tool **shareware** per creare gli snapshot dello schermo, in particolare cattura direttamente le finestre dei programmi in esecuzione. La grande utilità di questo tool sta nello scroll automatico del testo delle pagine web, per generare delle immagini intere di un sito, anche se non entra interamente nello schermo.
- **SmartDraw 5.0** è stato utilizzato per creare alcuni grafici ed è utile per generare organigrammi, schemi E/R e diagrammi in genere. Contiene anche un'ampia collezione di simboli.
- **Adobe Photoshop** è stato utilizzato per la conversione delle immagini in formato EPS (encapsuled postscript).

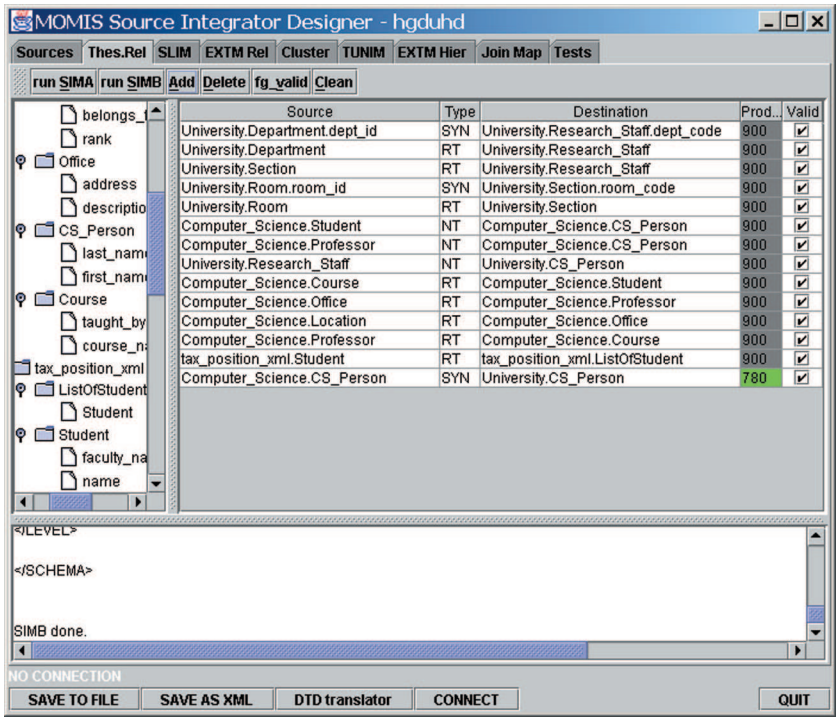

Figura 5.10: SI-Designer: Inserimento di una relazione

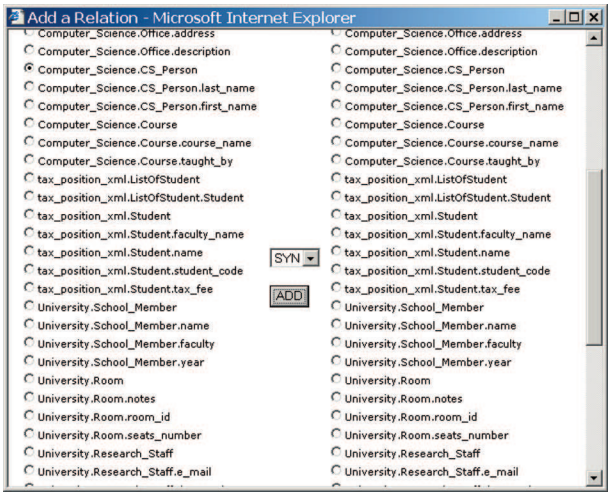

Figura 5.11: SIWeb: Inserimento di una relazione

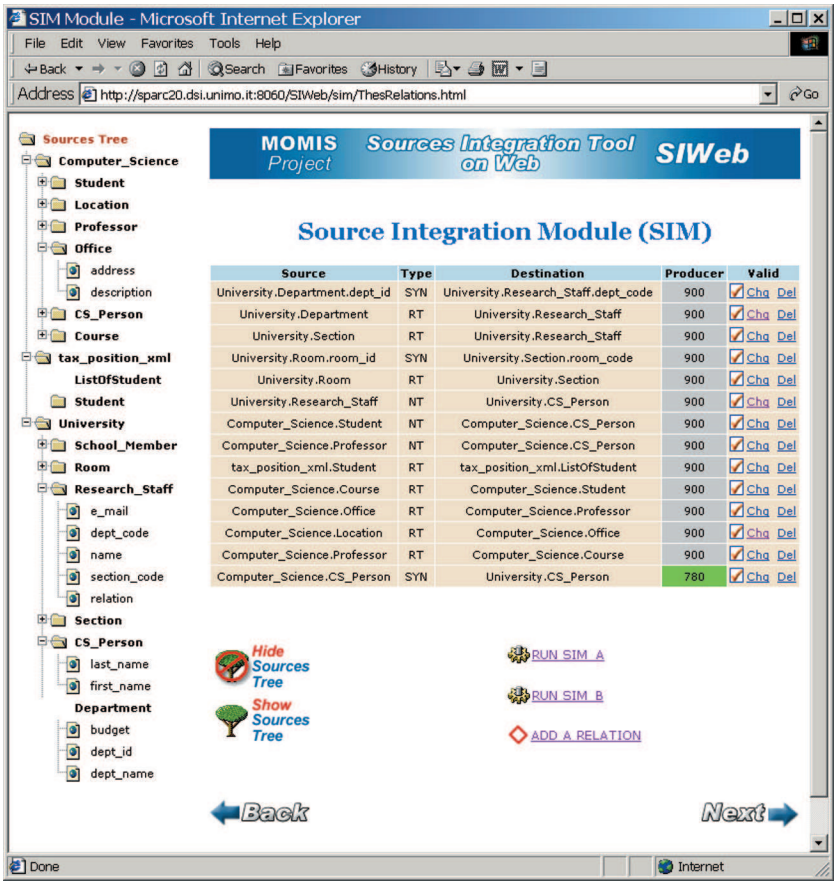

Figura 5.12: SIWeb: Risultato dell’inserimento di una relazione

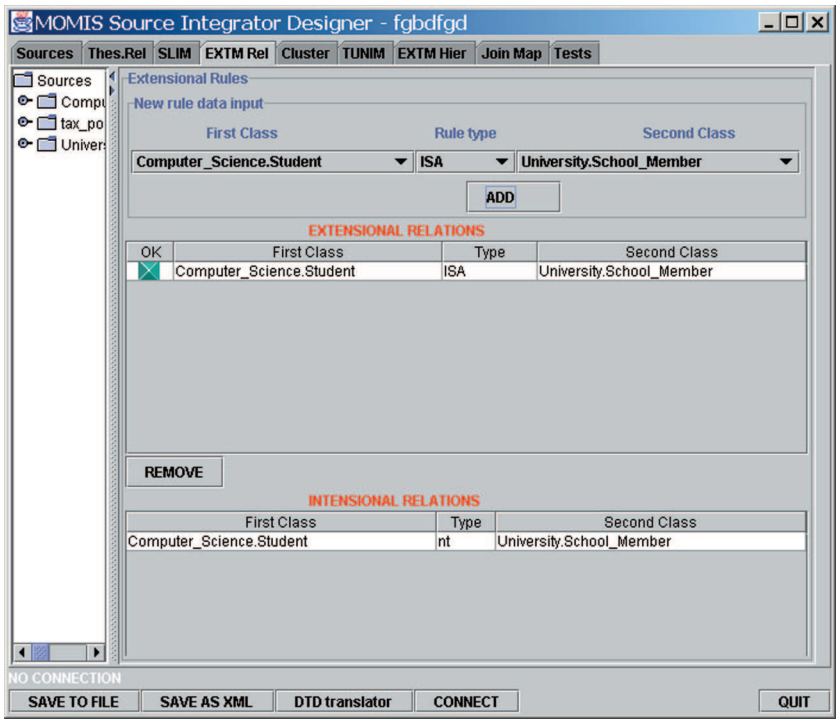

Figura 5.13: SI-Designer: Modulo EXTM

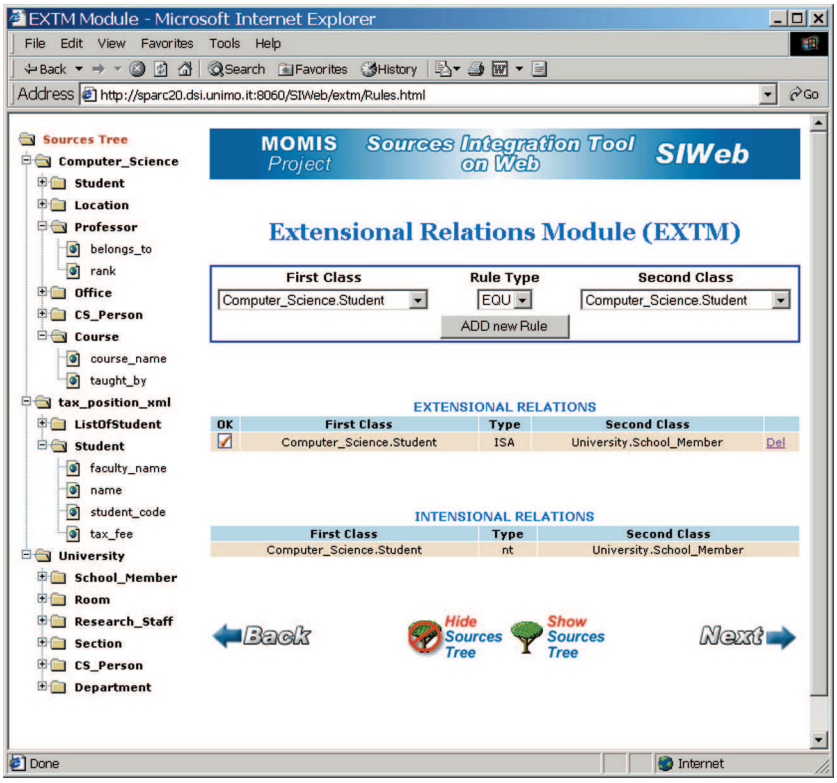

