Università degli studi di Modena e Reggio Emilia

Facoltà di Ingegneria

Corso di Laurea Specialistica in
Ingegneria Informatica

Progetto e realizzazione dell'interfaccia web del QueryManager del sistema MOMIS

Relatore
Prof. Sonia Bergamaschi

Candidato
Sara Quattrini

Anno Accademico 2008/2009

Alla mia famiglia

Indice

Elenco delle figure

Introduzione

Il problema dell'integrazione dei dati è cruciale per molti scenari applicativi: la diffusione di Internet comporta una crescita esponenziale di dati e informazioni disponibili per l'utente, e ogni tipo di sorgente richiede un'interrogazione specifica, conformata a una sintassi particolare; altro problema correlato all'espansione della rete è quello della ridondanza dei dati: esistono molte copie della stessa informazione, spesso denominate con nomi diversi.

Dopo anni di duro lavoro, il team di DATARIVER, spin-off della Facoltà di Ingegneria dell'Università di Modena e Reggio Emilia è pronto a lanciare la prima versione Open Source di MOMIS, sistema per l'estrazione e l'integrazione dei dati da sorgenti strutturate e semi-strutturate, che propone una soluzione ai problemi sopra citati.

Lo sviluppo di MOMIS è iniziato nel 1992 e, fino ad oggi, è continuato il suo sviluppo da parte del DBGroup, il gruppo di ricerca nelle Basi di Dati dell'Università di Modena (http://www.dbgroup.unimo.it/).

Il mio contributo è consistito nel progetto e nella realizzazione di un'interfaccia web che guidi l'utente nella composizione ed esecuzione di interrogazioni sullo Schema Globale prodotto dalla metodologia di integrazione di MOMIS, e nella stesura di un Manuale d'uso in lingua inglese.

In questa tesi sono stati affrontati diversi argomenti, correlati al sistema MOMIS e all'interfaccia da me realizzata: è stata affrontata l'analisi del paradigma emergente dell'Open Source a cui la reliese uscente del sistema MOMIS è legata; sono state approfondite le tematiche della Data Integration con particolare riferimento all'interrogazione di Schemi Globali ottenuti dall'integrazione di sorgenti dati locali, eterogenee e distribuite; sono stati valutati gli strumenti di realizzazione di interfacce web legati al mondo Java e Open Source e sono state acquisite tutte le conoscenze necessarie per lo sviluppo di interfacce di applicazioni web.

In base alle conoscenze acquisite è stato possibile realizzare l'interfaccia web per il QueryManager di MOMIS, obiettivo di questa tesi.

Alla fase di progetto è seguita l'implementazione ed il testing esaustivo con riferimento ad un dominio di applicazione complesso.

Infine il manuale di MOMIS, in lingua inglese, costituisce una guida all'utilizzo del componente di creazione degli Schemi Globali e dell'interfaccia web del QueryManager, con riferimento al dominio applicativo della realizzazione di un portale verticale che integri tre siti web di riferimento per le prenotazioni di hotel e ristoranti (si veda il manuale in Appendice A).

La struttura della tesi è la seguente:

- Il primo capitolo è dedicato all'Open Source: viene riportata la sua storia, dalle origini alla situazione attuale, la descrizione delle licenze e i vantaggi e gli svantaggi derivanti dall'utilizzo di questo tipo di distribuzione software;
- Nel secondo capitolo è descritto il problema dell'integrazione dei dati e viene presentato il sistema MOMIS in generale;
- Il terzo capitolo è dedicato alla descrizione dell'interfaccia web del QueryManager di MOMIS, da me progettata e realizzata;
- Nel quarto capitolo vengono presentati il progetto dell'interfaccia web e la sua struttura, descrivendo nel dettaglio l'implementazione di alcune delle funzionalità offerte;
- Nell'Appendice A è riportato il Manuale di MOMIS da me redatto, costituito da una parte introduttiva sul sistema, una parte di guida all'utilizzo del componente di creazione dello Schema Globale e una parte descrittiva dell'interfaccia web del QueryManager e delle sue funzionalità;
- Nell'Appendice B vengono descritti i Frameworks e le tecnologie utilizzati per realizzare l'interfaccia: JPA (Java Persistence Api), Spring Framework, Apache Struts2, Ext-js e Ajax.

Capitolo 1

Open Source

In informatica, *Open Source*(termine inglese che significa sorgente aperto) indica un software i cui autori (più precisamente i detentori dei diritti) ne permettono, anzi ne favoriscono il libero utilizzo e l'apporto di modifiche da parte di altri programmatori indipendenti. Questo è realizzato mediante l'applicazione di apposite licenze d'uso.

La collaborazione di più parti (in genere libera e spontanea) permette al prodotto finale di raggiungere una complessità maggiore di quanto potrebbe ottenere un singolo gruppo di lavoro. L'open source ha tratto grande beneficio da Internet, perché permette a programmatori geograficamente distanti di coordinarsi e lavorare allo stesso progetto.

I software open source attualmente più diffusi sono: Firefox, OpenOffice, VLC, Gimp, 7-Zip, oltre ad un gran numero di progetti rivolti non all'utente finale ma ad altri programmatori. Sono inoltre degne di nota le famiglie di sistemi operativi BSD, GNU e il kernel Linux, i cui autori e fautori hanno contribuito in modo fondamentale alla nascita del movimento. La comunità open source è molto attiva, comprende decine di migliaia di progetti, numero che cresce quotidianamente.

Alla filosofia del movimento open source si ispira il movimento *open content* (contenuti aperti): in questo caso ad essere liberamente disponibile non è il codice sorgente di un software ma contenuti editoriali quali testi, immagini, video e musica. Wikipedia è un chiaro esempio dei frutti di questo movimento. Attualmente l'open source tende ad assumere rilievo filosofico, una nuova concezione della vita, aperta e refrattaria ad ogni oscurantismo, che ha come punto focale la condivisione della conoscenza.

1.1 La storia

A partire dagli anni cinquanta, e soprattutto negli anni sessanta, è stato possibile riusare lo stesso codice e distribuirlo anche se in modo oggi ritenuto piuttosto artigianale, ovvero con nastri e schede perforate. Questo fenomeno diventò evidente soprattutto quando si affermò il vantaggio di usare una stessa porzione di codice, il che presupponeva di avere macchine uguali e problemi simili.

Fino a tutti gli anni settanta, anche se in misura decrescente, la componente principale e costosa di un computer era l'hardware, il quale era comunque inutile in assenza di software. Da ciò la scelta dei produttori di hardware di vendere il loro prodotto accompagnato da più software possibili e di facilitarne la diffusione, fenomeno che rendeva più utili le loro macchine e dunque più concorrenziali. Il software, tra l'altro, non poteva avvantaggiare la concorrenza in quanto funzionava solo su un preciso tipo di computer e non su altri, neanche dello stesso produttore.

L'introduzione dei sistemi operativi rese i programmi sempre più portabili, in quanto lo stesso sistema operativo veniva offerto dal produttore di diversi modelli di hardware. La presenza di sistemi operativi funzionanti per macchine di differenti produttori hardware ampliava ulteriormente le possibilità di usare lo stesso codice in modo relativamente indipendente dall'hardware usato. Uno di questi sistemi operativi era Unix, iniziato nel 1969 come progetto all'interno di un'impresa delle telecomunicazioni, la AT&T. Una famosa causa antitrust contro la AT&T le vietò di entrare nel settore dell'informatica. Questo fece sì che Unix venisse distribuito ad un prezzo simbolico a buona parte delle istituzioni universitarie, le quali si ritrovarono ad avere una piattaforma comune, ma senza alcun supporto da parte del produttore. Si creò spontaneamente una rete di collaborazioni attorno al codice di questo sistema operativo, coordinata dall'Università di Berkeley, da dove sarebbe poi uscita la versione BSD di Unix, che diventa un centro di sviluppo ed innovazione.

Considerato che la condivisione del codice è nata insieme all'informatica, piuttosto che di origini dell'Open Source potrebbe essere più appropriato parlare, invece, di origine del software proprietario, ed esaminare il contesto storico in cui questa origine ha avuto luogo.

L'utilità principale delle licenze restrittive consiste nella possibilità di rivendere un programma più volte, se necessario con alcune modifiche purché non rilevanti. Questo presuppone che esistano clienti diversi con esigenze simili, oltre che l'esistenza di più computer sul quale poter far eseguire il programma. Queste condizioni cominciano a determinarsi negli anni sessanta, grazie al fatto che esisteva un maggior numero di utilizzatori con esi-

genze standardizzabili come lo erano quelle delle organizzazioni economiche nell'area della contabilità, della logistica o delle statistiche.

L'introduzione dei sistemi operativi rese inoltre possibile l'utilizzo dello stesso programma anche su hardware differente aumentando così le possibilità di riutilizzo dello stesso codice e dunque l'utilità nell'impedire la duplicazione non autorizzata dei programmi.

La suddivisione della AT&T in 26 società, le cosiddette BabyBell, permise alla AT&T di usare logiche prettamente commerciali nella distribuzione del suo sistema operativo Unix, innalzando notevolmente i costi delle licenze. Il 1982 fu anche l'anno della divisione delle diverse versioni commerciali di Unix, portate avanti dai singoli produttori di hardware. Questi ultimi, effettuando delle piccole modifiche alla propria versione del sistema operativo, impedirono ai propri utenti l'utilizzo di altri sistemi, facendo in modo che i programmi scritti per la propria versione di Unix non funzionassero su versioni concorrenti.

Al MIT (Massachusetts Institute of Technology) la sostituzione dei computer fece sì che i programmatori - fra i quali Richard Stallman che sarebbe diventato il portabandiera del software libero - non potessero accedere al sorgente del nuovo driver di una stampante Xerox per implementarvi una funzionalità gradita in passato: la segnalazione automatica che vi erano problemi con la carta inceppata. Contemporaneamente, società private cominciarono ad assumere diversi programmatori del MIT, e si diffuse la pratica di non rendere disponibili i sorgenti dei programmi firmando accordi di non divulgazione (in inglese: NDA, ovvero Non-Disclosure Agreement).

In questo contesto Stallman si rifiutò di lavorare per una società privata e fondò nel 1985 la Free Software Foundation (FSF), un'organizzazione senza fini di lucro per lo sviluppo e la distribuzione di software libero. In particolare lo sviluppo di un sistema operativo completo, compatibile con UNIX, ma distribuito con una licenza permissiva, con tutti gli strumenti necessari altrettanto liberi. Si tratta del progetto nato l'anno precedente, ovvero GNU, acronimo ricorsivo per contemporaneamente collegarsi e distinguersi da UNIX, ovvero *GNU's Not UNIX*.

L'obiettivo principale di GNU era essere software libero. Anche se GNU non avesse avuto alcun vantaggio tecnico su UNIX, avrebbe avuto sia un vantaggio sociale, permettendo agli utenti di cooperare, sia un vantaggio etico, rispettando la loro libertà. Tale progetto, finanziato dalla FSF, venne pertanto prodotto da programmatori appositamente stipendiati. I principali contributi vennero da Stallman stesso: il compilatore gcc (GNU Compiler Collection) e l'editor di testo Emacs . Furono sviluppate anche altre componenti di sistema UNIX, alle quali si sono aggiunte applicazioni per veri e propri giochi. Questi programmi furono distribuiti per circa 150$ che oltre a

coprire i costi di riproduzione garantivano un servizio di supporto al cliente. L'unica condizione era che tutte le modifiche eventualmente effettuate su tali programmi venissero notificate agli sviluppatori.

Nacque così la GNU General Public License (GPL), il preambolo del cui manifesto comincia con: "Le licenze per la maggioranza dei programmi hanno lo scopo di togliere all'utente la libertà di condividerlo e di modificarlo. Al contrario, la GPL è intesa a garantire la libertà di condividere e modificare il free software, al fine di assicurare che i programmi siano liberi per tutti i loro utenti".

Gli anni ottanta sono caratterizzati da alcuni eventi importanti, tra i quali l'introduzione nel mercato di quello che verrà chiamato Personal Computer (PC), ovvero un elaboratore con un proprio processore concepito per essere utilizzato da un solo utente alla volta. Il prodotto di maggior successo, il PC della IBM, si differenziava dai progetti precedenti in quanto non utilizzava componenti IBM, ma sia per il software che per l'hardware si affidava alla produzione da parte di terzi. Ciò rese possibile da un lato ad altre imprese di clonare il PC IBM, abbattendone notevolmente i costi, dall'altro permise a parecchie società di produrre dei software applicativi standard, in concorrenza gli uni con gli altri, basandosi su un unico sistema operativo, anche se inizialmente i principali produttori di software erano identificabili con prodotti per specifiche applicazioni.

Il notevole ampliamento del mercato rese possibili economie di scala e si instaurò una sorta di sinergia tra quelli che sarebbero diventati i principali attori del settore: il produttore dei processori Intel e il produttore del sistema operativo e di applicativi per ufficio Microsoft. La maggiore potenza dei processori rese possibile lo sviluppo di programmi più complessi, la maggiore complessità degli applicativi e del sistema operativo richiesero processori più potenti instaurando in un certo modo un circolo vizioso di aggiornamenti continui.

Sia il sistema operativo che gli applicativi furono subito caratterizzati dall'essere destinati ad utenti con conoscenze informatiche relativamente scarse e dall'avere licenze d'uso strettamente commerciali, vietando da un lato agli utenti di farne delle copie, dall'altro agli sviluppatori di vedere o modificare il codice. Sempre negli anni ottanta vennero introdotte le workstation, ovvero un sistema basato su terminali (i client) e computer centrali (i server). Si tratta di sistemi derivati concettualmente dai mainframe e basati essenzialmente su sistemi operativi UNIX proprietari. L'hardware stesso varia sul lato server dai mainframe ai PC, mentre su lato client vengono impiegati soprattutto i PC. Ciò favorì lo sviluppo di software sia per i client, utilizzati spesso da persone con scarse conoscenze informatiche, che per i server, il cui funzionamento viene solitamente garantito da personale informatico

particolarmente qualificato.

Benché Internet avesse visto la luce già negli anni settanta, è soltanto agli inizi degli anni novanta, con la diffusione del protocollo HTTP e la nascita dei primi browser, che Internet cominciò ad essere diffuso prima in ambito accademico e poi in modo sempre più capillare anche tra semplici privati.

All'inizio degli anni novanta, il progetto GNU non aveva ancora raggiunto il suo obiettivo principale, mancando di completare il kernel del suo sistema operativo (HURD).

Intanto, Linus Torvalds, studente al secondo anno di informatica presso l'Università di Helsinki, decise di sviluppare un proprio sistema operativo imitando le funzionalità di Unix su un PC con un processore Intel 386. Tale processore venne scelto per il suo minor costo e per la sua maggiore diffusione rispetto alle piattaforme hardware per le quali erano disponibili i sistemi operativi Unix. Torvalds era spinto dall'insoddisfazione riguardante alcuni applicativi di Minix (un sistema Unix-like su piattaforma PC), dal desiderio di approfondire le proprie conoscenze del processore Intel 386, e dall'entusiasmo per le caratteristiche tecniche di Unix.

Torvalds distribuì il proprio lavoro tramite Internet e ricevette immediatamente un ampio riscontro positivo da parte di altri programmatori, i quali apportarono nuove funzionalità e contribuirono a correggere errori riscontrati. Nacque così il kernel Linux, il quale fu subito distribuito con licenza libera.

Internet dal canto suo, rende possibile la comunicazione tra persone molto distanti in tempi rapidi e a basso costo. Inoltre rende possibile la distribuzione di software direttamente dalla rete, riducendo ulteriormente i costi di duplicazione e le difficoltà a reperire il software stesso. La diffusione dei CD-ROM come supporto privilegiato di raccolte di software rese possibile il fenomeno delle cosiddette distribuzioni.

Linux può essere considerato come il primo vero progetto Open Source cioè come il primo progetto che faceva affidamento essenzialmente sulla collaborazione via Internet per progredire; fino ad allora, infatti, anche i progetti di software libero come Emacs erano stati sviluppati in maniera centralizzata seguendo un progetto prestabilito da un ristretto numero di persone, in base cioè ai principi 'standard' di ingegneria del software. Si assumeva valida anche per i progetti open source la 'legge di Brooks', secondo cui "aggiungere sviluppatori a un progetto in corso di implementazione in realtà rallenta il suo sviluppo", legge che ovviamente non è applicabile a un progetto di sviluppo open source.

Agli inizi degli anni novanta, l'idea delle licenze libere era rappresentata soprattutto da Richard Stallman e la sua FSF, ovvero le licenze libere per eccellenza erano la GPL e la LGPL che però venivano ritenute "contagiose",

in quanto a partire da un codice licenziato con la GPL qualsiasi ulteriore modifica doveva avere la stessa licenza. Le idee stesse di Stallman venivano viste con sospetto dall'ambiente commerciale statunitense, il che non facilitava la diffusione del software libero. Per favorire dunque l'idea delle licenze libere nel mondo degli affari, Bruce Perens, Eric S. Raymond, Ockman e altri cominciarono nel 1997 a pensare di creare una ridefinizione ideologica del software libero, evidenziandone cioè i vantaggi pratici per le aziende e coniarono il termine "Open Source". Ciò anche al fine di evitare l'equivoco dovuto al doppio significato di free nella lingua inglese, visto che spesso veniva interpretato come "gratuito" invece che come "libero".

L'iniziativa venne portata avanti soprattutto da parte di Raymond che, in occasione della liberalizzazione del codice sorgente di Netscape, voleva utilizzare un tipo di licenza meno restrittivo per le aziende di quanto fosse il GPL.

La scelta a favore dell'Open Source da parte di alcune importanti imprese del settore come Netscape, IBM, Sun Microsystems e HP, facilitò inoltre l'accettazione del movimento Open Source presso l'industria del software, facendo uscire l'idea della "condivisione del codice" dalla cerchia ristretta nella quale era rimasta relegata fino ad allora. Venne cioè accettata l'idea che l'open source fosse una metodologia di produzione software efficace.

1.2 La licenza

Una licenza Open Source è una licenza concessa dal detentore di un diritto d'autore utilizzata prevalentemente nell'ambito dell'informatica riguardante solitamente il software, ma che può riguardare qualsiasi altro ambito nel quale si applica la normativa sul diritto d'autore.

Questa licenza autorizza chiunque ad usare, modificare, integrare, riprodurre, duplicare e distribuire un programma (o qualsiasi lavoro tutelato dalle norme sul diritto d'autore), anche a scopi commerciali.

Alcune licenze (come per esempio la GNU GPL) impongono dei vincoli alle licenze da applicare ad eventuali modifiche, oppure degli obblighi di comunicazione a determinate persone o gruppi di persone. Ci sono licenze i cui autori si riservano dei diritti che non vengono concessi agli altri coautori del prodotto, altre invece permettono persino di integrare completamente il codice dentro un prodotto commerciale, ovvero soggetto a royalty, come ad esempio la licenza MIT.

Lo scopo primario delle licenze open source non è la gratuità del software, ma la sua sopravvivenza ovvero la certezza che vi sia la possibilità per chiun-

que e in qualunque momento, anche futuro, di apportare miglioramenti o comunque modifiche al programma, e di installarlo senza alcuna limitazione.

Per alcuni esponenti della comunità del Software Libero, come Stallman, lo scopo primario è la libertà del software in sé, in quanto più importante rispetto agli aspetti tecnologici. Secondo Stallmann, il software dovrebbe essere liberamente utilizzabile prima di tutto perché non è etico brevettarlo, e, solo in secondo luogo, perché è di migliore qualità.

La Open Source Definition definisce quali licenze possono essere considerate open source. Questa definizione è stata fatta dalla fondazione Open Source Initiative (OSI) che tuttora gestisce il marchio creato ad hoc. La definizione deriva dalle regole (dette Debian Free Software Guidelines) che erano state fissate per il progetto Debian per scegliere quali software includere nella propria distribuzione GNU/Linux. Secondo questa definizione è evidente che perché una licenza sia open source non si deve soltanto di avere accesso al codice sorgente, ma anche il permesso a chiunque di mettere mano al codice sorgente e contemporaneamente il permesso di ridistribuirlo, il tutto senza che alcuno possa pretendere anche il minimo compenso, però senza impedire di chiedere un compenso a chi è disposto a pagarlo.

Secondo la Open Source Definition affinché si possa parlare di una licenza open source è necessario che tale licenza soddisfi contemporaneamente tutte le condizioni sotto indicate:

- Ridistribuzione libera. La licenza non può impedire ad alcuna parte in causa la vendita o la cessione del software. Chiunque deve poter fare tutte le copie che vuole, venderle o cederle, e non deve pagare nessuno per poter fare ciò.

- Codice sorgente. Il programma deve includere il codice sorgente. Codice deliberatamente offuscato non è ammesso. Questo in quanto il codice sorgente è necessario per modificare o riparare un programma.

- Opere derivate. La licenza deve permettere modifiche e opere derivate e deve consentire la loro distribuzione sotto i medesimi termini della licenza del software originale, in quanto il software serve a poco se non si può modificare per fare la manutenzione ad esempio per la correzione di errori o il porting su altri sistemi operativi.

- Integrità del codice sorgente dell'autore. La licenza può proibire che il codice sorgente venga distribuito in forma modificata solo se la licenza permette la distribuzione di pezze ("patch file") con il codice sorgente allo scopo di migliorare il programma al momento della costruzione.

- Nessuna discriminazione contro persone o gruppi. La licenza deve essere applicabile per tutti, senza alcuna discriminazione per quanto nobile possa essere l'obiettivo della discriminazione. Ad esempio non si può negare la licenza d'uso neanche a forze di polizia di regimi dittatoriali.

- Nessuna discriminazione di settori. Analogamente alla condizione precedente, questa impedisce che si possa negare la licenza d'uso in determinati settori, per quanto questi possano essere deplorevoli. Non si può dunque impedire l'uso di tale software per produrre armi chimiche o altri strumenti di distruzione di massa.

- Distribuzione della licenza. I diritti relativi al programma devono applicarsi a tutti coloro ai quali il programma sia ridistribuito, senza necessità di esecuzione di una licenza aggiuntiva.

- La licenza non dev'essere specifica a un prodotto. I diritti relativi a un programma non devono dipendere dall'essere il programma parte di una particolare distribuzione di software.

- La licenza non deve contaminare altro software. La licenza non deve porre restrizioni ad altro software che sia distribuito insieme a quello licenziato.

- La licenza deve essere tecnologicamente neutra. Nessuna clausola della licenza deve essere proclamata su alcuna singola tecnologia o stile di interfaccia.

La OSI ha una lista di licenze open source. Perché una licenza vada in questa lista deve rispettare la Open Source Definition e deve seguire un processo di approvazione.

La Free Software Foundation (FSF) ha a sua volta una lista di licenze ritenute libere (nella lista ci sono anche licenze ritenute da alcuni erroneamente libere e la spiegazione del perché non lo sono), per ognuna c'è scritto se è compatibile o no con la GNU General Public License. La lista delle licenze open source (secondo la definizione OSI) e la lista delle licenze libere (secondo la definizione della FSF) sono quasi coincidenti, salvo alcune eccezioni.

In generale le licenze open source non sono a priori reciprocamente compatibili. Il titolare dei diritti d'autore può comunque distribuire il proprio codice con diverse licenze, sia open source che commerciali. Questo vale sia per l'iniziatore del progetto che per gli autori che contribuiscono al progetto, ciascuno per il proprio codice. Questa possibilità, detta pure dual-licensing o dual-system viene effettivamente praticata, per esempio dalla Sun per la propria Suite Star Office, ma anche da Larry Wall per l'interprete Perl.

1.3 Innovazione e qualità

In prospettiva informatica, i progetti Open Source hanno introdotto un modello di sviluppo software innovativo rispetto a quelli classici dell'Ingegneria del Software, dove viene data molta importanza alle attività di progettazione, costruzione e mantenimento del software. Pertanto, la qualità dei processi determina la qualità del prodotto finale, e di conseguenza una corretta definizione, progettazione e gestione dei processi di sviluppo è indispensabile per garantire la qualità di un prodotto software.

Il modello di sviluppo Open Source è basato sulla collaborazione volontaria ed indipendente di una comunità di programmatori. Conseguentemente è data molta importanza alle risorse umane, alle conoscenze e alle competenze dei singoli soggetti della comunità. I progetti Open Source di grandi dimensioni vengono divisi in sottoprogetti e le varie funzionalità sviluppate in modo incrementale, sfruttando l'esecuzione parallela delle attività di sviluppo per velocizzare il processo di convergenza verso la soluzione migliore.

Il modello Open Source riconosce infine l'importanza del feedback per la comprensione dei requisiti e la definizione delle specifiche. A causa della natura aperta e non regolamentata delle attività di sviluppo Open Source, il processo non è valutabile con i modelli di valutazione della qualità preesistenti, pensati e basati per un'organizzazione fortemente gerarchica e con regole codificate. Nonostante ciò, i prodotti Open Source sono in molti casi una valida alternativa ai prodotti proprietari non solo grazie al prezzo contenuto ma anche in virtù di alcune caratteristiche di qualità che li contraddistinguono.

Volendo effettuare un'analisi qualitativa, occorre definire quali siano i parametri utili per effettuare una valutazione oggettiva della vera qualità dei prodotti Open Source anche in riferimento ai prodotti proprietari alternativi. La valutazione dei prodotti software fa riferimento ai seguenti parametri: affidabilità, ovvero la capacità del prodotto di mantenere le proprie prestazioni nel tempo e nelle diverse situazioni; performance, ovvero prestazioni del software con particolare riferimento a velocità e funzionalità implementate; costo di possesso, ovvero la composizione di costo di acquisto e di mantenimento del prodotto.

Oltre alla valutazione della qualità, la diffusione dei prodotti Open Source dipende anche da parametri per i quali è difficile stabilire metriche e quindi valori quantitativi. Tra gli altri, la portabilità, ovvero la compatibilità del prodotto con sistemi hardware e altri sistemi software; la sicurezza, ovvero la capacità di un prodotto di difendersi da possibili violazioni dell'integrità; la flessibilità, ovvero la capacità di prodotto di rispondere a mutevoli esigenze degli utenti. Infine, è indispensabile valutare come e quanto un software sia aderente ad una determinata licenza, e quanto facile sia, per l'utente finale o

per uno sviluppatore, venire a conoscenza della licenza adottata e delle sue clausole.

1.4 I vantaggi

I vantaggi che il cliente trae dall'utilizzo di software Open Source rispetto a quello proprietario (chiamato anche privativo) si possono riassumere brevemente nei seguenti punti:

1. Minore Total Cost of Ownership (TCO)
2. Estrema personalizzazione
3. Svincolamento dal Single Vendor Lock-In
4. Protezione dall'inaccessibilità del fornitore
5. Migliore compatibilità e integrazione dei sistemi

Il costo totale sostenuto per l'adozione di un software, chiamato appunto Total Cost of Ownership, è dato da diversi fattori tra cui il costo iniziale di acquisto della licenza e il costo di mantenimento; benché il concetto di Open Source sia del tutto staccato dal concetto di gratuità del software è pur vero che una larga fetta di software Open Source non richiede alcuna spesa iniziale di acquisto, eliminando o riducendo fortemente la spesa per le licenze.

C'è da aggiungere inoltre che il software Open Source gode di un'ottima reputazione per quanto riguarda sicurezza e stabilità del codice; ciò è da imputare principalmente al metodo di sviluppo del software e in particolare alla revisione paritaria del codice.

Essendo il codice sorgente accessibile e visionabile da chiunque tendenzialmente il software Open Source ha una base di persone che testano e controllano il codice molto più ampia dei software proprietari; questo fattore si è dimostrato uno strumento eccezionalmente efficace per ottenere software di elevata qualità ad un costo minore.

Al risparmio iniziale della licenza si somma quindi una maggiore affidabilità del sistema che può essere, a lungo andare, una voce di spesa molto più corposa della licenza stessa.

La disponibilità del codice sorgente consente inoltre una personalizzazione e una raffinazione del prodotto estrema, permettendo di modificare e adattare ad esigenze specifiche ogni aspetto del software. Questa possibilità è spesso del tutto assente nel software proprietario o comunque incomparabilmente più limitata.

Il *Single Vendor Lock-In* è un fenomeno conosciuto in tutti i settori, non solo in quello del software, che porta ad essere legati in esclusiva ad un unico fornitore per alcuni servizi o prodotti.

Questa condizione di dipendenza diretta da un unico soggetto è particolarmente odiata perché sbilancia enormemente le capacità di contrattazione da parte del fornitore che può approfittare della posizione di esclusività per imporre al cliente condizioni inique.

Questa posizione di asimmetria contrattuale è la più ricercata dai fornitori di software; gli strumenti comunemente utilizzati sono principalmente rivolti ad eliminare o a limitare fortemente l'interoperabilità con altri sistemi per impedire che altri concorrenti possano subentrare.

Tale obbiettivo viene portato avanti utilizzando, ad esempio, formati per i dati segreti e incompatibili e nascondendo i dettagli tecnici del software impedendo la collaborazione con sistemi di terzi parti.

Lo scenario appena presentato non è possibile con il software Open Source: avendo a disposizione il codice sorgente i dettagli interni del software sono automaticamente esposti così come il formato utilizzato per la memorizzazione dei dati, risolvendo alla radice il problema.

Inoltre, nel momento in cui la casa madre non dovesse essere più disponibile o in grado di far fronte agli impegni, o, per un qualunque motivo, il cliente volesse cambiare fornitore, la disponibilità del codice garantisce la continuità del progetto oltre la vita stessa del fornitore originale.

Il software Open Source permette intrinsecamente una migliore interoperabilità tra i sistemi, proprio perché ogni aspetto del software è aperto e ispezionabile; l'interoperabilità dei prodotti proprietari è invece comunemente garantita solo per il software della stessa casa madre.

E' evidente come per il cliente l'Open Source, anche a parità di costo iniziale, sia la strada più conveniente e affidabile nel tempo.

1.5 Strutture ed enti che utilizzano prodotti Open Source

In ambito istituzionale gli utilizzatori preferenziali di prodotti Open Source sono in particolare tutti quegli enti dove sicurezza, flessibilità e performance hanno un'importanza chiave come:

- La National Security Agency (NSA), che ha anche contribuito direttamente allo sviluppo del codice sorgente di Linux sviluppandone un'estensione, chiamata Security Enhanced Linux (SELinux).

- L'FBI, che utilizza diversi software Open Source (tra cui Apache e MySQL) per il sistema di risposta alla emergenze per lo stato del Texas.

- La marina americana, che utilizza Linux per i sonar impiegati nei sottomarini nucleari.

- La NASA, uno degli utilizzatori storici di Linux e del software Open Source in generale; già dal 1997 il controllo degli esperimenti sullo space Shuttle è affidato a software open.

- Lo United States Departement of Energy, che già nel 2002 possedeva il centro di calcolo Linux più potente del mondo.

- Il servizio postale americano, che utilizza software open dal 1997 per il riconoscimento automatico degli indirizzi sulle buste.

- La città di Monaco, impegnata dal 2002 nella migrazione dell'intera infrastruttura informatica comunale a software open.

- La città di Berlino, impegnata dal 2005 nella migrazione dell'intera infrastruttura del comune a software Open Source, operazione che coinvolge più di 58000 postazioni di lavoro e diverse migliaia di server.

- Il governo svizzero, impegnato dal 2005 nella migrazione dell'infrastruttura dei server utilizzati dagli enti pubblici a Linux.

- Il ministero degli affari interni giapponese, impegnato dal 2005 nella migrazione a software open delle infrastrutture chiave del paese con l'intento di diminuire la dipendenza del paese dall'estero e in particolare da Microsoft.

A livello italiano il software open risulta distribuito, in ambito istituzionale, a macchia di leopardo, con alcune punte di eccellenza come la provincia autonoma di Trento che viene spesso citata come esempio di realtà virtuosa in questo ambito.

In ambito locale nel comune di Modena esistono diversi enti che già utilizzano software Open Source, come ad esempio il servizio ospedaliero locale; in Comune si è inoltre avviato da qualche tempo un progetto per la migrazione graduale da Microsoft Office a OpenOffice.org.

1.6 Considerazioni finali

Concludendo, in prospettiva giuridica un filone di studi ha teorizzato l'idoneità del modello open source a garantire agli utenti di software (e più in generale a tutta la collettività) un godimento più intenso di alcuni diritti fondamentali, quali ad esempio la libertà di manifestazione del pensiero, della ricerca scientifica e tecnica, di accesso ai fenomeni culturali ed all'informazione; ed ancora la funzionalità di questo modello quale mezzo per elevare il tasso di democraticità delle istituzioni di governo delle reti telematiche, che nella società attuale sono ormai infrastrutture fondamentali per ogni genere di relazioni culturali, politiche, sociali ed economiche.

Capitolo 2

MOMIS

MOMIS (Mediator envirOnment for Multiple Information Sources) è un sistema Open Source per l'estrazione e l'integrazione dei dati da sorgenti strutturate e semi-strutturate, che verrà distribuito, a partire da Aprile 2010, da DataRiver (http://wwww.datariver.it), spin-off universitaria istituita nel giugno 2009.

Lo sviluppo di MOMIS e' iniziato nell'ambito del progetto nazionale INTERDATA, e nell'ambito del progetto D2I, sotto la direzione della Professoressa S. Bergamaschi. L'attività di ricerca è continuata all'interno del progetto di ricerca europeo IST "SEWASIE: Semantic Webs and Agents in Integrated Economies" (2002/2005). E' stato inoltre esteso nell'ambito del progetto MUR "NeP4B: Networked Peers for Business" (2006/2009) e del progetto IST-EU RDT "STASIS (SofTware for Ambient Semantic Interoperable Services)" (2006-2009). Nei seguenti paragrafi viene descritto lo scenario in cui MOMIS si inserisce, e i problemi principali che si impone di affrontare.

2.1 Scenario

Siamo nell'era di Internet e dei motori di ricerca, la disponibilità e la quantità di informazioni cresce ogni giorno di più.

L'utente che si presta a fare una ricerca si ritrova sommerso da risultati, fonti e collegamenti, risentendo così del cosiddetto problema di "information overload": la quantità elevata di informazioni porta confusione e il rischio di deviare l'attenzione da ciò che si stava cercando. Senza l'applicazione di opportuni metodi, gli utenti hanno a disposizione grandi quantità di dati, ma trovano inevitabili difficoltà nel sintetizzare l'informazione utile ai propri scopi.

In questo contesto si posiziona anche il problema della ridondanza dei dati: molte informazioni sono ripetute, in fonti diverse o addirittura in sorgenti di tipo diverso, e la situazione attuale rivela il bisogno di fare in modo che la ricerca di informazioni riporti risultati univoci, diversi tra di loro.

Altra problematica relativa alla ricerca di informazioni è l'eterogeneità e l'indipendenza delle sorgenti: sorgenti di tipo diverso richiedono interrogazioni conformi alla loro struttura, e informazioni correlate distribuite su sorgenti diverse sono difficili da reperire con un'unica interrogazione.

In questo scenario si colloca il problema della "Data Integration", l'integrazione di dati provenienti da sorgenti diverse, interrogabili dall'utente in modo univoco e trasparente rispetto al tipo di sorgente.

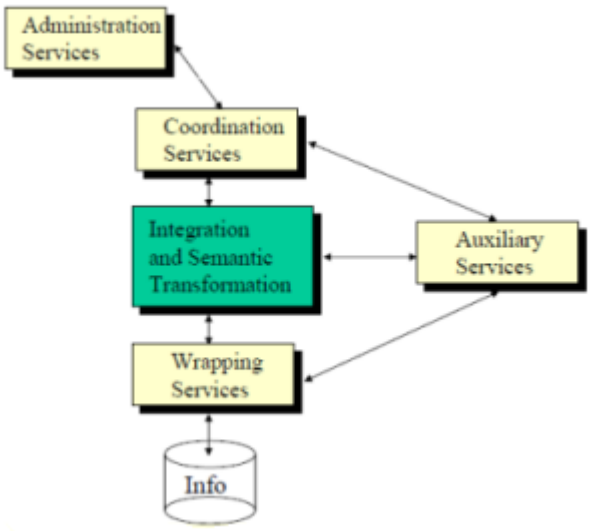


Fig. 2.1: I3: Architettura di riferimento

Il programma I3 (Integrazione Intelligente dell'Informazione) definisce l'architettura di riferimento (figura 2.1) per l'integrazione automatica di sorgenti eterogenee strutturate, semi-strutturate e non strutturate e si pone l'obiettivo di consentire l'integrazione dei dati evitando la loro replicazione, risolvendo eventuali conflitti e arricchendo i dati e le loro relazioni con ulteriori informazioni di tipo semantico (grazie a tecniche di Intelligenza Artificiale), rendendo l'interrogazione delle sorgenti sempre più specifica ed efficace.

2.2 Data Integration

Lo sviluppo di metodi e strumenti per l'integrazione di dati provenienti da sorgenti fortemente e strutturalmente eterogenee, ossia da sorgenti di tipo strutturato (ad esempio, basi di dati), o semistrutturato (ad esempio, documenti HTML e XML) è una problematica di interesse crescente nel campo della gestione di dati Web.

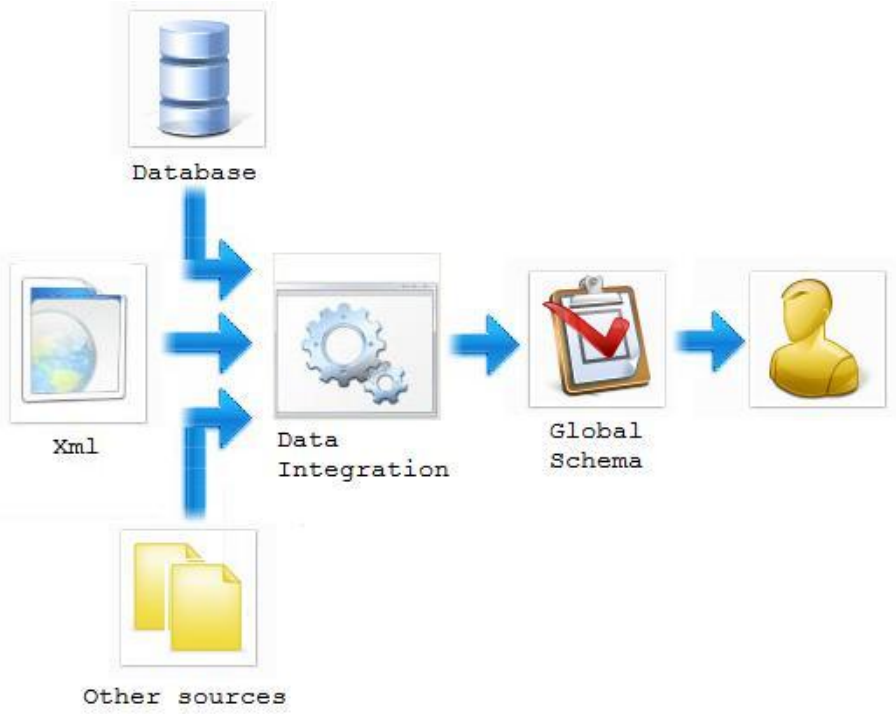


Fig. 2.2: Data integration

Il problema dell'integrazione ha ricevuto moltissima attenzione negli ultimi anni, ma esistono, allo stato attuale, poche soluzioni complete. La causa principale di questo fatto è probabilmente la complessità intrinseca del problema. E' infatti estremamente difficile fornire una visione unitaria di un insieme di sorgenti di dati eterogenee e sviluppate in modo autonomo.

Il processo di integrazione può essere schematizzato come il risultato di varie attività principali. Il punto di partenza è tipicamente la ricerca di corrispondenze tra schemi (il cosiddetto *schema matching*), in cui vengono scoperte analogie tra le descrizioni e il contenuto delle varie sorgenti di dati, definendo modalità per passare da uno schema all'altro. Successivamente, si pone il problema della traduzione delle istanze, in cui, sulla base delle corrispondenze tra schemi scoperte al passo precedente, vengono affrontati

tutti i problemi che riguardano la traduzione dei dati (*data translation*); in questa fase si pongono, tipicamente, problemi di data cleaning, incluso il rilevamento e la soluzione di possibili inconsistenze tra formati e vincoli delle sorgenti. Infine, è necessario risolvere il problema dell'esecuzione delle query (*query execution*), che consente effettivamente di stabilire la cooperazione applicativa tra le sorgenti di dati.