Figura 5.14: SIWeb: Modulo EXTM

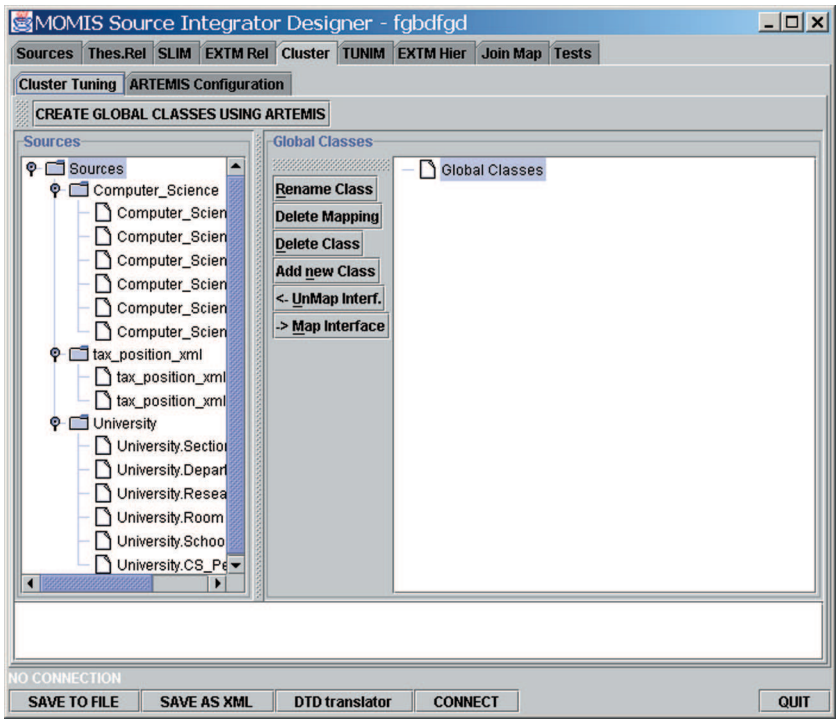

Figura 5.15: SI-Designer: Modulo Artemis

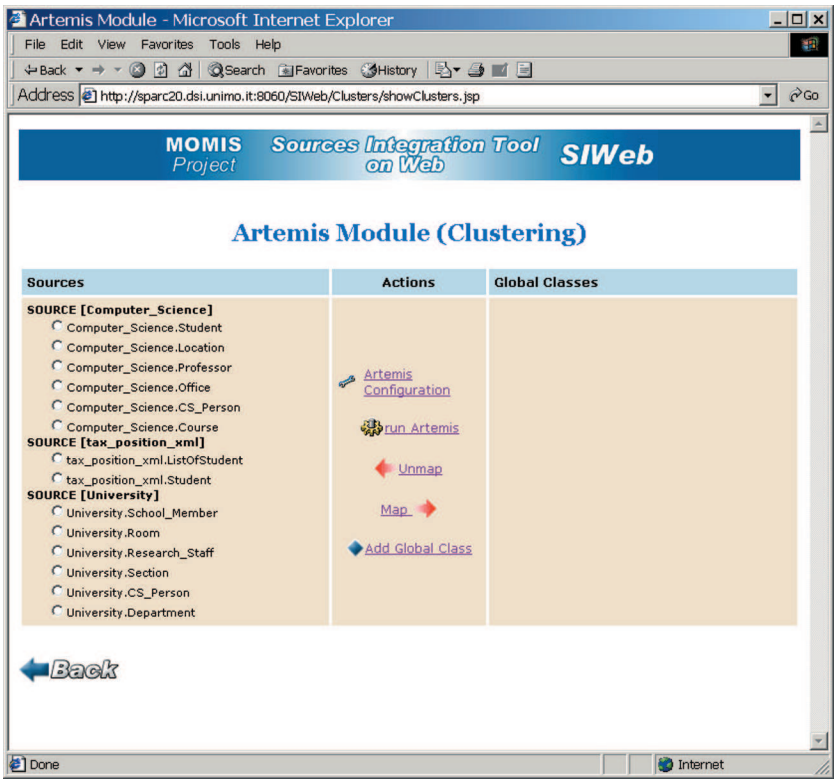

Figura 5.16: SIWeb: Modulo Artemis

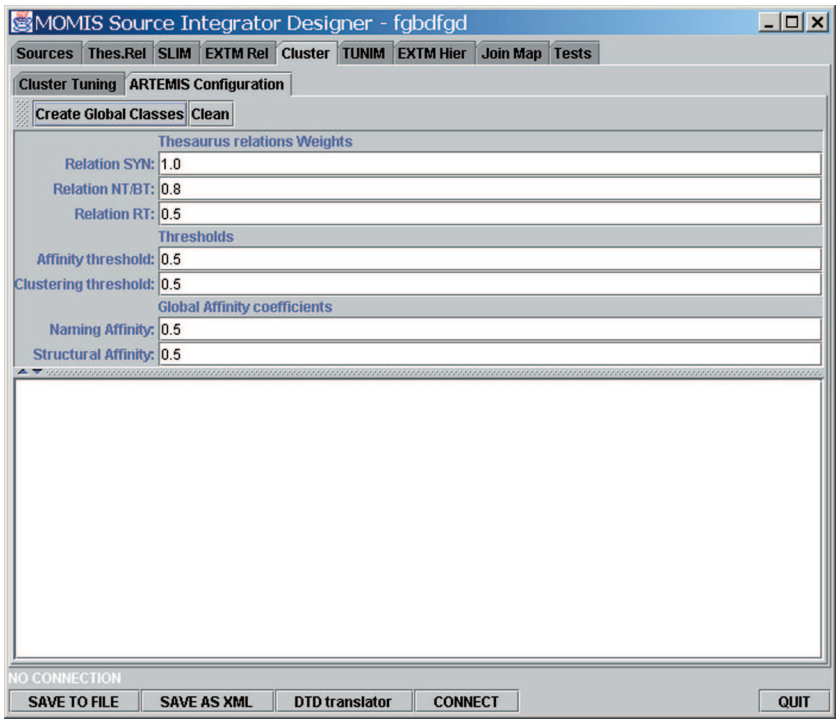

Figura 5.17: SI-Designer: Configurazione Artemis

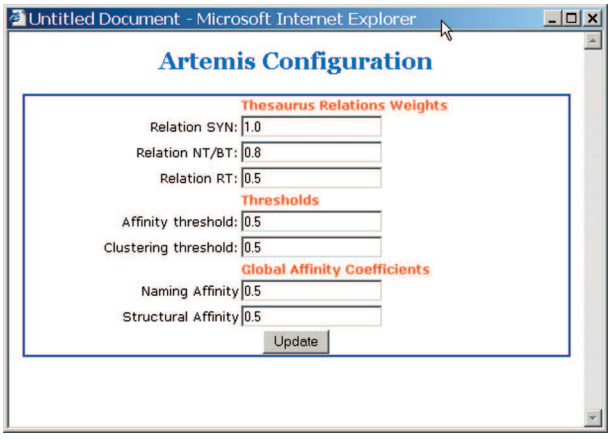

Figura 5.18: SIWeb: Configurazione Artemis

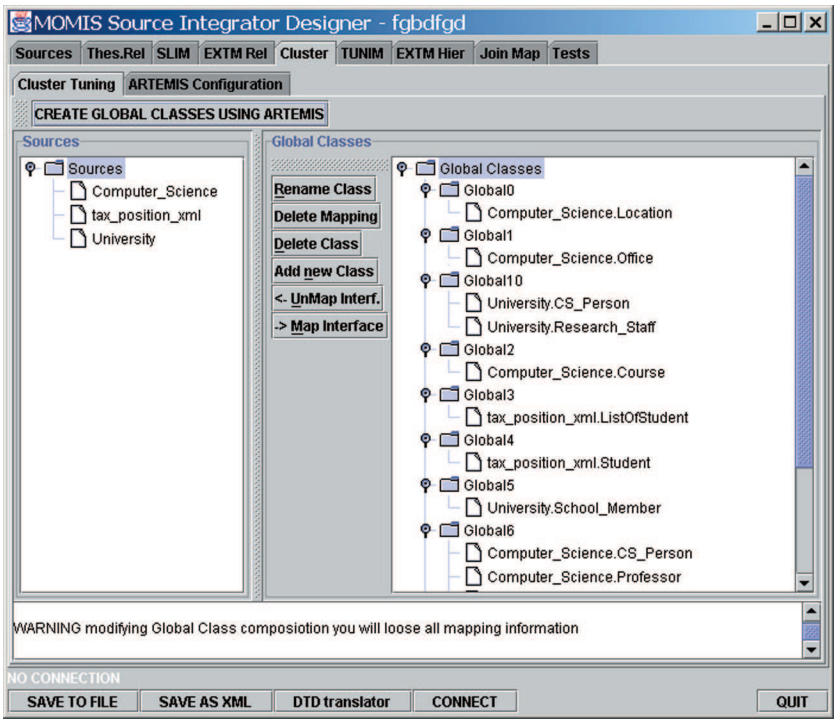

Figura 5.19: SI-Designer: Clusters generati da Artemis

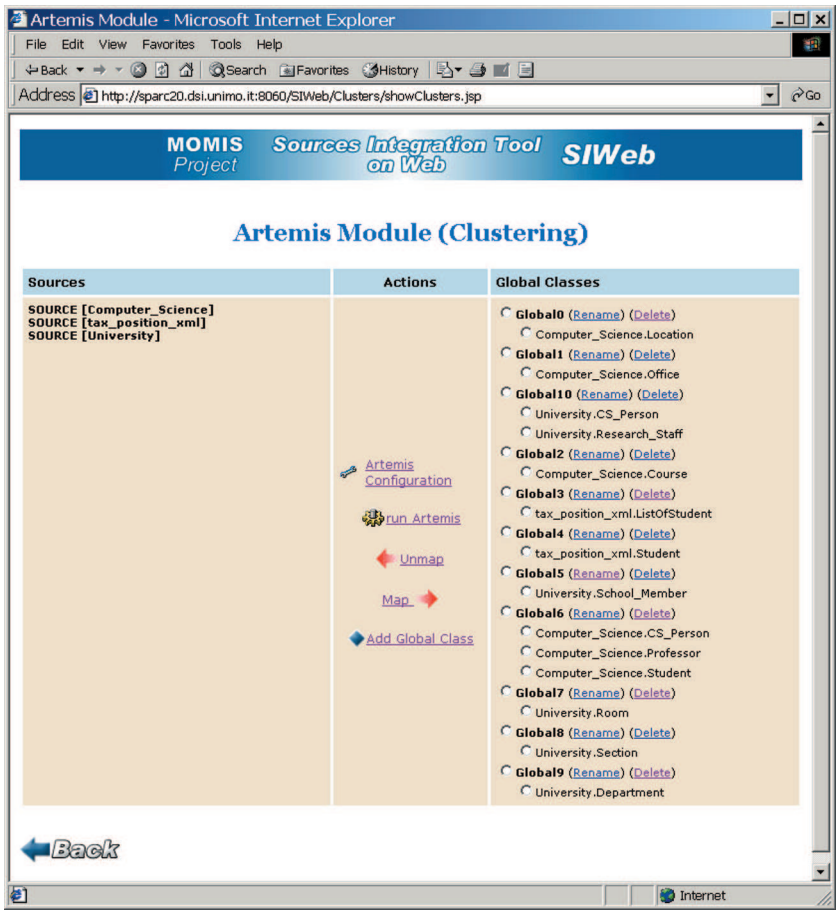

Figura 5.20: SIWeb: Clusters generati da Artemis

Conclusioni e Sviluppi Futuri

In questa tesi è stato descritto SIWeb, un insieme di applicazioni fruibili tramite un web browser, che permette di interfacciare l'utente con il sistema MOMIS.

MOMIS è uno strumento che, seguendo le linee guida tracciate nell'ambito I^3, permette la reale integrazione di sorgenti distribuite, eterogenee sia strutturate sia semistrutturate. Nei capitoli 1 e 2 è stata ampiamente descritta l'architettura del sistema.