Risulta quindi importante ricercare nuove metodologie per l'integrazione di sorgenti eterogenee di dati e per la scoperta di nuovi collegamenti e proprietà non facilmente intuibili all'interno di una sorgente o di sorgenti diversi (*data mining*).

Gli sviluppi dell'informatica e delle telecomunicazioni hanno reso disponibile l'accesso ad un numero sempre più vasto di banche dati strutturate e semistrutturate, create in tempi diversi, su sistemi diversi e con criteri organizzativi diversi.

I metodi di rappresentazione dei dati presenti nelle sorgenti devono tenere conto di sorgenti semistrutturate, e di possibile coesistenza di versioni diverse dei dati. Si deve poi tenere conto che la scoperta di proprietà inter-schema è cruciale per caratterizzare le relazioni semantiche tra dati in diverse sorgenti. Infine, il processo che conduce alla risposta ad interrogazioni poste in termini di viste globali pone problemi sia per la suddivisione della query in sottoquery, sia per la ricostruzione della risposta.

Tali problematiche hanno riguardato la definizione di una metodologia di integrazione di sorgenti fortemente eterogenee, la definizione di tecniche semiautomatiche di clustering di sorgenti basate su proprietà di affinità e corrispondenze semantiche, la progettazione di algoritmi per la riscrittura di interrogazioni su viste globali in termini di interrogazioni sulle sorgenti, la definizione di metodi per la gestione di versioni diverse delle sorgenti, la caratterizzazione di opportuni parametri per descrivere la qualità dei dati, e di tecniche per la riconciliazione di dati provienti da sorgenti diverse, la progettazione e la realizzazione di un ambiente che supporti l'attività d'integrazione, basato sulla gestione di meta-dati.

I problemi principali relativi al processo di data integration (schematizzato in figura 2.3) sono:

- wrapping delle sorgenti: i wrapper sono correlati al tipo di sorgente e si occupano della generazione degli schemi locali in un formato comune.
- riconoscimento di proprietà interschema e gestione delle stesse.
- mapping delle sorgenti locali nel Global Schema.
- modelling delle classi globali del Global Schema.

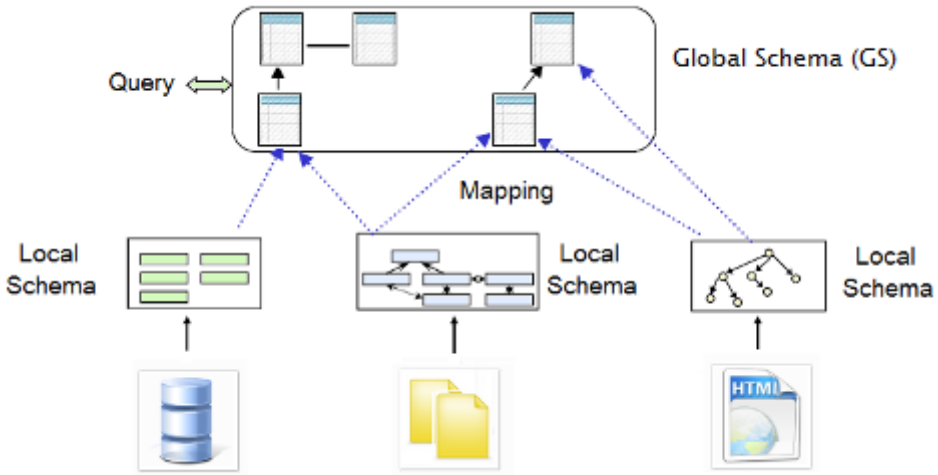


Fig. 2.3: Processo di Data Integration

- gestione delle query che interrogano lo schema globale: traduzione della query globale in distinte query locali (una per ogni tipo di sorgente) e fusione di tutte le risposte locali in un'unica risposta globale.

MOMIS ha affrontato questi problemi, risolvendoli e implementando una soluzione efficace.

L'architettura funzionale di MOMIS e la descrizione dell'applicazione non verranno riportate in questo capitolo poichè presenti nel Manuale da me redatto (che si trova nell'Appendice A di questa tesi). Il manuale è in inglese e contiene una parte descrittiva del sistema e una parte di guida all'utilizzo di MOMIS e dell'interfaccia web per il QueryManager da me realizzata.

Capitolo 3

Interfaccia web del QueryManager di MOMIS

3.1 Il progetto

L'idea di questa interfaccia web nasce dalla volontà di fornire un ulteriore strumento utile per interrogare gli Schemi Globali creati con MOMIS. Quello che si voleva realizzare era uno strumento dalla grafica semplice ma allo stesso tempo accattivante, che aiutasse l'utente, passo dopo passo, nella composizione ed esecuzione delle queries. In sostanza gli obiettivi posti erano i seguenti:

1. Facilitare la comprensione logica della struttura dello Schema Globale (tramite la visualizzazione grafica delle sorgenti globali e della *Mapping Table* di classi globali);

2. Facilitare all'utente il processo di composizione delle queries (tramite strumenti grafici e logici);

3. Presentare i risultati delle queries eseguite in modo chiaro e ordinato;

4. Dare all'utente la possibilità di salvare la sessione di lavoro (query e Schema Globale su cui viene eseguita).

Ciascuno di questi obiettivi è stato raggiunto con successo, l'applicazione fornisce tutte le funzionalità sopra elencate e provvederò a mostrarle nel dettaglio nei seguenti capitoli.

3.2 L'interfaccia

All'avvio l'applicazione presenta una schermata di autenticazione, necessaria per poter salvare successivamente queries e Schemi Globali per l'utente. La pagina di autenticazione è mostrata in Fig. 3.1.

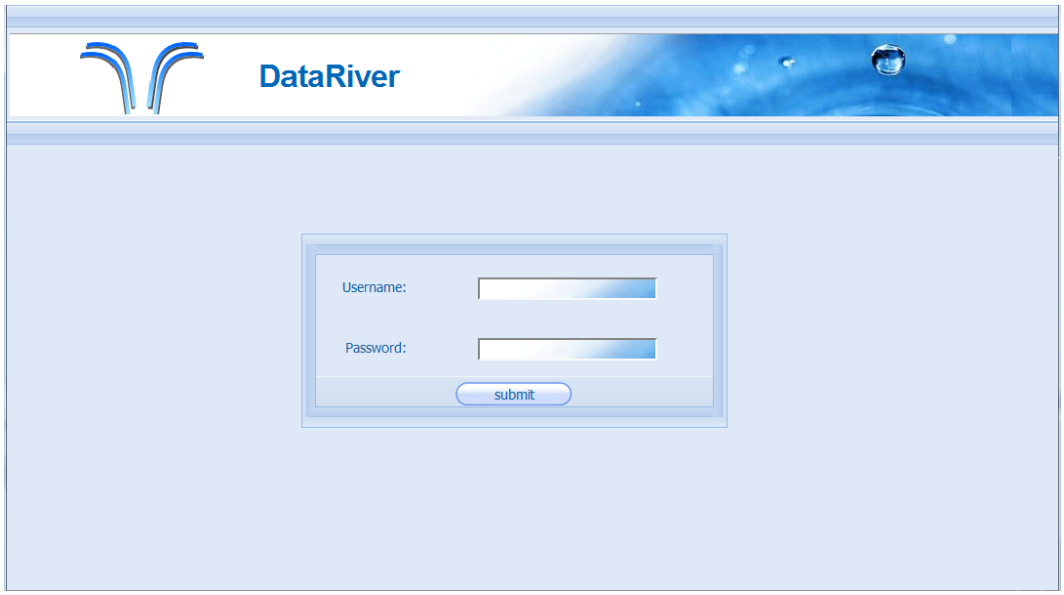


Fig. 3.1: Pagina di autenticazione

Una volta autenticato l'utente può utilizzare l'applicazione in tutte le sue parti, e viene rediretto alla pagina principale (Fig. 3.2) che si compone di 4 pannelli:

1. *Global Sources Panel*, contiene l'albero delle sorgenti globali;
2. *Class Attributes Panel*, inizialmente vuoto, viene successivamente riempito con gli attributi delle classi selezionate dall'albero;
3. *Query Panel* è il pannello principale, contiene l'editor (inizialmente disabilitato) in cui scrivere la query e i comandi principali per la composizione e l'esecuzione della stessa;
4. *Results Panel*, inizialmente vuoto, conterrà i risultati delle queries eseguite.

L'applicazione è dotata di un pannello di aiuto (Fig. 3.3), che si apre premendo il pulsante *Help* e contiene tutte le informazioni utili all'utente per il corretto utilizzo dell'interfaccia.

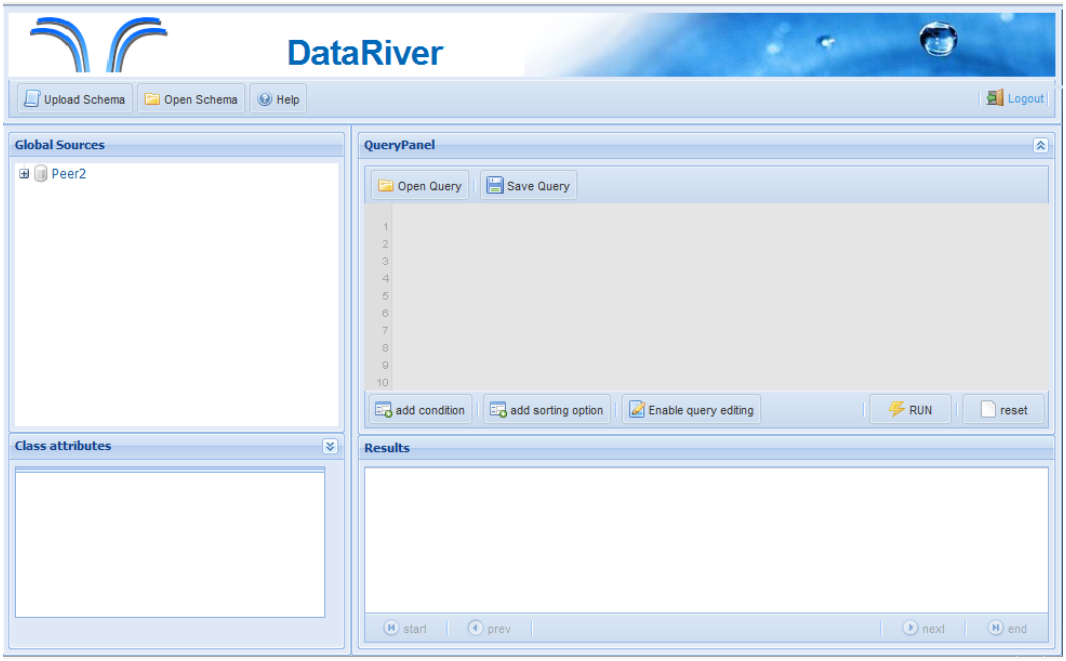


Fig. 3.2: Interfaccia web del Query Manager

Fig. 3.3: Pannello di Help

3.3 Rappresentazione grafica dello Schema Globale

Gli strumenti realizzati per facilitare la comprensione logica della struttura dello Schema Globale sono:

- l'albero delle sorgenti globali
- la *Mapping Table* delle classi globali.

3.3.1 L'albero delle sorgenti globali

L'albero delle sorgenti globali permette all'utente di navigare tra le classi globali scoprendone gli attributi, le chiavi primarie e le referenze.Inoltre, fermando il cursore per qualche secondo su una classe globale, è possibile visualizzare in un "QuickTip" le classi locali mappate nella stessa, mentre fermando il cursore sulla sorgente globale è possibile visualizzare nel "QuickTip" tutte le sorgenti locali su cui è stato costruito.

In Fig. 3.4 è mostrato l'albero di una sorgente globale chiamata Peer2, che ha 3 classi globali (Campings, Facilities e Hotels); in particolare è aperto il nodo della classe Facilities, che mostra la sua chiave primaria e le classi che referenzia (Campings e Hotels).

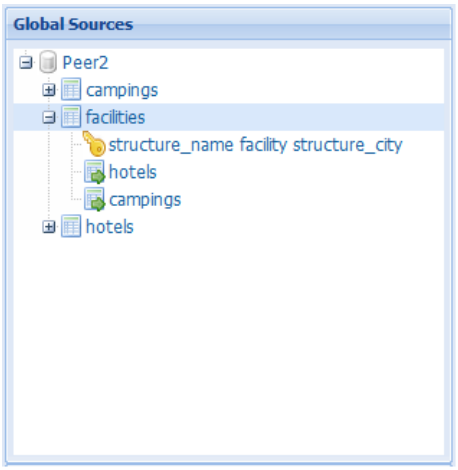
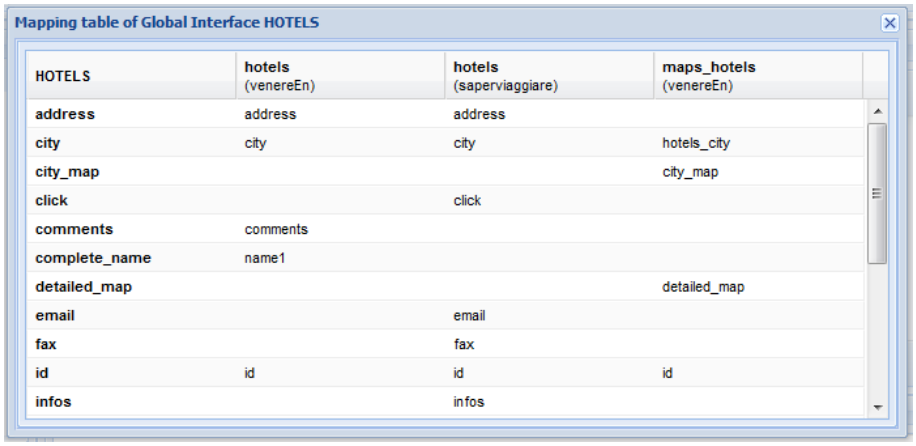


Fig. 3.4: L'albero delle sorgenti globali

3.3.2 La Mapping Table

La Mapping Table è uno strumento grafico che mostra quali attributi locali vengono 'mappati' in un attributo globale; è presente anche in MOMIS, in una forma più articolata, ma si è pensato di proporla anche in questa interfaccia web perchè all'utente potrebbe tornare utile vedere le informazioni sul 'Mapping' anche nel momento di interrogazione dello Schema Globale.



HOTELS	hotels (venereEn)	hotels (saperviaggiare)	maps_hotels (venereEn)
address	address	address	
city	city	city	hotels_city
city_map			city_map
click		click	
comments	comments		
complete_name	name1		
detailed_map			detailed_map
email		email	
fax		fax	
id	id	id	id
infos		infos	

Fig. 3.5: Mapping Table della classe globale Hotels

In Fig. 3.5 è mostrata la *MappingTable* della classe globale Hotels: nella riga di ogni attributo globale sono presentati gli attributi locali mappati in esso, e le interfacce locali di provenienza (per esempio nell'attributo globale "city" sono mappati gli attributi locali "city" dell'interfaccia Hotels della sorgente VenereEn,"city" dell'interfaccia Hotels della sorgente Saperviaggiare e "hotels_city" dell'interfaccia Maps_Hotels della sorgente VenereEn).

3.4 Processo di composizione delle queries

L'applicazione fornisce un servizio di composizione semi-automatica delle query. Il processo è diviso in 3 passi:

1. Selezione della/e classe/i globale/i
2. Selezione degli attributi globali di interesse
3. Aggiunta di eventuali condizioni o opzioni di ordinamento dei risultati

La selezione della classe globale avviene cliccando sulla classe stessa nell'albero delle sorgenti globali; gli attributi della classe selezionata compaiono

nel pannello "Class Attributes" in forma di checkbox, e se selezionati vengono scritti direttamente nell'editor insieme alla loro classe di appartenenza, secondo la sintassi

```
select <attribute-list> from <class-list> where <conditions>
```

In caso si voglia effettuare un JOIN tra due classi, basta cliccare su una classe e poi sulla classe referenziata di interesse; gli attributi di entrambe le classi vengono riportati nel pannello "Class Attributes", dove è possibile selezionarli, e la condizione di Join tra le due classi viene scritta automaticamente nell'editor.

Un'esempio è presentato in Fig. 3.6

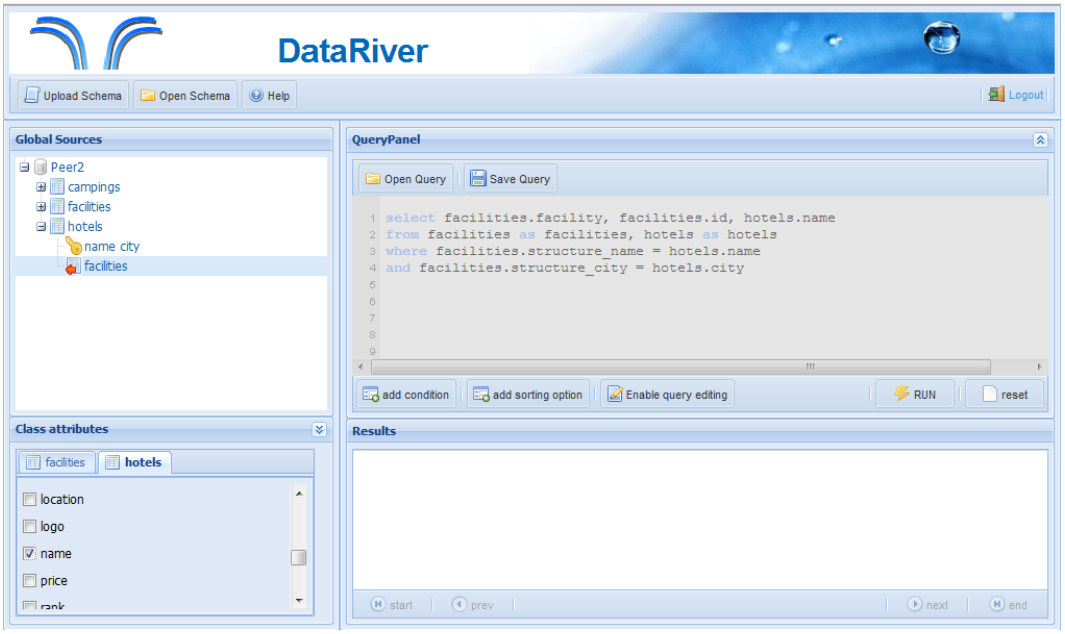


Fig. 3.6: Esempio di Join tra Classi Globali

Dopo avere selezionato classi e attributi, l'utente, se vuole, può aggiungere delle condizioni: premendo il bottone corrispondente (*add condition* per condizioni sugli attributi, *add sorting option* per condizioni di ordinamento) comparirà una finestra con campi di testo da compilare e un menu a tendina tra cui scegliere l'attributo su cui porre la condizione (Fig. 3.8). Non c'è limite al numero di condizioni che si possono aggiungere.

L'editor viene infine aggiornato con le condizioni inserite. A questo punto la parte semi-automatica del processo di composizione della query finisce e

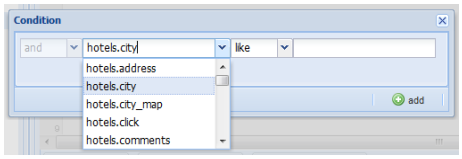


Fig. 3.7: Add condition panel

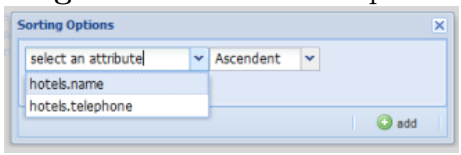


Fig. 3.8: Add sorting options panel

l'utente può decidere se eseguire la query così composta o modificarla manualmente nel caso voglia aggiungere altri dettagli (premendo il bottone *enable Query Editing* l'editor viene abilitato alla scrittura).

3.5 Presentazione dei risultati

Una volta eseguita una query, i risultati vengono riportati nel pannello *Results*, paginati cinquanta alla volta; il pannello dispone di pulsanti per scorrere le pagine dei risultati. Per una migliore visualizzazione sono stati implementate due opzioni distinte:

- la prima opzione è quella di collassare il *Query Panel* premendo l'apposito pulsante (Fig. 3.9); in questo modo il pannello dei risultati si estende in altezza occupando tutto lo spazio disponibile;
- la seconda opzione è fare apparire una nuova finestra, in cui vengono elencati in modo ordinato i nomi degli attributi e il loro valore (Fig. 3.10); basta un click sulla riga di interesse tra i risultati proposti per vedere comparire la finestra relativa. Questa seconda opzione è utile nel caso gli attributi richiesti nella query siano in numero elevato, comportando un elevato numero di colonne nella tabella dei riultati e quindi maggiore difficoltà di lettura.

3.6 Salvataggio della sessione di lavoro

In qualsiasi momento l'utente può decidere di salvare la query che sta scrivendo, semplicemente premendo il pulsante *save query* e inserendo il nome

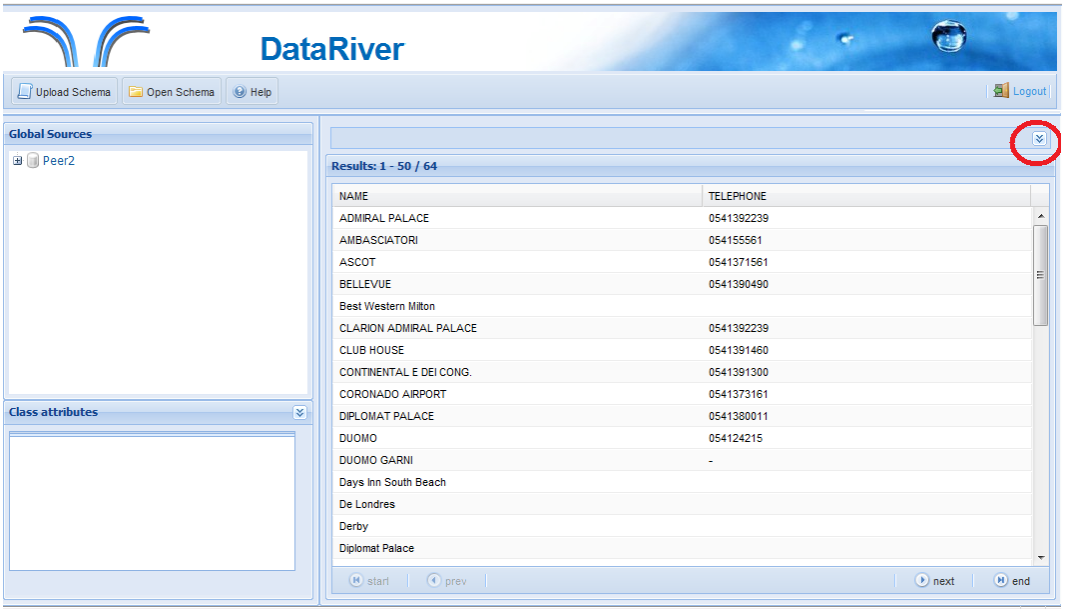


Fig. 3.9: Il pannello *Results* espanso per una migliore visualizzazione dei risultati

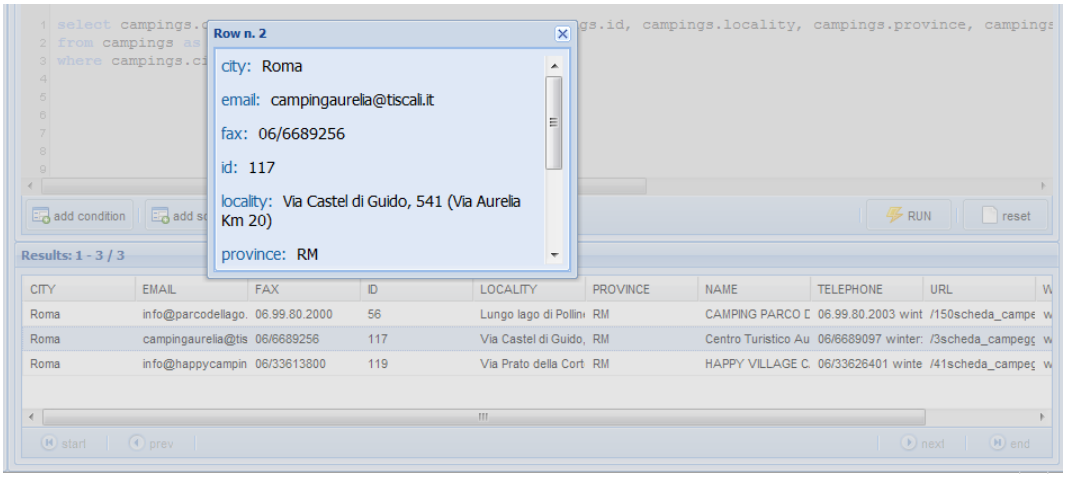


Fig. 3.10: Elenco di attributi e relativi valori del risultato nr. 2

con cui la vuole salvare. L'applicazione provvederà a memorizzarla in un database, insieme allo schema su cui la query viene eseguita. Le query salvate dall'utente sono riportate in una finestra che si apre col pulsante *open query*, come mostrato in Fig. 3.11.

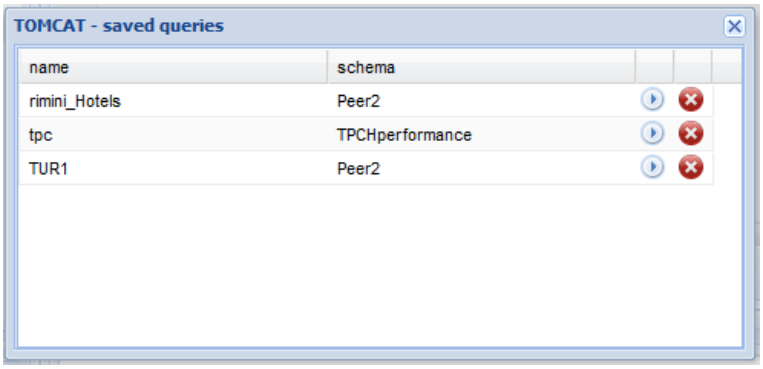


Fig. 3.11: Elenco delle queries salvate dall'utente Tomcat

La finestra contiene una tabella con i campi *name* e *schema*: il primo è il nome della query, il secondo è lo Schema Globale su cui la query va eseguita. In ogni record della tabella sono presenti due pulsanti:

- *play*, che permette di ripristinare l'ambiente su cui eseguire quella query: la pagina viene ricaricata, lo Schema Globale relativo alla query viene caricato nell'albero delle sorgenti e viene settato come schema corrente per il QueryManager, mentre la query in esame viene riportata nell'editor, abilitato in modalità scrittura per eventuali modifiche;
- *delete*, per eliminare una query precedentemente salvata; se l'eliminazione dal DataBase avviene con successo la query scompare dalla tabella, in caso contrario l'utente viene avvisato con un messaggio di errore.

Capitolo 4

Interfaccia web del QueryManager di MOMIS: implementazione

4.1 Strumenti

L'interfaccia è stata realizzata durante un periodo di tirocinio di sei mesi presso Quix S.r.l.

Per realizzare l'applicazione si è fatto uso dei seguenti strumenti:

- Ambiente di sviluppo: Eclipse Java EE IDE
- Linguaggio di programmazione: Java (software: Java Platform, Enterprise Edition 5 SDK Update 7)
- RDBMS (Relational DataBase Management System): MySql
- Librerie esterne:
  - Java Persistence Api (JPA) 2.0
  - Spring Framework 2.5.1
  - Struts2 2.1.6
  - Ext-Js 3.0.0

I frameworks utilizzati sono diversi, e ho deciso di dedicare l'Appendice B all'approfondimento di ciascuno di essi.

In ambito web sono frameworks particolarmente diffusi ed efficaci e, nonostante io non li conoscessi all'inizio del mio lavoro, sono risultati anche di facile apprendimento.

Ognuno di questi framework si è rivelato utile in una parte specifica del progetto:

- JPA per la persistenza dei dati e il salvataggio della sessione;
- Spring per l'istanziamento del QueryManager e per la gestione delle transazioni con il DataBase;
- Struts 2 per la gestione delle funzionalità dell'applicazione lato server;
- Javascript,Ext-js e Ajax (che non è un framework ma una tecnologia, come spiegato nell'appendice B) per la grafica dell'interfaccia e la comunicazione sincrona e asincrona con il server.

Nei prossimi capitoli descriverò l'implementazione generale dell'interfaccia e analizzerò nel dettaglio l'implementazione delle funzionalità più importanti.

4.2 Descrizione generale

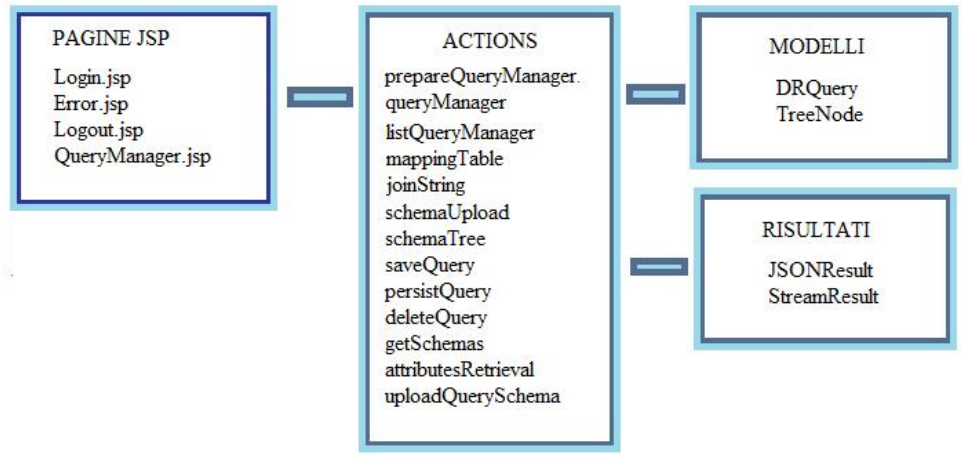


Fig. 4.1: Schema dei componenti del progetto web

Il progetto dell'interfaccia (Fig. 4.1) si compone di:

- quattro pagine JSP: una per il login, una per il logout, una per eventuali errori al momento dell'autenticazione e la pagina principale dell'interfaccia, QueryManager.jsp che contiene tutto il codice javascript e Ajax per l'interazione con il server;

- tre package Java: uno che contiene le Action di Struts2, uno per i Modelli (DRQuery per le query e TreeNode per i nodi dell'albero delle sorgenti) e uno contenente i possibili risultati delle Action (JSONResult e StreamResult, il primo per gestire oggetti di tipo JSON, il secondo per fare in modo che la Action ritorni uno stream di caratteri che sarà poi riportato in pagina);
- immagini e css per rendere l'interfaccia piacevole dal punto di vista grafico;
- file xml di configurazione per i framework utilizzati.

Le actions costruite appositamente per ogni funzionalità dell'applicazione sono definite nella tabella seguente :

Action	Obiettivo	Funzionamento
prepareQueryManager	inizializza alcuni attributi prima dell'esecuzione della action queryManager	inizializza i valori di alcuni attributi tra cui la la lista che contiene i risultati dell'esecuzione di ogni query
queryManager	Si occupa dell'esecuzione delle query	recupera il QueryManager istanziato da Spring ed esegue la query, salvandone i risultati in una lista

Action	Obiettivo	Funzionamento
listQueryManager	si occupa del reperimento dei risultati della query eseguita, riportandoli in pagina	converte la lista dei risultati in un oggetto JSON che verrà poi usato per popolare la griglia del pannello Results
mappingTable	compone la Mapping Table di una determinata Classe Globale	recupera dallo Schema Globale i dati necessari e li inserisce in un oggetto JSON per restutirli in pagina
joinString	compone la condizione di Join tra due Classi Globali	recupera dallo Schema Globale le Foreign Keys e compone la condizione di Join che verrà poi scritta nell'editor
schemaUpload	si occupa dell'upload di un nuovo Schema Globale	salva il nuovo Schema Globale e lo imposta come schema corrente di riferimento per il QueryManager

Action	Obiettivo	Funzionamento
schemaTree	compone l'albero delle sorgenti	recupera dallo Schema Globale i dati necessari e li inserisce in un oggetto JSON per restituirli in pagina
saveQuery	recupera le query precedentemente salvate dall'utente	si connette al DB e recupera le query salvate dall'utente. Restituisce la lista di query in un oggetto JSON
persistQuery	salva la query	si connette al DB memorizza la query insieme al nome dello Schema Globale su cui va eseguita
deleteQuery	elimina la query	ricerca la query nel DB e la elimina
getSchemas	recupera gli Schemi Globali memorizzati	si connette al DB e recupera i nomi di tutti gli Schemi Globali salvati

Action	Obiettivo	Funzionamento
attributesRetrieval	recupera tutti gli attributi di una Classe Globale	recupera dallo Schema Globale gli attributi e li inserisce in un oggetto JSON per restutirli in pagina
uploadQuerySchema	imposta come Schema Globale corrente lo schema correlato alla query che si vuole eseguire	recupera lo Schema Globale e lo imposta come schema corrente di riferimento per il QueryManager

Le action, per alcuni tipi di operazioni, necessitano di modelli con cui interagire.

Nell'ambito di questo progetto i modelli che sono risultati necessari sono:

- *DRQuery* è la modellazione della query che verrà memorizzata nel DataBase; ha cinque attributi:
  - *id* (identificativo della query)
  - *name* (nome assegnato alla query al momento del salvataggio)
  - *query* (il testo della query)
  - *schemaName* (il nome dello Schema Globale su cui la query va eseguita)
  - *userId* (identificativo dell'utente)

  Questi attributi costituiranno le colonne della tabella del DataBase; questo è reso possibile da JPA che permette la modellazione del DataBase direttamente tramite classi Java (vedi Appendice B).

- TreeNode rappresenta invece un nodo dell'albero delle sorgenti; ha sei attributi:
  - *id* (identificativo del nodo)

– *text* (il testo che verrà visualizzato nell'albero)
– *leaf* (è un valore booleano che identifica se si tratta di un nodo foglia)
– *qtip* (il quickTip che sarà visibile fermando il cursore sul nodo)
– *children* (la lista dei figli del nodo, se non è un nodo foglia)
– *icon* (il nome dell'icona da assegnare al nodo)

I nodi vengono concatenati in una lista, sulla base della quale verrà costruito l'albero delle sorgenti.

Nei capitoli successivi saranno analizzati nel dettaglio i processi di esecuzione e di memorizzazione delle queries.

4.3 Esecuzione di una query e reperimento dei risultati

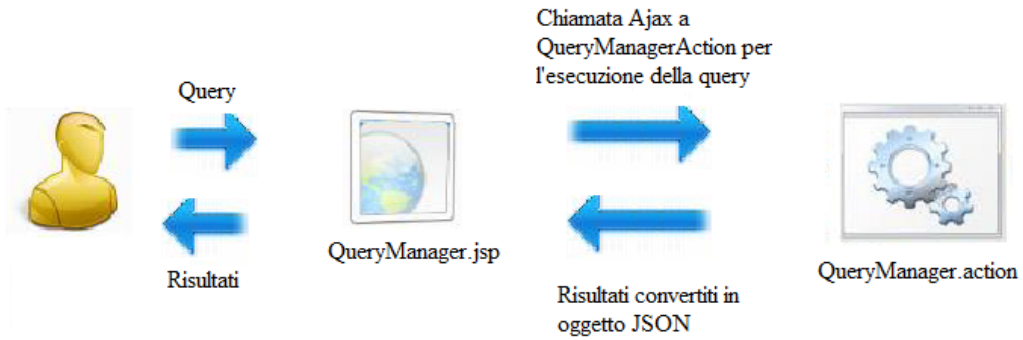


Fig. 4.2: Il processo di esecuzione di una query

Il processo di esecuzione di una query e di recupero e presentazione dei risultati (schematizzato in Fig. 4.2) è piuttosto lungo e complesso, ma provvederò a elencarne le parti principali.

Innanzitutto, la query scritta nel *queryEditor* viene recuperata nel momento in cui l'utente preme il pulsante *Run* e inoltrata alla Action *QueryManager* tramite chiamata Ajax; la action provvede ad eseguire la query e, in caso di successo, il sistema esegue un'ulteriore chiamata Ajax alla action

listQueryManager che si occupa di trasformare la lista dei risultati in un oggetto JSON, che andrà a popolare la tabella Results dell'interfaccia.

La funzione javascript richiamata al momento della pressione del pulsante *Run* è la seguente:

```
//esegue la query inserita e riporta i primi 50 risultati
//nella tabella
function getQueryResults(query) {

//...

var queryString = escape(query);

//inoltro alla queryManager Action la query da eseguire
Ext.Ajax.request( {

url : '<s:url action="queryManager" escapeAmp="false"
         includeParams="none"><s:paramname="task">execute
          </s:param></s:url>&query='+queryString,

success : function(response) {

//se l'esecuzione della query ha successo recupero i primi
//50 risultati richiamando la action listQueryManager
Ext.Ajax.request( {
url : '<s:url action="listQueryManager" escapeAmp="false"
           includeParams="none">
     <s:param name="task">listGridSettings</s:param></s:url>
        &start='+start+'&end='+end,

success : function(response) {

//...(recupero l'oggetto JSON e lo riporto in pagina)
}});
}});
};
```

La Action *QueryManager* provvede a fare eseguire la query dal Query-Manager di MOMIS e a recuperarne i risultati, salvandoli in una lista:

```
public class QueryManagerAction extends ActionSupport implements
JSONResultAware, StreamResultAware {

//faccio iniettare da Spring l'istanza del QueryManager di MOMIS
@Resource(name = "queryManager")
private QueryManager queryManager;

//..(definizione e inizializzazione degli attributi della action)

public String execute() throws Exception {

//...

resultSetColumnNames = new ArrayList<String>();
resultSetData = new ArrayList<List<Object>>();
ResultSet rs = null;
try{
rs = queryManager.executeQuery(query);}
catch(QueryManagerException e){
rs = null;
error=e.getMessage()+e.getCause();
return ERROR;
}

// recupero anche i meta-dati relativi ai risultati
// (informazioni su nomi,tipo e numero totale delle colonne)
ResultSetMetaData rsmd = rs.getMetaData();
int numColonne = rsmd.getColumnCount();
for (int i = 1; i <= numColonne; i++)
resultSetColumnNames.add((String) rsmd.getColumnName(i));

//... (salvo ogni record nella lista resultSetData)

rs.close();

return SUCCESS;
}}
```

La action *listQueryManager* si occupa invece della trasformazione della lista dei risultati in un oggetto JSON, che costituirà lo store della tabella dei

risultati.

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati in applicazioni client-server.

La semplicità di JSON ne ha decretato un rapido utilizzo specialmente nella programmazione in AJAX. Ogni oggetto JSON è costituito da coppie nome/valore contenute tra parentesi graffe, e ciò che lo rende particolarmente potente è che siccome il valore può essere qualunque tipo di dati, si possono memorizzare altri array ed altri oggetti, annidati profondamente secondo necessità. Questo si è rivelato molto utile nella fase di costruzione dell'albero delle sorgenti (ogni oggetto JSON rappresentante un nodo aveva al suo interno un'array di oggetti JSON rappresentanti i figli, e in alcuni casi i figli avevano altri figli..). Ritornando alla fase di recupero dei risultati di una query, riporto di seguito un'esempio di oggetto JSON costruito dalla Action *listQueryManager*:

```
[
[{id:'NAME',header: 'NAME', width: 100, sortable: false,
                                          dataIndex: 'NAME'},
{id:'ID',header: 'ID', width: 100, sortable: false,
                                          dataIndex: 'ID'},
{id:'CITY',header: 'CITY', width: 100, sortable: false,
                                          dataIndex: 'CITY'}],

[{name:'NAME'},{name:'ID'},{name:'CITY'}],

[['CAMPING PARCO DEL LAGO','56','Roma'],
['Centro Turistico Aurelia Club','117','Roma'],
['HAPPY VILLAGE CAMPING','119','Roma']],

3
]
```

L'oggetto JSON, in questo caso, è composto di 4 parti differenti:

```
[
[definizione_colonne],
[nomi_colonne],
[risultati_query],
numero_risultati
]
```

L'oggetto JSON viene decodificato all'interno della funzione di successo della chiamata AJAX alla action *listQueryManager*:

```
success : function(response) {

//decodifico l'oggetto JSON resitutito dalla Action,
//che contiene i risultati della query
gridSettings = Ext.util.JSON.decode(response.responseText);

columns = gridSettings[0];    //definizione_colonne
fields = gridSettings[1];     //nomi_colonne
data = gridSettings[2];       //risultati_query
totRecord = gridSettings[3];  //numero_risultati

//costruisco il nuovo Store per la tabella dei risultati
var newResultsStore = new Ext.data.ArrayStore({
fields: fields,
data : data
});
   var newColMod = new Ext.grid.ColumnModel(columns);

//riconfiguro la tabella del pannello Results con i
//nuovi risultati
resultsGrid.reconfigure(newResultsStore, newColMod);

//...

}
```

Il risultato finale è visibile in Fig. 4.3.