L'obiettivo principale di questa tesi è stato quello di effettuare una conversione di una preesistente applicazione Java, SI-Designer, in modo da renderla fruibile anche sul web. In tal modo ho avuto la possibilità di approfondire la conoscenza del linguaggio Java, focalizzando in particolar modo gli argomenti relativi alle tecnologie web, che sono state descritte nel capitolo 3.

Si è reso necessario effettuare dapprima una analisi dell'interfaccia grafica, per evidenziare l'interattività con l'utente e come essa potesse essere riprodotta con un browser. Inizialmente è stata valutata la possibilità di convertire SI-Designer in una java applet, ipotesi subito abbandonata per diversi motivi, primo fra tutti la pesantezza dell'applicazione stessa. Si è giunti quindi alla decisione di riprogettare l'intero software in maniera modulare e soprattutto a tre livelli, in modo da separare la logica applicativa dalla presentazione all'utente. Tutto ciò è descritto nel capitolo 4.

Il risultato ottenuto è quello di una piena compatibilità di SIWeb con la vecchia applicazione. Svolgendo una sessione di lavoro, illustrata dettagliatamente nel capitolo 5, si può notare la familiarità degli stumenti messi a disposizione sul web con quelli utilizzabili nell'applicazione Java, oltre alla assoluta corrispondenza dei risultati.

Pur essendo ad uno stadio già utilizzabile dagli utenti, SIWeb è da ritenersi un prototipo non ancora completo, che può essere ulteriormente ampliato e migliorato. Il primo passo è quello dell'inserimento di due fasi importanti del processo di integrazione, che sono state attualmente tralasciate: 1) l'interazione con Wordnet e 2) il tuning della mapping table.

Vi sono ulteriori miglioramenti che possono essere fatti e che sono sicuramente importanti per la fruibilità del prodotto finale:

- non è possibile cancellare una sorgente acquisita così come non è possibile aggiornare lo schema di una sorgente se essa è stata modificata dopo l'acquisizione. Queste due funzioni andranno certamente implementate nelle versioni successive.

- potrebbe essere utile creare dei profili utente. Ogni utente potrebbe entrare quindi con la propria UserID e password, accedendo al sistema con una configurazione personalizzata dando la possibilità eventualmente di salvare le sessioni di lavoro.

- i wrapper devono essere già installati, configurati per dialogare con la sorgente dati e devono essere ovviamente in esecuzione per essere utilizzati con SIWeb. Un passo avanti potrebbe essere la realizzazione di un tool che permetta l'automatizzazione dell'installazione di un wrapper. In tal modo si potrebbe dare la possibilità all'utente di scaricare direttamente dal sito un nuovo wrapper, installarlo e collegare quindi la propria sorgente al sistema MOMIS.

- il sistema è predisposto per essere visualizzato con un browser. Un possibile ampliamento potrebbe essere la modifica dello strato di logica applicativa per rendere SIWeb compatibile con altri tipi di client, ad esempio cellulari WAP, client java, palmari ed eventuali future applicazioni.