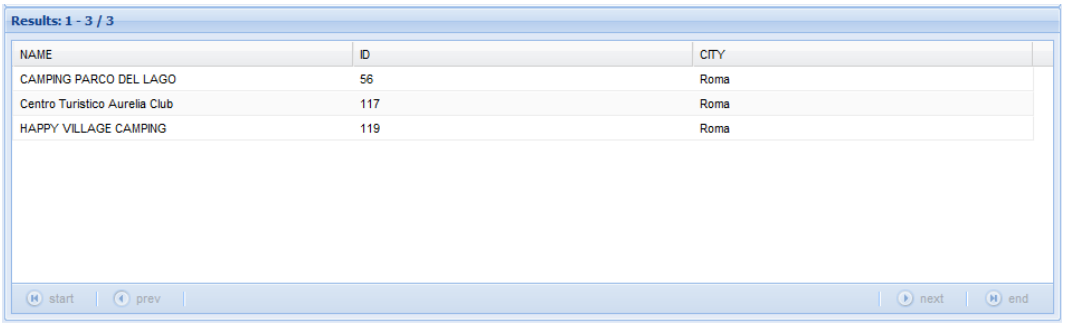


Fig. 4.3: Risultati di una query inseriti nella tabella Results

4.4 Salvataggio di una query

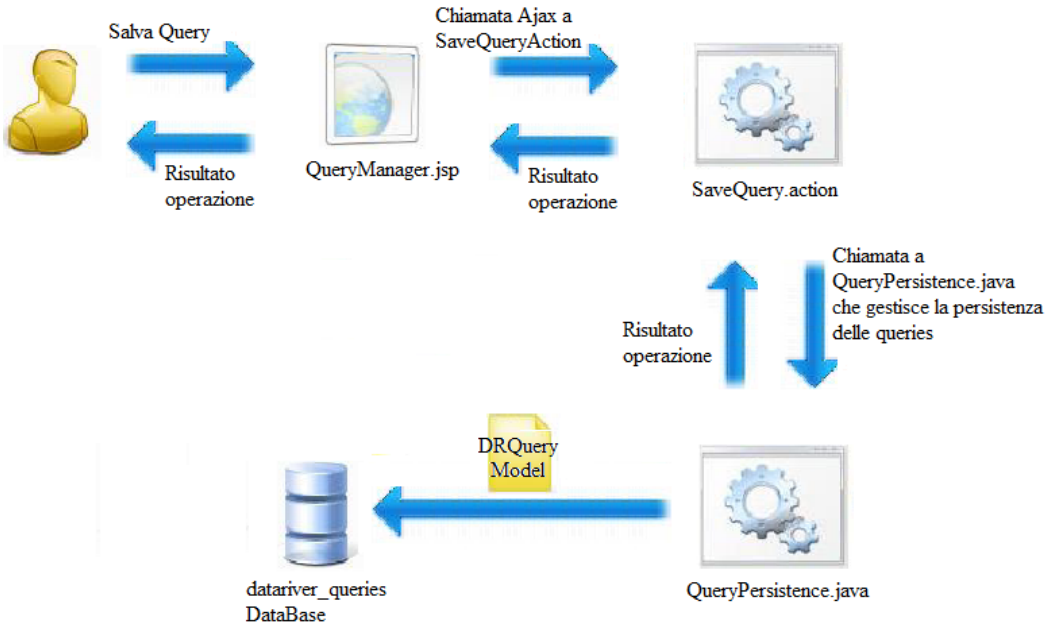


Fig. 4.4: Il processo di salvataggio di una query

Il processo di salvataggio di una query è schematizzato in Fig. 4.4. La funzione javascript richiamata dopo che l'utente ha inserito il nome con cui salvare la query è la seguente:

```
  //salva la query eseguita
function persist(query,queryname) {
Ext.Ajax.request( {
url : '<s:url action="persistQuery" escapeAmp="false"
            includeParams="none"></s:url>?task=persistQuery
                      &query=' + query+'&name='+queryname,
success : function() {
Ext.Msg.alert('The query has been correctly saved.');},
failure : function() {
Ext.Msg.alert('Error in persisting query, try again.');
}});
};
```

La funzione javascript esegue una chiamata Ajax alla Action *persistQuery* che si occupa del salvataggio della query:

```
public String persistQuery() throws Exception{
request = ServletActionContext.getRequest();
user = request.getUserPrincipal().getName();
  queryPersistence.insertQuery(user,query,name);
return "success";
}
```

Questa Action crea un'istanza di un oggetto Java chiamato *queryPersistence* che implementa i metodi per la persistenza e la gestione delle transazioni.

Il metodo *insertQuery* della classe *QueryPersistence* recupera il nome dello schema corrente (che verrà caricato ogni volta che si richieda l'esecuzione della query salvata), crea un'istanza del modello DRQuery e ne setta i parametri; provvede poi al salvataggio della query nel DataBase.

```
@Transactional(readOnly=false)
public void insertQuery(String user,String query,String name)
                                                  throws Exception{

String fullFileName = "/DRschemas/currentschema.xml";
Schema schema = QueryManagerUtils.getOdli3Schema(fullFileName);

newQuery=new DRQuery();
newQuery.setQuery(query);
newQuery.setUserId(user);
newQuery.setName(name);
newQuery.setSchemaName(schema.getName());

//salvo la query
em.persist(newQuery);
return;
}
```

L'oggetto *em* è un'istanza dell' EntityManager iniettato da Spring nella classe; l'Entity Manager fornisce metodi per iniziare e finire transazioni, memorizzare, aggiornare, cancellare e trovare entità nel *persistence context* (contesto di persistenza, l'insieme di entità che saranno mappate nel DataBase). L'entità in questione è l'oggetto *DRQuery*, che modella i record della tabella che viene creata nel DataBase:

```
@Entity
@Table(name="QUERY")
public class DRQuery {
```

```
@Id
@Column(name="QUERY_ID")
@GeneratedValue(strategy=GenerationType.IDENTITY)
public int id;

@Column(length=1000)
public String query;

@Column(name="USER_ID", length=256)
public String userId;

@Column(length=256)
public String name;

@Column(length=256)
public String schemaName;

public DRQuery() {}

//...

}
```

La forza di JPA è proprio questa: permette di definire direttamente dal codice Java la struttura del DataBase di cui necessita l'applicazione.

La classe DRQuery viene annotata come entità (@Entity) e ogni suo attributo costituisce una colonna della tabella. L'attributo *id* (marcato con l'annotazione @Id) è la chiave primaria della tabella, ed è un valore intero autogenerato. Le impostazioni di configurazione del contesto di persistenza si trovano nel file persistence.xml:

```
<persistence-unit name="datariver_queries"
                  transaction-type="RESOURCE_LOCAL">
<provider>
org.eclipse.persistence.jpa.PersistenceProvider
</provider>
<class>it.unimo.datariver.model.DRQuery</class>

//... (parametri per la connessione al DataBase)

</persistence-unit>
```

Il DataBase creato si chiama quindi "datariver_queries" ed è costuito da un'unica tabella (Fig. 4.5), definita nella classe *DRQuery*.

QUERY_ID	SCHEMANAME	QUERY	NAME	USER_ID
1	TPCHperformance	select partsupp.ps_suppkey, partsupp.ps_partkey from part...	tpc	tomcat
3	Peer2	select facilities.facility, hotels.name, hotels.city from facilities ...	TUR1	tomcat
4	Peer2	select hotels.name, hotels.telephone from hotels as hotels w...	rimini_Hotels	tomcat
6	Peer2	select campings.city, campings.email, campings.fax from ca...	campings	tomcat
7	Peer2	SELECT facilities.facility, facilities.id FROM facilities AS facili...	fac2	tomcat

Fig. 4.5: La tabella "Query"

Conclusioni

Durante la stesura dell'elaborato sono state affrontate diverse tematiche attuali di interesse.

Innanzitutto è stato descritto il paradigma di distribuzione Open Source, fornendone una visione d'insieme ed elencando vantaggi e svantaggi derivanti all'utilizzo di questo tipo di distribuzione software.

Il software Open Source si sta rivelando una soluzione sempre più ottimale sia per l'utente medio che per l'utente specializzato; inoltre la disponibilità del codice porta al continuo miglioramento del software e alla collaborazione tra sviluppatori di ogni parte del mondo.

In seguito, è stata affrontata la tematica della Data Integration, con rilievo alle metodologie di interrogazione di Schemi Globali.Il successo che avrà MOMIS una volta uscita la versione Open Source non è possibile predirlo, ma la sua efficacia nell'ambito dell'integrazione dei dati è senz'altro indubbia.

Infine è stata presentata l'interfaccia web del QueryManager, insieme a una descrizione dettagliata della sua implementazione.

Pensando a un suo possibile sviluppo futuro, l'interfaccia potrebbe essere ottimizzata e resa ancora più funzionale, aggiungendo in particolare:

- la possibilità di visualizzare in un'immagine la struttura dello Schema Globale nella sua totalità o in parti richieste dall'utente;
- la possibilità di visualizzare le Funzioni di Risoluzione degli attributi globali e le Funzioni di Trasformazione degli attributi locali mappati in attributi globali.

Infine il Manuale di MOMIS potrebbe essere trasposto in documento HTML interattivo e incluso nel sistema stesso, nella pagina di *welcome*.

Bibliografia

1. "Struts 2 in Action", Donald Brown, Chad Michael Davis and Scott Stanlick, 2008
2. "Spring in Action", Craig Walls and Ryan Breidenbach, 2005
3. "Producing Open Source Software", Karl Fogel, 2005
4. "Java Persistence with Hibernate", Christian Bauer and Gavin King, 2006
5. "Open Source come modello di Business per le PMI: analisi critica e casi di studio.", S. Bergamaschi, F. Nigro, L. Po, M. Vincini, 2008
6. http://wiki.eclipse.org/Introduction_to_Java_Persistence_API(ELUG)
7. http://java.sun.com/developer/technicalArticles/J2SE/Desktop/persistenceapi/
8. http://www.itacatech.it/integrazionedati.asp
9. http://it.wikipedia.org/wiki/Open_source
10. http://it.wikipedia.org/wiki/Licenza_Opensource
11. http://www.ricercaitaliana.it/prin/dettaglio_completo_prin-2005121515.htm
12. http://it.wikipedia.org/wiki/JavaScript
13. http://javascript.html.it/articoli/leggi/3233/ext-js-il-desktop-sbarca-sui-browser/
14. http://it.wikipedia.org/wiki/AJAX
15. http://www.dbgroup.unimo.it/Momis/
16. http://www.dbgroup.unimo.it/Momis/prototipo/2004MomisManual_ISWC-MOMIS.pdf

Appendice A

MOMIS Tutorial

A.1 Overview

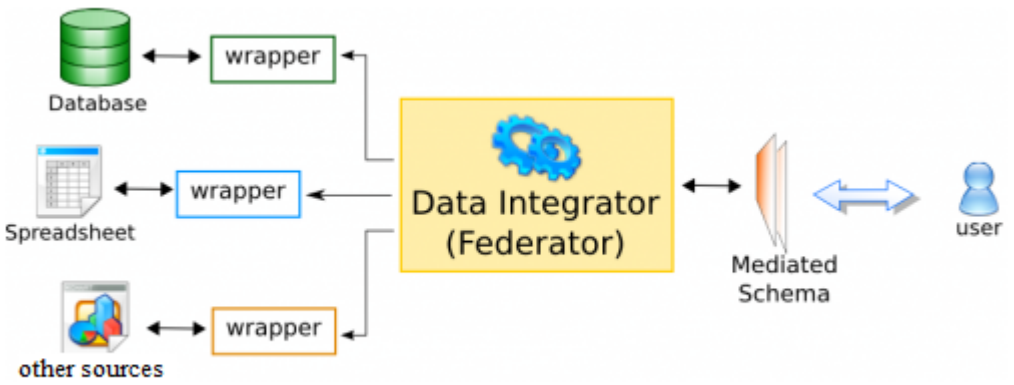


Fig. A.1: Data Integration Process

MOMIS (Mediator envirOnment for Multiple Information Sources) is a framework to perform information extraction and integration from both structured and semistructured data sources.

An object-oriented language, with an underlying Description Logic, called ODL-I3, derived from the standard ODMG is introduced for information extraction. Information integration is then performed in a semi-automatic way, by exploiting the knowledge in a Common Thesaurus (defined by the framework) and ODL-I3 descriptions of source schemas with a combination of clustering techniques and Description Logics.

This integration process (Fig. A.1) gives rise to a virtual integrated view of the underlying sources (the Global Schema) for which mapping rules and integrity constraints are specified to handle heterogeneity.

The MOMIS system, based on a conventional wrapper/mediator architecture, provides methods and open tools for data management in Internet-based information systems.

Sources integration is based on the individuation of an ontology shared by each source; the ontology is represented as a set of terminological relationships called Common Thesaurus. The MOMIS' GUI (Graphic User Interface) supports the designer in the overall integration process, schematically presented in Fig. A.2. The figure shows the local schemas' generation, where local schemas are annotated according to the lexical ontology WordNet, the Common Thesaurus generation, and finally the global classes. In particular, these ones are connected by means of mapping tables to the local schemas and are (semiautomatically) annotated according to WordNet.

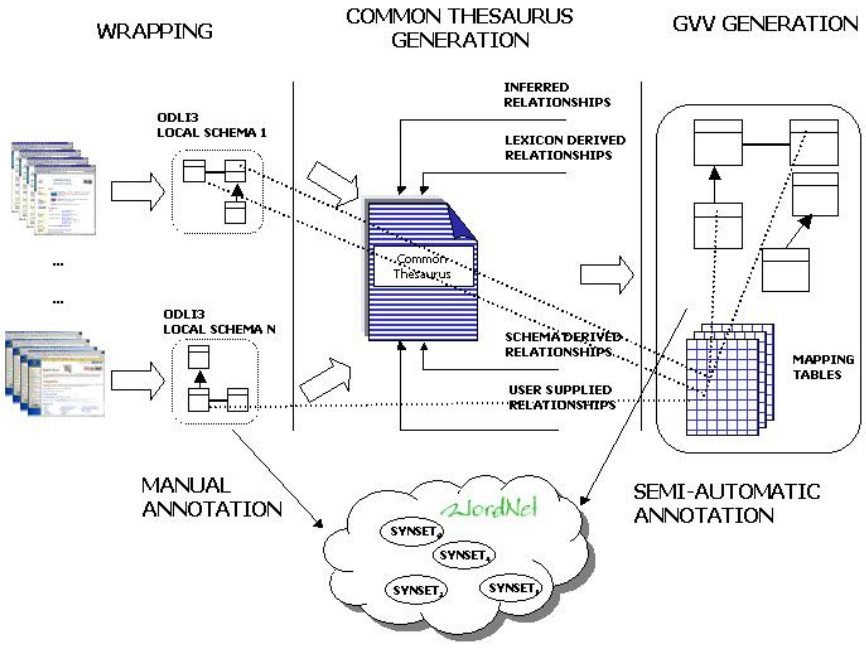


Fig. A.2: Overview of the ontology-generation process

The GUI of MOMIS is a sequence of panels (each panel performs a phase of the integration process):

1. *Data Sources acquisition:* The user is asked to upload the local sources to be mapped in the Global Schema. A wrapper logically converts the

source data structure into the ODLI3 model. Wrappers are the focal point for managing the diversity of data sources.

2. *Manual annotation of a local source with WordNet:* For each element of the local schema, the user has to manually choose the appropriate meaning in the WordNet lexical database.

3. *Common Thesaurus Generation:* MOMIS constructs a Common Thesaurus describing intra and inter-schema knowledge in the form of SYN (Synonymous terms), BT (borrower term), NT (narrower term), and RT (related terms) relationships. The Common Thesaurus is incrementally built by adding *Schema-derived relationships* (automatic extraction of intra-schema relationships for each schema separately), *Lexicon-derived relationships* (inter-schema lexical relationships derived by annotated sources and WordNet interaction), *Designer-supplied relationships* (specific domain knowledge capture) and *Inferred relationships* (via Description Logics equivalence and subsumption computation).

4. *Global Schema Generation:* The MOMIS methodology allows us to identify similar ODLI3 classes, that is, classes that describe the same or semantically related concepts in different sources.

The Global Schema is the main CORBA object to access the MOMIS integration services. A Global Schema object contains all information about integration for querying the resulting global schema by a query manager object, allowing a user to pose a query and to receive a single unified answer.

The Query Manager CORBA object performs query processing and optimisation. It generates the OQL_I3 queries for wrappers, starting from a global OQL_I3 query formulated by the user on the global schema. Using Description Logics techniques, the component automatically generates the translation of the global query into sub-queries for each involved local source, sends such sub-queries to the sources, collects the answers and computes a unified answer.

A.2 Starting Momis

The System starts presenting a Welcome Page (Fig. A.3), in wich you can find some information about Data Integration and Momis Application.

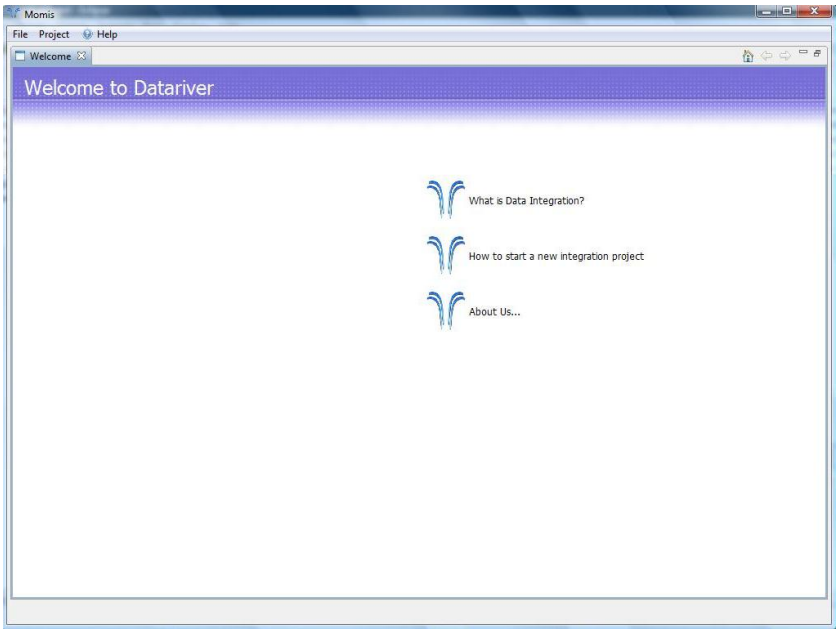


Fig. A.3: Welcome Page

If you close the Welcome page, you will see the MOMIS GUI, composed of three panels (Fig. A.4):

1. *Source Explorer*, wich will contain all the local sources you load;

2. *Global Schema Explorer*, wich will contain all the Global Schema created;

3. *Editor Panel*, where you will be able to edit or create a Global Schema.

The menu of the application is composed by two main voices:

1. *Project* wich enables the upload of local sources and permits you to create a new integration project, upload an existing one or save the current project.