Appendice A

Glossario I^3

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

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

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

A.1 Architettura

- agente = strumento che realizza un servizio, sia per il suo proprietario, sia per un cliente del suo proprietario.
- applicazione = configurazione persistente o transitoria dei componenti, rivolta a risolvere un problema del cliente, e che può coprire diversi domini.
- Architettura = insieme di componenti.
- architettura di riferimento = linea guida ed insieme di regole da seguire per l'architettura.

- cliente (customer) = proprietario dell'applicazione che gestisce le interrogazioni, o utente finale, che usufruisce dei servizi.
- collante (glue) = software o regole che servono per per collegare i componenti o per interoperare attraverso i domini.
- componente = uno dei blocchi sui quali si basa una applicazione o una configurazione. Incorpora strumenti e conoscenza specifica del dominio.
- configurazione = istanza particolare di una architettura per una applicazione o un cliente.
- conoscenza = metadata, relazione tra termini, paradigmi…, utili per trasformare i dati in informazioni.
- contenuto = risultato informativo ricavato da una sorgente.
- dato = registrazione di un fatto.
- dominio = area, argomento, caratterizzato da una semantica interna, per esempio la finanza, o i componenti elettronici…
- facilitatore = componente che fornisce i servizi di coordinamento, come pure l'instradamento delle interrogazioni del cliente.
- interoperare = combinare sorgenti e domini multipli.
- informazione = dato utile ad un cliente.
- informazione azionabile = informazione che forza il cliente ad iniziare un evento.
- 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.
- 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…

- regole limitative (constraint rules) = definizione di regole per l'assegnamento di componenti o di protocolli a determinati strati.
- risorsa = base di dati accessibile, server ad oggetti, base di conoscenze...
- servizio = funzione fornita da uno strumento in un componente e diretta ad un cliente, direttamente od indirettamente.
- strato = grossolana categorizzazione dei componenti e degli strumenti in una configurazione. L'architettura I^3 distingue tre strati, ognuno dei quali fornisce una diversa categoria di servizi:
  1. Servizi di Coordinamento = coprono le fasi di scoperta delle risorse, distribuzione delle risorse, invocazione, scheduling...
  2. Servizi di Mediazione = coprono la fase di query processing e di trattamento dei risultati, nonché il filtraggio dei dati, la generazione di nuove informazioni, etc.
  3. Servizi di Wrapping = servono per l'utilizzo dei wrappers e degli altri strumenti simili utilizzati per adattarsi a standards di accesso ai dati e alle convenzioni adoperate per la mediazione e per il coordinamento.
- strumento (tool) = programma software che realizza un servizio, tipicamente indipendentemente dal dominio.
- testo = dato, informazione o conoscenza in un formato relativamente non strutturato, basato sui caratteri.
- wrapper = strumento utilizzato per accedere alle risorse conosciute, e per tradurre i suoi oggetti.

A.2 Servizi

- accertamento dell'impatto = servizio che riporta quali risorse saranno interessate dalle interrogazioni o dagli aggiornamenti.
- accesso = collegamento agli oggetti nelle risorse per realizzare interrogazioni, analisi o aggiornamenti.
- accoppiamento (matchmaking) = servizio che accoppia i sottoscrittori di un servizio ai fornitori.
- accoppiamento spaziale = servizio di mediazione per riconoscere e risolvere differenze nelle unità di misura spaziali utilizzate dalle risorse.

- accoppiamento temporale = servizio di mediazione per riconoscere e risolvere differenze nelle unità di misura temporali utilizzate dalle risorse.
- aggiornamento = trasmissione dei cambiamenti dei dati alle risorse.
- amministrazione del modello = servizio di mediazione per permettere al cliente ed al proprietario del mediatore di aggiornare il modello.
- analisi del contenuto = trattamento degli oggetti testuali per creare informazioni.
- astrazione = servizio per ridurre le dimensioni del contenuto portandolo ad un livello superiore.
- attivo (activeness) = abilità di un impulso di reagire ad un evento.
- browsing = servizio per permettere al cliente di spostarsi attraverso le risorse.
- caching = mantenere le informazioni memorizzate in un livello intermedio per migliorare le prestazioni.
- classificazione (ranking) = servizio di mediazione per assegnare dei valori agli oggetti ritrovati.
- contenuto = risultato prodotto da una risorsa in risposta ad interrogazioni.
- controllo (monitoring) = osservazione delle risorse o dei dati virtuali e creazione di impulsi da azionare ogniqualvolta avvenga un cambiamento di stato.
- controllo della concorrenza = assicurazione del sincronismo degli aggiornamenti delle risorse, tipicamente assegnato al sistema che amministra le transazioni.
- decomposizione dell'interrogazione (query decomposition) = determina le interrogazioni da spedire alle risorse o ai servizi disponibili.
- direttorio = servizio per localizzare e contattare le risorse disponibili, come le pagine gialle, pagine bianche...
- filtraggio = servizio di mediazione per aumentare la pertinenza delle informazioni ricevute in risposta ad interrogazioni.
- indicizzazione = creazione di una lista di oggetti (indice) per aumentare la velocità dei servizi di accesso.

- 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.
- integrazione = servizio di mediazione che combina i contenuti ricevuti da una molteplicità di risorse, spesso eteorgenee.
- intermediazione (brokering) = servizio di coordinamento per localizzare le risorse migliori.
- instanziazione del mediatore = popolamento di uno strumento indipendente dal dominio con conoscenze dipendenti da un dominio.
- ottimizzazione = processo di manipolazione o di riorganizzazione delle interrogazioni per ridurne il costo o il tempo di risposta.
- pubblicità (advertising) = presentazione del modello di una risorsa o del mediatore ad un componente o ad un cliente.
- ragionamento (reasoning) = metodologia usata da alcuni componenti o servizi per realizzare inferenze logiche.
- riformulazione dell'interrogazione (query reformulation) = programma per ottimizzare o rilassare le interrogazioni, tipicamente fa uso dello scheduling.
- rilassamento = servizio che fornisce un insieme di risposta maggiore rispetto a quello che l'interrogazione voleva selezionare.
- scheduling = servizio di coordinamento per determinare l'ordine di invocazione degli accessi e di altri servizi; fa spesso uso dei costi stimati.
- scoperta delle risorse = servizio che ricerca le risorse.
- servizi di descrizione = metaservizi che informano i clienti sui servizi, risorse...
- Servizio = funzionalità fornita da uno o più componenti, diretta ad un cliente.
- servizio di transazione = servizio che assicura la consistenza temporale dei contenuti, realizzato attraverso l'amministrazione delle transzioni.
- sottoscrizione = richiesta di un componente o di un cliente di essere informato su un evento.
- spiegazione = servizio di mediazione per presentare i modelli ai clienti.

- stimatore = servizo di basso livello che stima i costi previsti e le prestazioni basandosi su un modello, o su statistiche.
- strumento di configurazione = programma usato nel coordinamento per aiutare a selezionare ed organizzare i componenti in una istanza particolare di una configurazione architetturale.
- traduzione = trasformazione dei dati nella forma e nella sintassi richiesta dal ricevente.
- 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...

A.3 Risorse

- amministrazione della transazione = assicurare che la consistenza temporale del database non sia compromessa dagli aggiornamenti.
- antenato (ancestor) = oggetto di livello superiore, dal quale derivano attributi ereditabili.
- 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.
- cambiamento di stato = stato successivo ad una azione di aggiornamento, inserimento o cancellazione.
- costo = prezzo per fornire un servizio o un accesso ad un oggetto.
- database = risorsa che comprende un insieme di dati con uno schema descrittivo.
- database attivo = database in grado di reagire a determinati eventi.
- database deduttivo = database in grado di utilizzare regole logiche per trattare i dati.
- datawarehouse = deposito di dati integrati provenienti da una molteplicità di risorse.

- dato virtuale = dato rappresentato attraverso referenze e procedure.
- deposito di metadata = database che contiene metadata o metainformazioni.
- dizionario = lista dei termini, fa parte dell'ontologia.
- eterogeneità = incompatibilità trovate tra risorse e servizi sviluppati autonomamente, che vanno dalla piattaforma utilizzata, sistema operativo, modello dei dati, alla semantica, ontologia,...
- evento = ragione per il cambiamento di stato all'interno di un componente o di una risorsa.
- gerarchia = struttura di un modello che assegna ogni oggetto ad un livello, e definisce per ogni oggetto l'oggetto da cui deriva.
- impatto della transazione = riporta le risorse che sono state coinvolte in un aggiornamento.
- interoperabilità = capacità di interoperare.
- livello = categorizzazione concettuale , dove gli oggetti di un livello inferiore dipendono da un antenato di livello superiore.
- modello del database = descrizione formalizzata della risorsa database, che include lo schema.
- network = struttura di un modello che fa uso di relazioni relativamente libere tra oggetti.
- oggetto = istanza particolare appartenente ad una risorsa, al modello del cliente, o ad un certo strumento.
- oggetto root = oggetto da cui tutti gli altri derivano, all'interno di una gerarchia.
- 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.
- regola = affermazione logica, unità della conoscenza trattabile in modo automatico.

- 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.
- ristrutturare = dare una struttura diversa ai dati seguendo un modello differente dall'originale.
- 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.
- server di oggetti = fornisce dati oggetto.
- simulazione = risorsa in grado di fare proiezioni future sui dati e generare nuove informazioni, basata su un modello.
- sistema di amministrazione delle regole = software indipendente dal dominio che raccoglie, seleziona ed agisce sulle regole.
- stato = istanza o versione di una base di dati o informazioni.
- valore = contenuto metrico presente nel modello del cliente, come qualità, rilevanza, costo.
- vista = sottoinsieme di un database, sottoposto a limiti, e ristrutturato.
- 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.

A.4 Ontologia

- algebra dell'ontologia = insieme delle operazioni per definire relazioni tra ontologie.
- classe = definisce metaconoscenze come metodi, attributi, ereditarietà, per gli oggetti in essa istanziati.
- comparatore di ontologie = strumento per determinare relazioni tra ontologie, utilizzato per determinare le regole necessarie per la loro integrazione.