2. *File* wich permits you to create a new global schema, upload an existing one or save the current global schema.

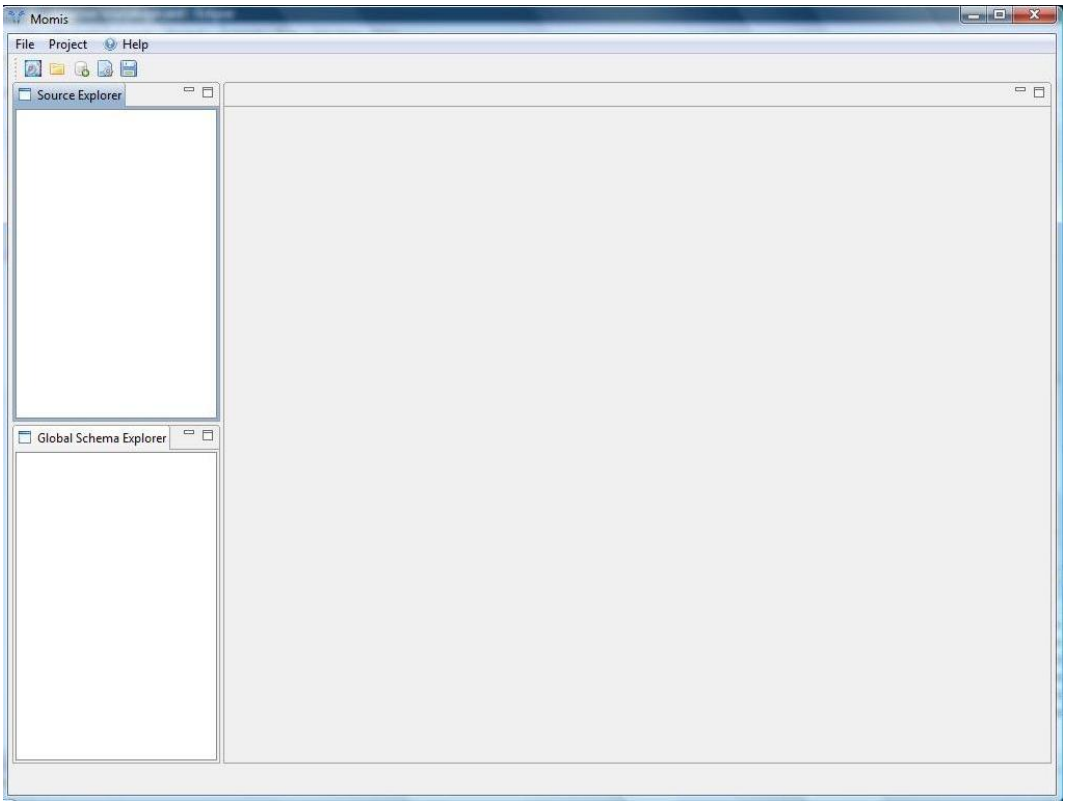


Fig. A.4: MOMIS GUI

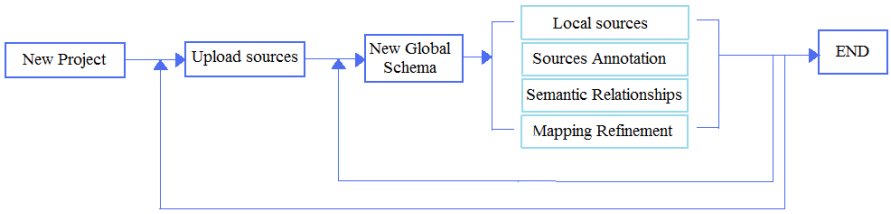


Fig. A.5: Project generation process

The steps to be followed are shown in Fig. A.5. First of all, you have to create a new project, then you have to upload the local sources and start the creation of a new Global Schema by editing each of the section displayed (Local sources, Sources Annotation, Semantic Relationships and Mapping Refinement); once completed the Global Schema you have three possible choiches:

1. Launch the Query Manager working on that Global Schema;
2. Start the creation of a new Global Schema (there may be different Global Schemas in the same project);
3. Upload other local sources.

If you click *Project -> New Project* and insert the name, the new Project will be created, and the System will guide you in the Sources Upload Process (see next section).

A.2.1 Acquiring Local Data Sources

The first step after creating a new project is to upload the local data sources to be integrated. A new window is shown (automatically presented by the System after creating a new Project, but available at any time by clicking on the *New local source* button), and you are asked to insert the type of relational local source you want to connect to (by choosing among "MySQL database", "MS SQL Server database", "Oracle database", "JDBC source" and "ODBC source") and the name; then you have to insert the correct parameters for the connection (Fig. A.6):

- Server Domain
- Port
- Username and password
- Source name

You can start the connection by clicking the *Create connection* button and choose the source from the list of all the available sources found.

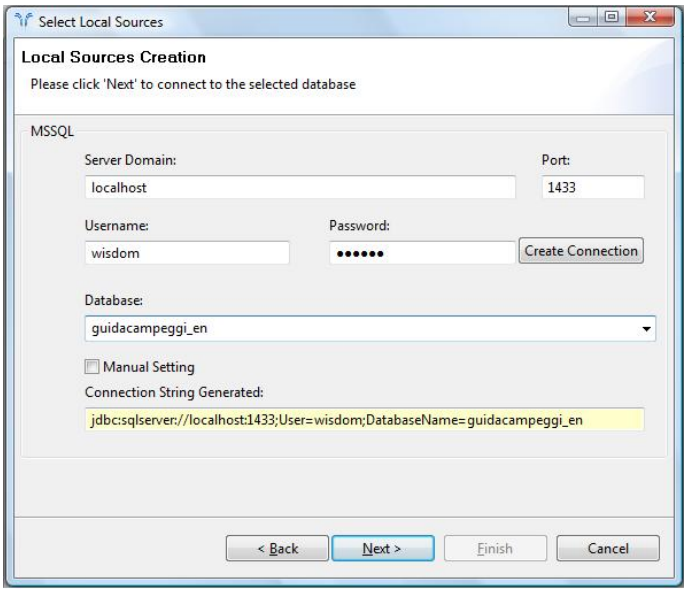


Fig. A.6: Connection parameters for Source Uploading

Click the *Next* button to reach the last section of the source uploading activity: you will see the list of the classes of the uploaded sources, and you are asked to choose the ones you want to add to your project(Fig. A.7).

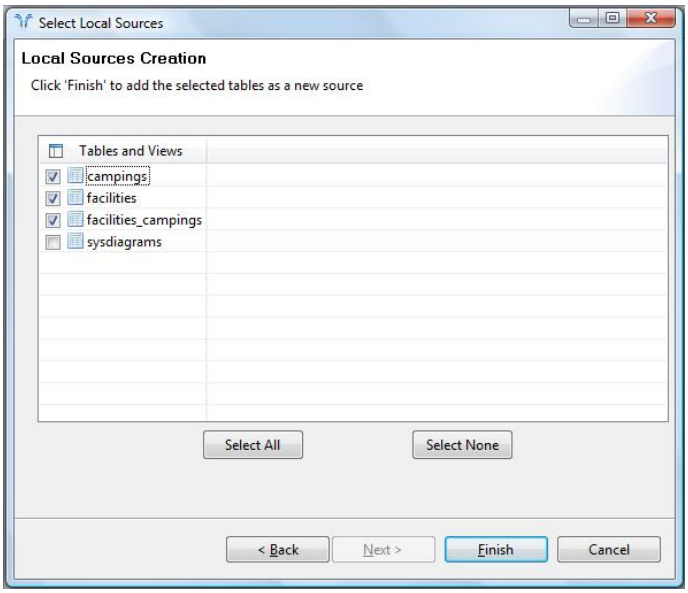


Fig. A.7: Data Sources Classes

Then click on *Finish* to complete the source's upload; the source's classes will appear in the *Project Explorer*. Repeat the process for all the local sources you need to include in your project session.

A.2.2 Create a New Global Schema

When the voice *File -> New Global Schema* is selected, the GUI shows the window in Fig. A.8, where you have to insert the name of the Global Schema you are going to create.

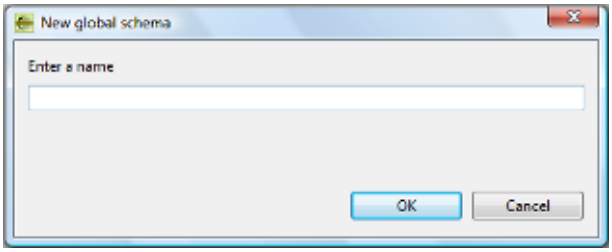


Fig. A.8: Insert a Global Schema Name

The MOMIS system starts visualizing the main page of the Global Schema Editor (Fig. A.9).

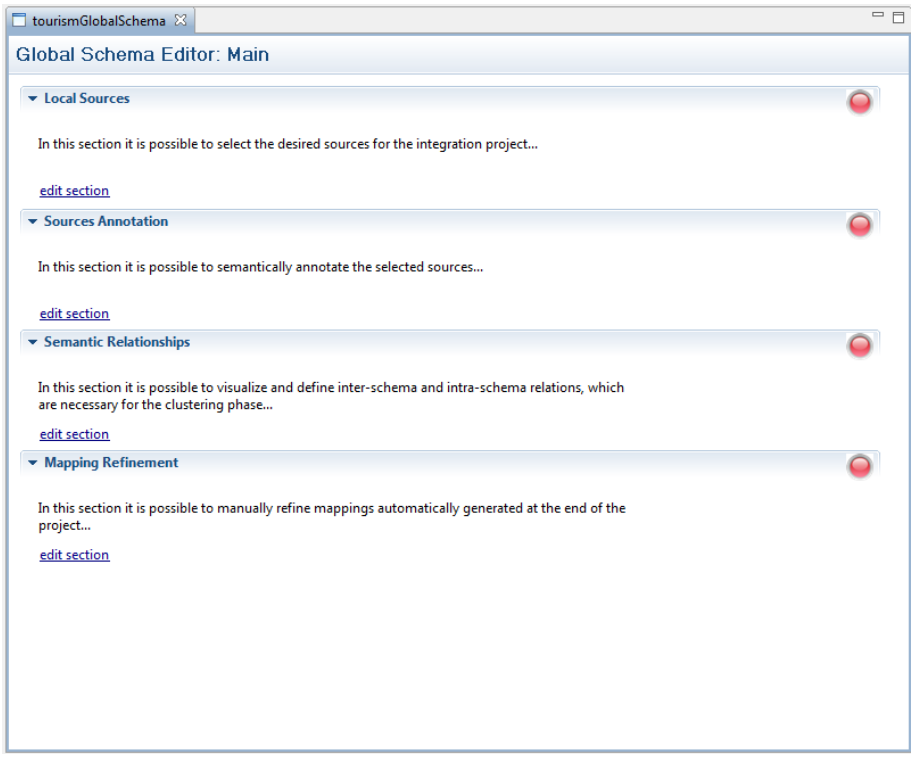


Fig. A.9: Main page of the Global Schema Editor

From here you can edit any of the following sections:

- Local sources
- Annotation
- Semantic Relationships
- Mapping Refinement

You have to complete each section, in order to obtain a correct Global Schema.

In Fig. A.10, we report the icons used by the GUI and their intended meanings.

Fig. A.10: Global Schema Editor: Icons Legend

A.2.3 Selection of Local sources

In this section you have to point out the sources you wanto to include in your Global Schema, choosing them among the sources you uploaded before. Right-click on a source from the *Project Explorer* and click on *Add selected source to the GS*; for each selected source you can see more information in the *Source Details* section, as shown in Fig. A.11.

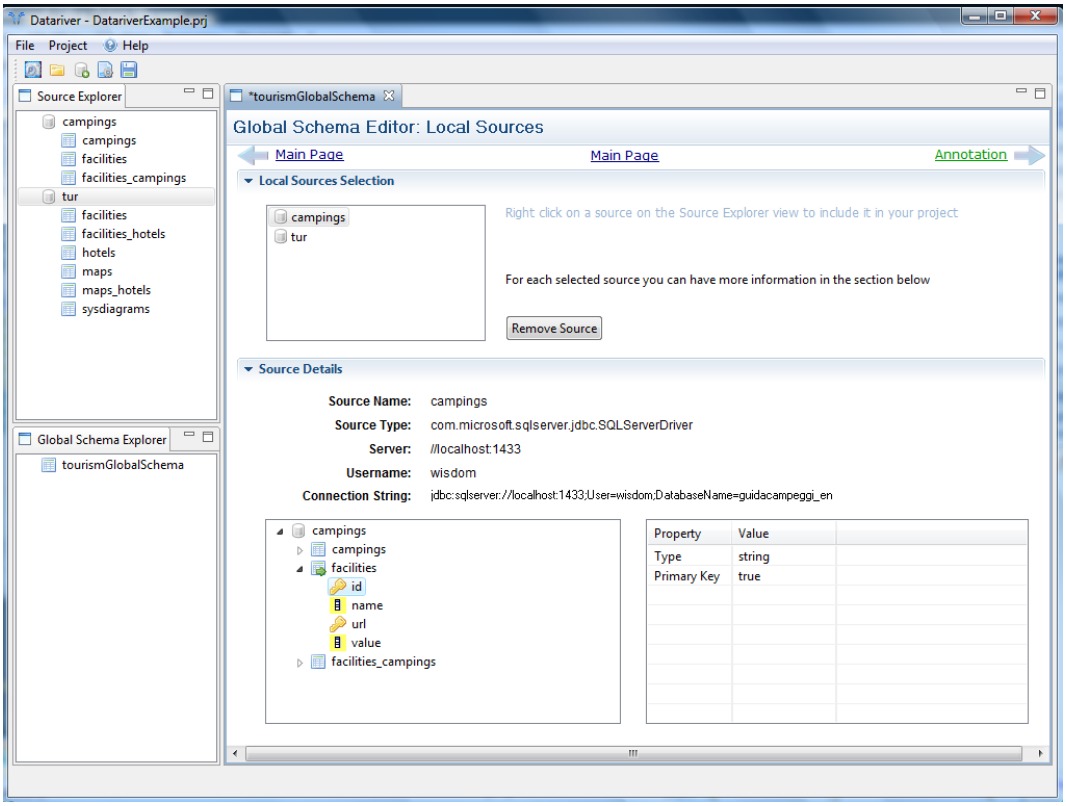


Fig. A.11: Local Sources Selection

A.2.4 Local Sources Annotation with WordNet

Once you have acquired the sources, you have to annotate each label of sources' elements (class and attribute names) with respect to the "WordNet© Lexical Database" (http://www.cogsci.princeton.edu/~wn/), version 3.0.

The annotation phase consists of two steps to be repeated for each class and attribute:

1. Choice of a word form (an English term denoting the meaning of the label);

2. Choice of its meanings (zero, one or more senses for each word form).

Word forms and senses are the ones proposed by the WordNet database.

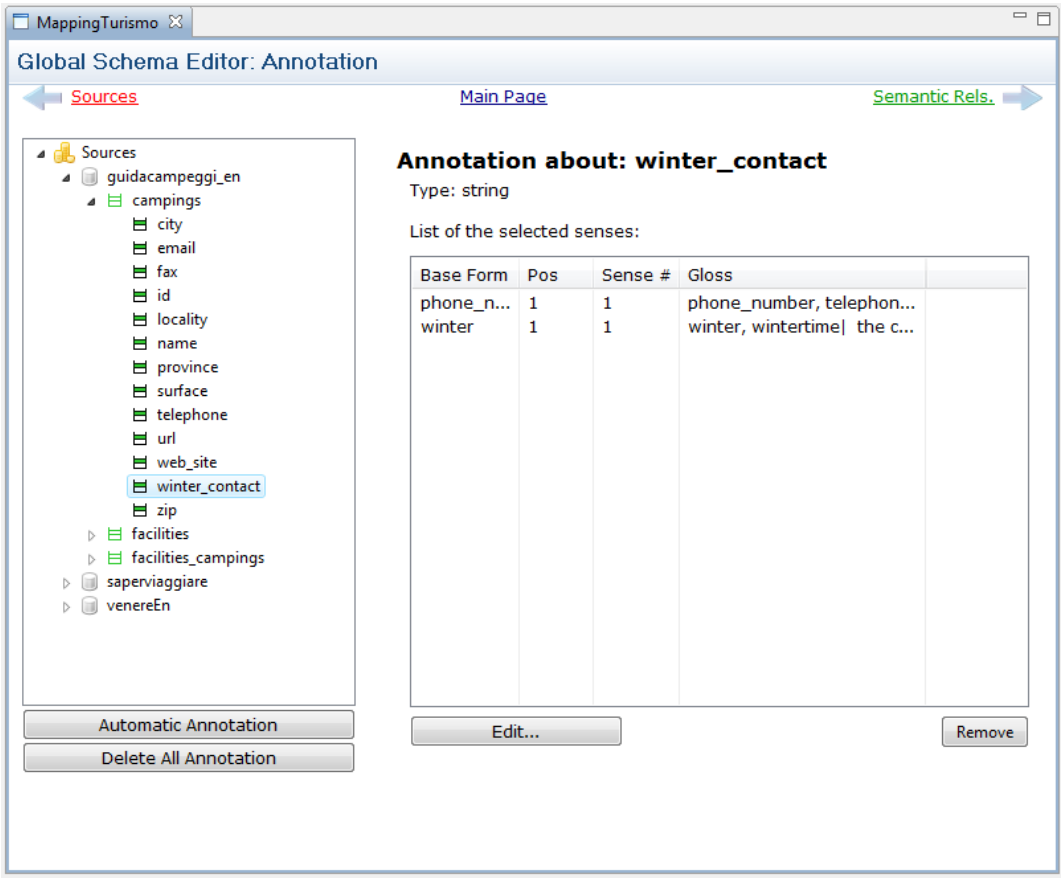


Fig. A.12: Annotation

The GUI shows on the left a tree representing the involved sources and classes(Fig. A.12) and uses colored icons in order to help you to find the labels to be annotated (see Fig. A.13).

	Local Class label	selected Word Form	no chosen sense
	Local Class label	Word Form not found	no chosen sense
	Local Class label	selected Word Form	chosen sense
	Local Class label	ignored label	
	Local Attribute label	selected Word Form	no chosen sense
	Local Attribute label	Word Form not found	no chosen sense
	Local Attribute label	selected Word Form	chosen sense
	Local Attribute label	ignored label	

Fig. A.13: Annotation Icons

You can navigate across the tree and choose the classes/attributes' labels to be annotated by selecting them and press the *Add annotation* button; a new window will appear, where you can find the base form(s) for the label you are annotating and choose among all the possible senses (Fig. A.14).

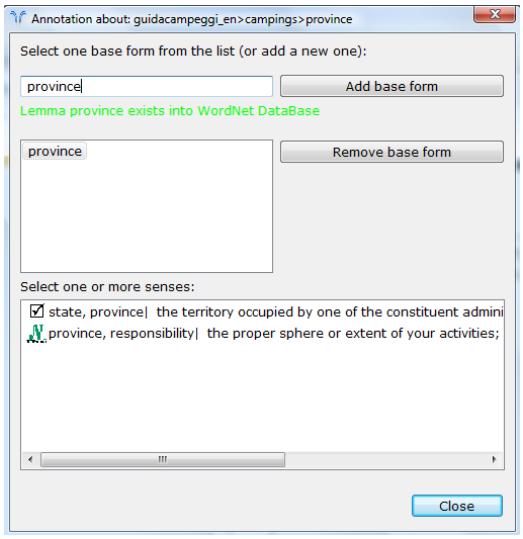


Fig. A.14: Label Annotation

Inside WordNet there are nouns, verbs, adjectives, adverbs, identified by the symbols shown in Fig. A.15.

If no Word Base Form is found, the system shows a message: "ATTENTION: lemma .. doesn't exist into WordNet database"; in this case you can

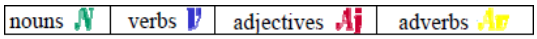


Fig. A.15: WordNet Icons

ignore the label or insert a different word for that label and see if it is found in WordNet. An example is shown in Fig. A.16.

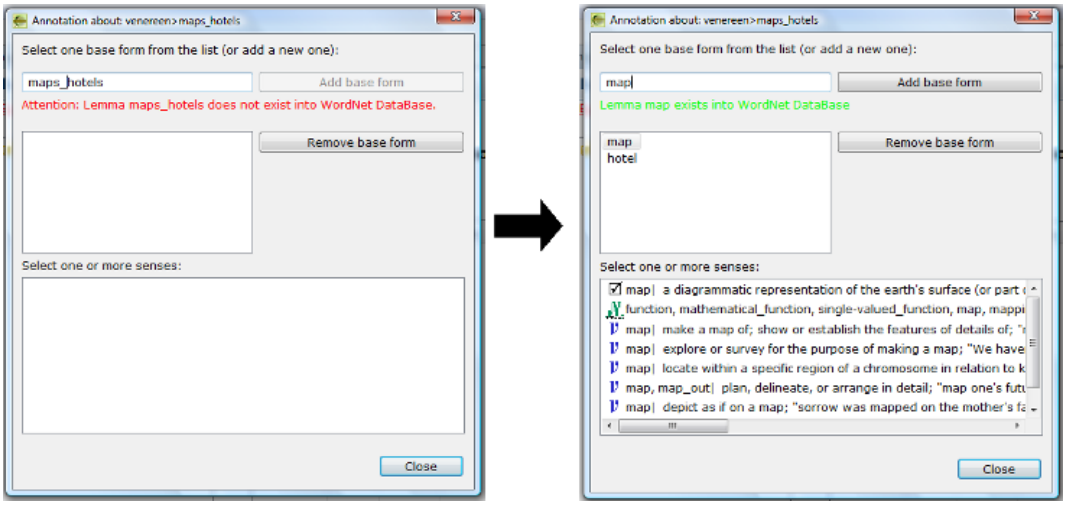


Fig. A.16: Handling "word not existing" problem for the word "maps_hotels"

Annotations may be incomplete, but the more labels you annotate, the more lexical relationships will likely be automatically extracted among labels of different local sources.

The most frequent sense of a label among all the possible ones will be automatically proposed after clicking the *Automatic Annotations* button. This operation has good results only if labels are espressed in English without special characters, compound terms, etc.

A.2.5 Semantic Relationships

Relationships among schemata labels allow for richer and more efficient management of the underlying content. In order to extract and manage the relationships, you have to click on the *Next: Semantic Rels* link at the top of the window (see Fig. A.12).

MOMIS is able to manage different kinds of relationships, classifyable with respect to their kind and their provider.

Kind of relationships:

- SYN (A SYN B means that A and B are synonyms);
- NT (A NT B means that A is more specific then B)
- BT (A BT B means that A is broader then B)
- RT the labels are related (other relationships among terms held in WordNet, excluding the previous mentioned ones)

Relationships Producers:

Structural	Schema-derived relationships
Inferred	Inferred relationships
Lexical	Lexicon-derived relationships
User-Defined	Designer-provided relationships

In Fig. A.17 you can see the GUI for the section *Sematic Relationships*. The method used to compute the relationships is the following:

1. Schema-derived and Lexicon-derived relationships extraction. The relationships are automatically extracted by the system. Press the button *Compute Structural and Lexical Rels*.
2. New relationships provided manually by the user. Press the button *Add* and the window shown in Fig. A.18 will appear; it allows you to add one ore more relationships among attributes or classes.
3. Inferring new relationships. The relationships are inferred by means of a Description Logics engine (ODB-Tools). Press the button *Compute Inferred Rels*.

The GUI permits you also to filter the resulted relationship by Producer, Source, Type (SYN, RT, BT/NT) or Destination, as shown at the bottom of Fig. A.17.

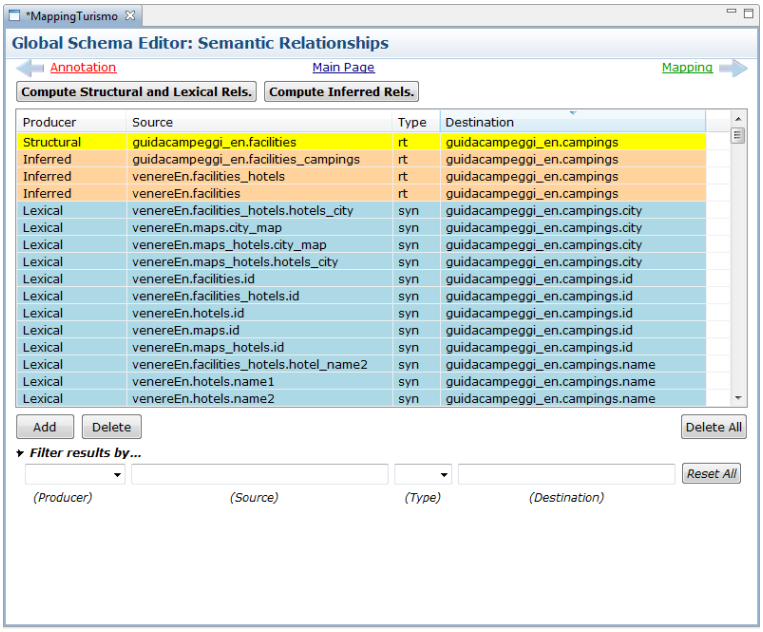


Fig. A.17: Semantic Relationships Editor

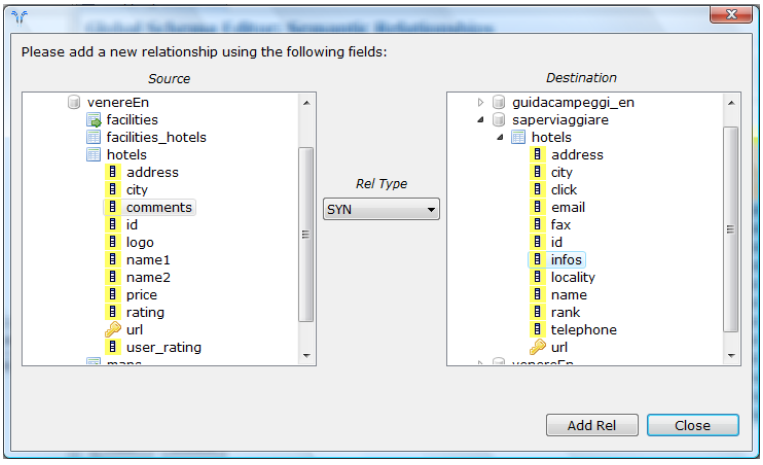


Fig. A.18: User-provided Relationships Interface

A.2.6 Mapping Refinement

The firt step of mapping refinement is the generation of similar classes clusters.

By means of this interface you can change the parameters used by MOMIS to compute the clusters (and the global classes). You can set those parameters in order to obtain the right Global Schema. In Fig. A.19 the default values are shown.

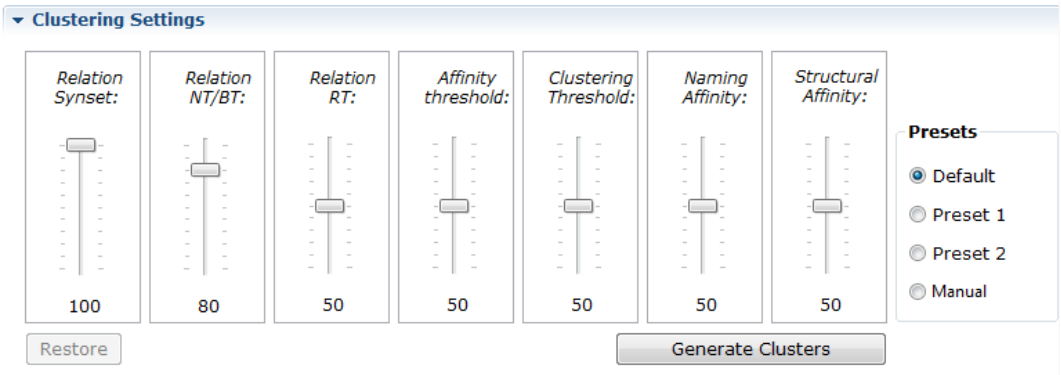


Fig. A.19: Clusters Generation

If you click on the button *Generate Clusters*, the global classes will be created and loaded in the Global Sources tree of the *Mapping Refinement* section (Fig. A.20).

The context menu of the Global Sources tree (right click on global source, global class or global attribute) allows you to:

- add, remove or rename global classes/global attributes;
- set or modify the type of an attribute;
- edit the resolution function of a global attribute;
- edit the transformation function of a local attribute;
- set an attribute as ”join attribute”(involved in a join condition);
- edit join functions of global classes.

The ”Mapping” phase permits the user to visualize and manage the Global Attributes of each Global Class created.

By selecting a node or a leaf of the *Global Classes Tree* the contents of the selected Global Class is shown. In particular, the user may visualize and

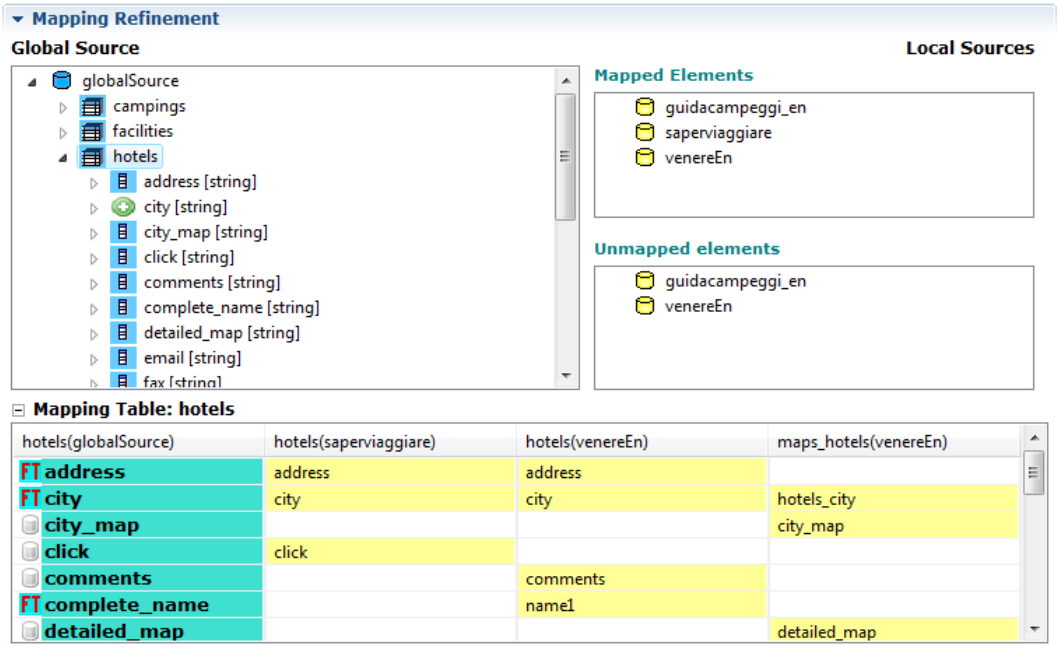


Fig. A.20: Mapping refinement

manage the Global Attributes (blue icon) and, by opening each node, the local attributes the are mapped on it (yellow icon).

If you right-click on a Global Attribute and choose *Remove Global Attribute* the attribute will be removed and the previously local mapped attributes are moved in the *Unmapped elements* panel on the right. Attributes from *Unmapped elements* panel can be mapped in a global attribute moving them on it, using Drag&Drop.

When a global class is selected, the corresponding *Mapping Table* is shown in the lower panel of the window.

The Mapping Table shows how local attributes are ”mapped” into a global attribute. The leftmost column of the *Mapping Table* represents the list of all the global attributes , the first row represents all the local classes belonging to the global class; the table elements are the local attributes (that are part of a local class) ”mapped” in a specific global attribute (row). More attributes can be mapped by the same global attributes. Each global or local attribute which has been transformed by a Resolution or a Transformation Function has a particular icon (the two letters ”FT”), and if you double-click on it you can see and edit the corresponding function.

Transformation Function

A local transformation function is applied to each local attribute mapped by the mapping table. The simplest transformation function is the identity (default). By right-clicking on a mapped local attribute and choosing *Edit Transformation Function* a new window appears (Fig. A.21).

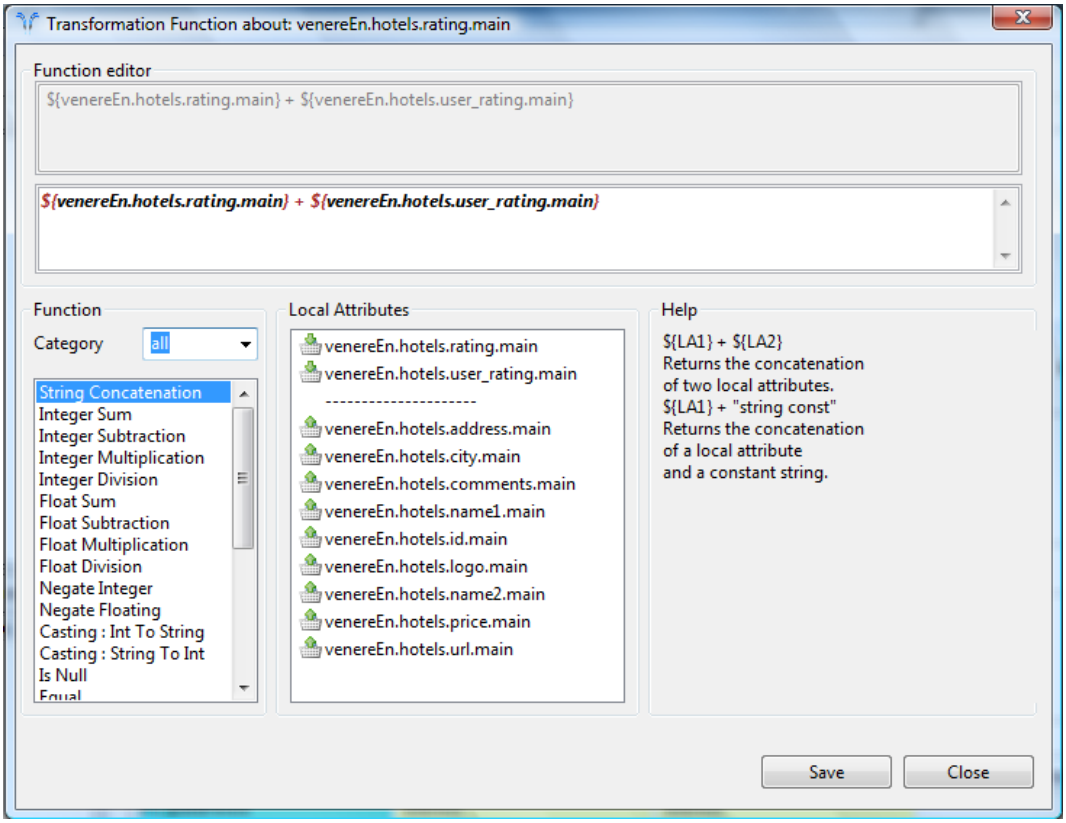


Fig. A.21: Transformation function

The *Function editor* area will show the created function, the *Local attributes* area will show the local attributes, the *Function* area will show the functions applyable to the chosen attributes.

In order to compose the function you have to doubleclick on a function (so it will be written in the *Function editor*), then choose the attributes you want to involve and put them into the function as parameters by clicking twice. At any time you may manually edit the function in the *Function editor*. Finally click on *Save* for saving your work.

Global Resolution Function

The Global Attribute value is obtained by applying the Global Resolution Function to the values obtained by the local to global function. By right-clicking on a global attribute and choosing *edit Resolution Function* a new window appears (Fig. A.22). The interface working is similar to the Local Transformation Function, but the function allowed are different:

- If function :

```
functionif (${@condition}, ${@true}, ${@false})
```

- Coalesce function:

```
coalesce(${@function1},${@function2})
```

The *Coalesce function* gives the priority to attributes not null and the *If function* allows you to impose a condition among attributes: if the condition is true the function returns the first attribute, else returns the second one. By clicking on *Save*, the resolution function will be applied to the global attribute involved.

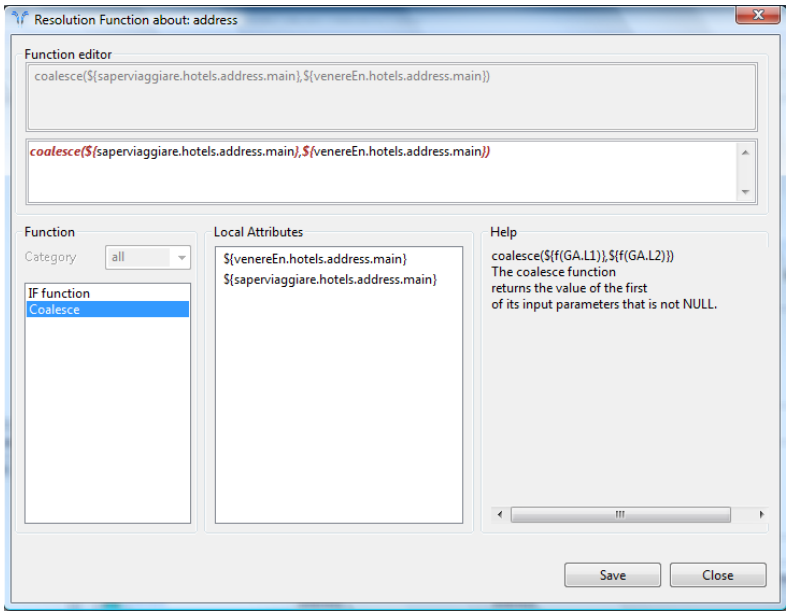


Fig. A.22: Global Resolution function

Join Function

Lastly from the Mapping Refinement interface you can impose join conditions between local classes mapped on a global class, by right-clicking on a global class,choosing *edit Join Function*, writing the function in the Join Function Panel and save it. The system can provide a *default Join Function* as shown in Fig. A.23.

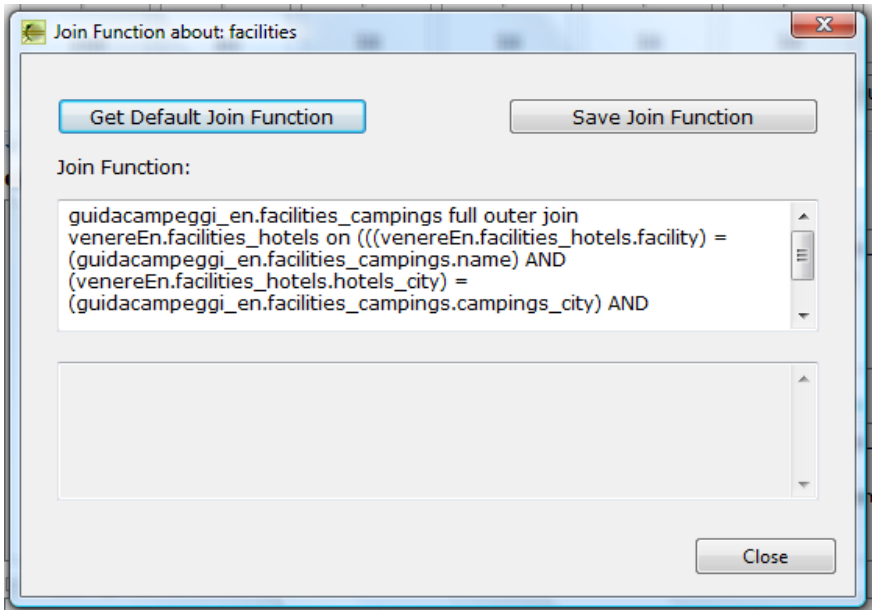


Fig. A.23: Join Function

A.2.7 The Query Manager

After completing each section for building the Global Schema, you can start executing queries by right-clicking on it and choosing *Launch Query Manager* (Fig. A.24).