- concetto = definisce una astrazione o una aggregazione di oggetti per il cliente.
- consistenza temporale = è raggiunta se tutti i dati si riferiscono alla stessa istanza temporale ed utilizzano la stessa granularità temporale.
- editing = trattamento di un testo per assicurarne la conformità ad una ontologia.
- indipendente dal dominio = software, strumento o conoscenza globale applicabile ad una molteplicità di domini.
- mapping tra ontologie = trasformazione dei termini tra le ontologie, attraverso regole di accoppiamento, utilizzato per collegare utenti e risorse.
- Ontologia = descrizione particolareggiata di una concettualizzazione, i.e. l'insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti, spesso ambigui tra domini diversi.
- ontologia condivisa = sottoinsieme di diverse ontologie condiviso da una molteplicità di utenti.
- ontologia unita (merged) = ontologia creata combinando diverse ontologie, ottenuta mettendole in relazione tra loro (mapping).
- regole di accoppiamento (matching rules) = dichiarazioni per definire l'equivalenza tra termini di domini diversi.
- relazione = collegamento tra termini, come *is-a*, *part-of*,...
- 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.
- specifico ad un dominio = relativo ad un singolo dominio, presuppone l'assenza di incompatibilità semantiche.
- trasformazione dello schema = adattamento dello schema ad un'altra ontologia.

Appendice B

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

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

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

⟨attr_dcl⟩ ::= **[readonly] attribute**
⟨domain_type⟩ ⟨attribute_name⟩
[⟨fixed_array_size⟩] [⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ ::= **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ ::= ⟨rule⟩ | ⟨rule⟩**,**⟨rule_list⟩
⟨rule⟩ ::= ⟨local_attr_name⟩ | **'**⟨identifier⟩**'**
⟨and_expression⟩ | ⟨or_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and** ⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **and** ⟨and_list⟩
⟨or_expression⟩ ::= **(** ⟨local_attr_name⟩ **or** ⟨or_list⟩ **)**
⟨or_list⟩ ::= ⟨local_attr_name⟩ | ⟨local_attr_name⟩ **or** ⟨or_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩**.**⟨class_name⟩**.**⟨attribute_name⟩
...
⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** | ⟨relationship_dcl⟩**;** ⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_attr_name⟩ ⟨relationship_type⟩ ⟨local_attr_name⟩
⟨relationship_type⟩ ::= **syn** | **bt** | **nt** | **rt**
...
⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_pre⟩ **then** ⟨rule_post⟩
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** | ⟨rule_body⟩ |
⟨rule_body_list⟩ **and** ⟨rule_body⟩ |
⟨rule_body_list⟩ **and (** ⟨rule_body_list⟩ **)**
⟨rule_body⟩ ::= ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in** ⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
⟨rule_const_op⟩ ::= $=$ | $\geq$ | $\leq$ | $>$ | $<$
⟨rule_cast⟩ ::= **(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**⟨dotted_name⟩
⟨forall⟩ ::= **for all** | **forall**

Appendice C

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

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

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

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

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

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

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

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

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

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

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

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


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


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

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


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


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

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

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

Bibliografia

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

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

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

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

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

[6] Manlio Marchica. *La conoscenza di MOMIS: il ruolo di XML-Schema e RDF*. Master's thesis, Università degli studi di Modena e Reggio Emilia, Italy, 2001. Available at http://sparc20.dsi.unimo.it/tesi/index.html.

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

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

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

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

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

[12] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specification. Technical report, Stanford University, 1995. ftp://db.stanford.edu /pub/papakonstantinou/1995/ medmaker.ps.

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

[14] M.J.Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Object exchange across heterogeneous information sources. Technical report, Stanford University, 1994.

[15] M.T. Roth and P. Scharz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proc. of the 23rd Int. Conf. on Very Large Databases*, Athens, Greece, 1997.

[16] Y. Arens, C.Y. Chee, C. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

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

[18] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *ACM Symp. on Principles of Database Systems*, pages 51–61, 1997.

[19] Sap web site. Available at http://www.sap.com.

[20] Sonia Bergamaschi. Extraction of informations from highly heterogeneous source of textual data. In *First International Workshop CIA-97 COOPERATIVE INFORMATION AGENTS - DAI meets Database Systems, University of Kiel, Kiel, Germany*, 1997.

[21] W.H. Inmon and C. Kelley. Rdb/vms: Developing the data warehouses, 1993.

[22] AA. VV. The common object request broker: Architecture and specification. Technical report, Object Request Broker Task Force, 1993. Revision 1.2, Draft 29, December.

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

[24] Francesco Venuta. Il componente query manager di momis: esecuzione di interrogazioni. Master's thesis, Università degli studi di Modena e Reggio Emilia, Italy, 1999-2000.

[25] Cgi web site. Available at http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[26] Microsoft active server pages web site. Available at http://msdn.microsoft.com/asp.

[27] Sun java servlet web site. Available at http://java.sun.com/servlet.

[28] M. Risso, editor. *Java 2 programmazione distribuita*. Tecnes, Milano, Italy, 1999.

[29] Sun java 2 enterprise edition web site. Available at http://java.sun.com/j2ee.

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

[31] Dix, Finlay, Abowd, and Beale, editors. *Human Computer Interaction*. Prentice Hall Europe, 1998.