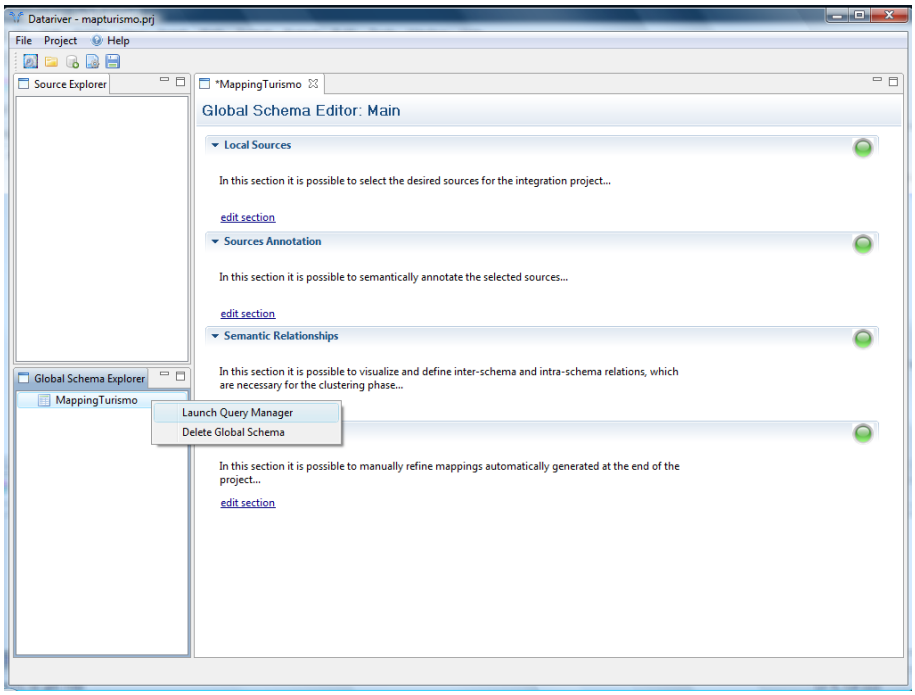


Fig. A.24: Launch Query Manager

All the local sources have to be available in order to start the Query Manager.

To show the Query Manager functionalities, we load a previously created Global Schema of three sources (named ”campings”, ”facilities” and ”hotels”), containing real data related to tourist structures.

The syntax of a query is

Q = select <Q_select-list> from G where <Q_condition>

where <Q_condition> is a Boolean expression of positive atomic constraints: (GA1 op value) or (GA1 op GA2), where GA1 and GA2 are attributes of the global class G and op is a relational operator and value is a constant value (all constant values are denoted in quotation mark: ’12’, ’house’, ..)..

In the panel at the top of the interface you can write your query, for example:

Q1: select name, province, url from campings where province like 'RM'

After clicking on *Run Query*, the query will be executed and the results will be loaded in a table as shown in Fig: A.25.

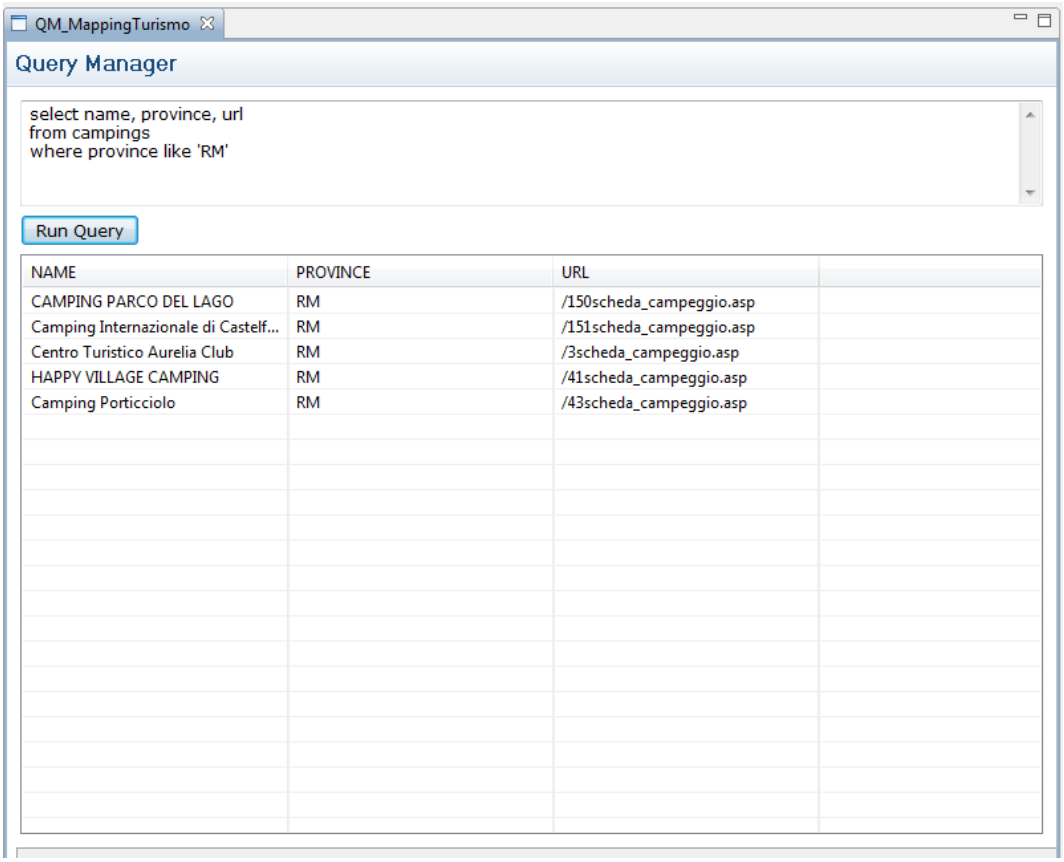


Fig. A.25: Query Manager Interface

A.3 Web interface of the MOMIS Query Manager

A.3.1 Query composition

For composing and executing your query you have to follow some main steps (Fig. A.26):

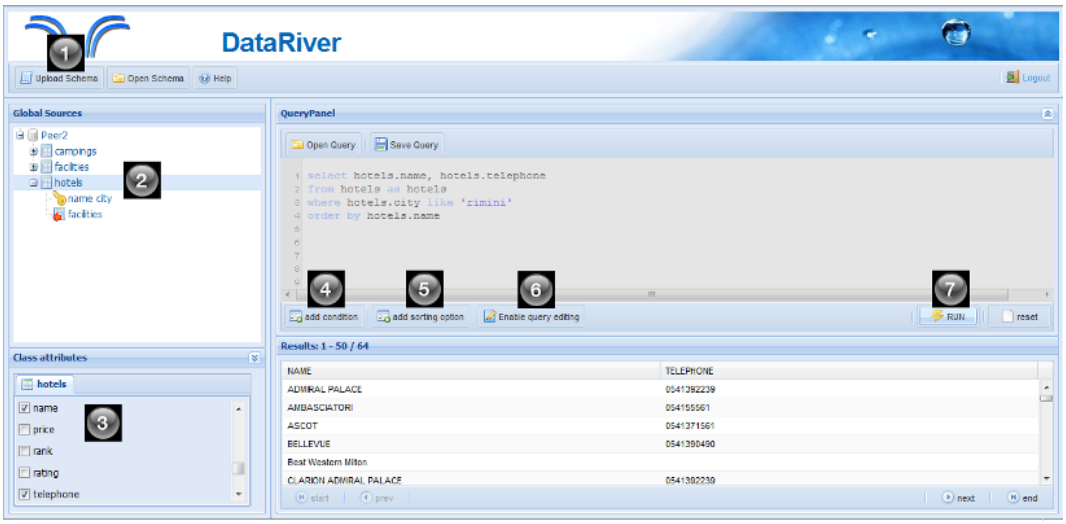


Fig. A.26: Main steps for query composition and extraction

1. Upload the Global Schema you previously created with MOMIS; the left-panel of the interface will load the Global Source tree.

2. Click on a class of the tree and you will see its attributes as checkbox down in the "Class attributes" panel where you can select them.

3. The attributes you choose will be written in the *query editor*, together with the class.

   Note: If you want to perform a two-way JOIN between classes, you have to click on the first class, then click on a class referenced; an alert message will ask you if you want to join the classes; after clicking the "yes" button you will see the attributes of both classes in the "Class attributes" panel, in two different tabs, and the join condition will be automatically written in the query editor. In the *GlobalSources Tree* the node referred to the second class will be expanded, and the reference to the first class will be enlighted.

4. After choosing the attributes you can add and/or conditions by clicking the ”add condition” button (Fig. A.27). Conditions may concern any attribute of the classes involved, even the ones you didn't select.

5. If you want the query result to be ordered by a specific attribute set you may also add sorting options (Fig. A.28), by clicking the corresponding button. Sorting options may concern only the attributes included in the query.

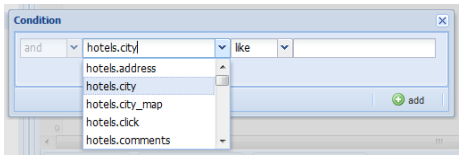


Fig. A.27: Add condition panel

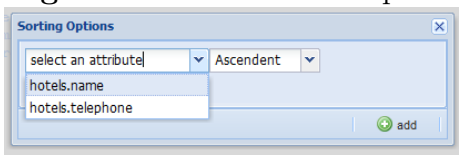


Fig. A.28: Add sorting options panel

6. At any step of this sequence you may manually modify the code in the query editor, but it's recommended to do it only after point 5, just before running the query.

7. Run the query and you will see the corresponding output in the ”Results” grid.

The results are paginated 50 at a time,and you can look through them using the bottombar of the grid. You may expand the grid by collapsing the top panel of the viewport.

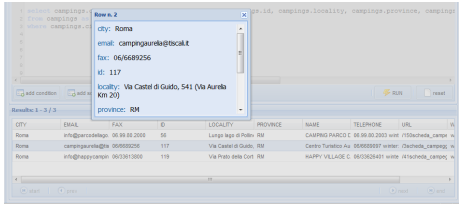


Fig. A.29: Row data in a new window

Lastly, if you click on one of the rows of the resultsGrid, you will see a new window (Fig. A.29) containing all the attributes of that row, in order to look through them in a more readable way.

A.3.2 Query saving

At any time you can save your query, by clicking on the "Save Query" button and inserting the name you want to assign to it (Fig. A.30).

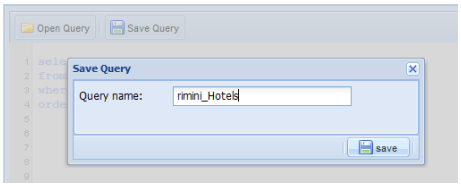


Fig. A.30: Query saving

If you click on the "Open query" button a new window will appear (Fig. A.31), containing all the queries you saved before, their names and the Global Schema on wich they had been executed.

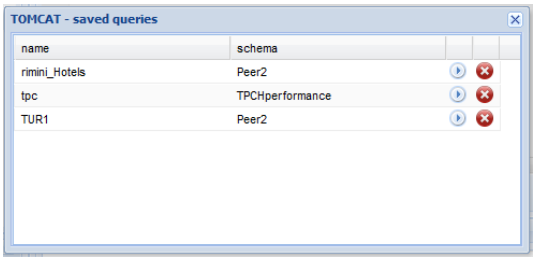


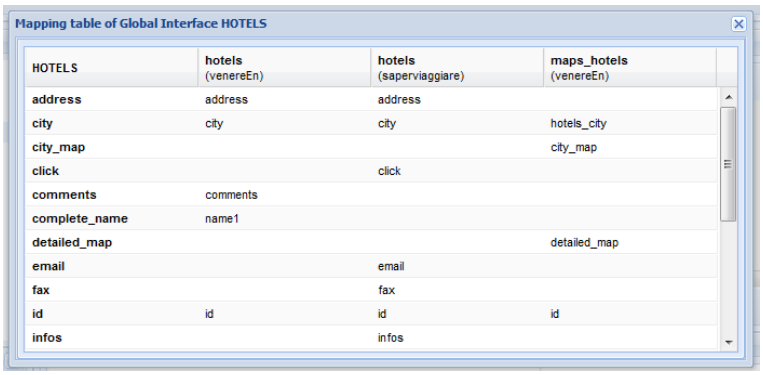
Fig. A.31: Open a previously saved query

From there you can run or delete any of this queries by clicking on the corresponding button.

A.3.3 Mapping table

If you right-click on a *globalClass* of the *globalSource Tree*, a new window will appear, containing the Mapping Table of the class (Fig. A.32), that shows how local attributes are "mapped" into a global attribute. The leftmost column of the "Mapping Table" represents the list of all the global attributes, the first row represents all the local classes belonging to the global class; the table elements are the local attributes (that are part of a local class) "mapped"

in a specific global attribute (row). More attributes may be mapped into the same global attribute.



HOTELS	hotels (venereEn)	hotels (saperviaggiare)	maps_hotels (venereEn)
address	address	address	
city	city	city	hotels_city
city_map			city_map
click		click	
comments	comments		
complete_name	name1		
detailed_map			detailed_map
email		email	
fax		fax	
id	id	id	id
infos		infos	

Fig. A.32: Hotels Mapping Table

Appendice B

Tecnologie e Frameworks

B.1 Java Persistence Api

Java Persistence Api (JPA) è un framework che permette agli sviluppatori di gestire dati relazionali in piattaforma Java. Nella progettazione orientata agli oggetti, la parte più importante è la definizione del *domain model* (modello di dominio). È in questa fase che si stabiliscono le relazioni tra gli oggetti e quindi le modalita di gestione a livello applicativo.

Portare un modello a oggetti in un database relazionale è una cosa molto complessa in quanto, pur potendo avere dei punti di contatto, si tratta di due paradigmi differenti, in particolare quando si fa riferimento alle relazioni.

A livello applicativo è molto importante capire come la presenza di uno strumento che faciliti (automatizzandole) le operazioni di collegamento tra le entità che compognono il modello di dominio sia di notevole aiuto alla manutenzione della logica di persistenza. Una volta definite le relazioni, infatti, attraverso la codifica delle regole che collegano le entità tra di loro, JPA permette di mantenere allineati lo schema della base di dati e il relativo modello di dominio.

Attraverso la definizione delle entità che compongono l'applicazione e di come tra di loro sono relazionate, in automatico avremo una rappresentazione relazionale e relativa persistenza.

Le classi che rappresentano i modelli sono composte da semplici oggetti Java, i cosiddetti POJO. Questi oggetti sono dei Javabean che incapsulano le informazioni in unità ben distinte e facilmente identificabili dal punto di vista funzionale. Non implementano alcuna interfaccia, sono semplici classi Java con le regole seguite dai Javabean.

Jpa sfrutta annotazioni (o deployment descriptor) per fornire al persistence provider le seguenti informazioni:

- quali sono gli oggetti del domain model
- come identificare univocamente tali oggetti
- quali relazioni esistono tra gli oggetti
- come mappare un oggetto in una tabella del database

Le annotazioni principali sono:

- @Entity : marca un POJO come un oggetto del domain model (entity bean).
- @Table : necessaria solo se si vuole dare alla tabella sul database un nome diverso rispetto al nome della classe.
- @Column : necessaria solo se si vuole dare all'attributo sul database un nome diverso rispetto al nome dell'attributo nella classe.
- @Id : marca l'attributo che rappresenta la chiave primaria del modello.
- @IdClass : viene utilizzata nel caso in cui la chiave primaria sia rappresentata da un insieme di attributi.
- @Transient : evita che una proprietà diventi persistente.
- @OneToOne, @OneToMany, @ManyToOne, @ManyToMany : definiscono il tipo di relazione tra modelli.

Il file persistence.xml contiene configurazioni necessarie per definire la persistenza; in particolare definisce la *Persistence Unit* (che contiene le tabelle che saranno mappate nel Database) e i parametri necessari per la connessione al database.

Una volta definita la persistenza, si potranno creare metodi per accedere ai dati ed eseguire transazioni. L'*Entity Manager* fornisce metodi per iniziare e finire transazioni, memorizzare, aggiornare, cancellare e trovare entità nel *persistence context*; una volta istanziato, sarà lo strumento per eseguire le queries.

Il linguaggio utilizzato per effettuare query su entità memorizzate sul database è il *Java Persistence Query Language* (JPQL). La sintassi è simile a quella di SQL, ma opera su oggetti entità piuttosto che direttamente sulle tabelle del database e permette di scrivere query portabili, indipendentemente dal sottostante data store.

Fig. B.1: Architettura di *Spring*

B.2 Spring

Spring è un framework open source per lo sviluppo di applicazioni su piattaforma Java ed è stato largamente riconosciuto all'interno della comunità Java quale valida alternativa al modello basato su Enterprise JavaBeans (EJB). Rispetto a quest'ultimo, il framework Spring lascia una maggiore libertà al programmatore fornendo allo stesso tempo un'ampia e ben documentata gamma di soluzioni semplici adatte alle problematiche più comuni.

L'architettura di Spring è rappresentata in figura B.1 e si compone di diversi moduli che forniscono una vasta gamma di servizi:

- Inversion of Control (Core)
- Programmazione orientata agli aspetti (AOP)
- Accesso ai dati (DAO): per interagire con RDBMS in piattaforma Java tramite JDBC e strumenti per il mapping degli oggetti.

- Gestione delle transazioni (ORM): unifica diverse API di gestione delle transazioni e coordina le operazioni tra oggetti Java
- Model-view-controller
- Testing : classi di supporto per la scrittura di unit test e test di integrazione

B.2.1 Inversion of Control

L'Inversion of Control (IoC) è un principio architetturale, che ha avuto presa sulla comunità dei programmatori grazie a *Spring*, basato sul concetto di invertire il controllo del flusso di sistema (Control Flow) rispetto alla programmazione tradizionale.

Nella programmazione tradizionale la logica del Control Flow è definita esplicitamente dallo sviluppatore, che si occupa tra le altre cose di tutte le operazioni di creazione, inizializzazione ed invocazione dei metodi degli oggetti. IoC invece inverte il control flow facendo in modo che non sia più lo sviluppatore a doversi preoccupare di questi aspetti, ma il framework, che reagendo a qualche "stimolo" se ne occuperà per suo conto. Questo principio è anche conosciuto come *Hollywood Principle* ("Non chiamarci, ti chiameremo noi").

La *Dependency Injection* è una delle tecniche con le quali si può attuare l'IoC. Essa prende il controllo su tutti gli aspetti di creazione degli oggetti e delle loro dipendenze consentendo, tra le altre cose, di eliminare dal codice applicativo ogni logica di inizializzazione. Normalmente, senza l'utilizzo di questa tecnica, se un oggetto necessita di accedere ad un particolare servizio, l'oggetto stesso si prende la responsabilità di gestirlo, o avendo un diretto riferimento al servizio, o individuandolo con un *Service Locator* che gli restituisce un riferimento ad una specifica implementazione del servizio.

Con l'utilizzo della *dependency injection*, l'oggetto ha in sè solamente una proprietà che può ospitare un riferimento a quel sevizio, e quando l'oggetto viene istanziato, un riferimento ad una implementazione di questo servizio gli viene iniettata dal framework esterno (*Spring*), senza che il programmatore che crea l'oggetto sappia nulla sul suo posizionamento del servizio o altri dettagli sullo stesso.

B.2.2 Programmazione orientata agli aspetti

La programmazione orientata agli aspetti (AOP) è un paradigma di programmazione basato sulla creazione di entità software (denominate aspetti) che

sovrintendono alle interazioni fra oggetti finalizzate ad eseguire un compito comune. Il vantaggio rispetto alla tradizionale Programmazione orientata agli oggetti (OOP) consiste nel non dover implementare separatamente in ciascun oggetto il codice necessario ad eseguire questo compito comune.

Uno dei principi fondamentali dell' OOP è la modellazione del programma come uno scambio di messaggi tra oggetti, i quali sono entità tra loro indipendenti. Tale principio garantisce sicuramente la modularità del sistema software sviluppato con un linguaggio object oriented, ma al tempo stesso rende difficile l'implementazione di alcune funzionalità che per loro natura sono comuni a più oggetti (quali, ad esempio logging e sicurezza). Ci sono cioè alcuni compiti del sistema informativo che non possono essere modellati come oggetti, semplicemente perché interessano l'applicazione nel suo insieme.

Prendiamo ad esempio un'applicazione che deve effettuare il logging di alcune transazioni che richiedano l'interazione tra più oggetti: ognuno degli oggetti coinvolti deve, nei propri metodi, contenere codice in grado di gestire il suddetto problema; si viene così a creare, nella stesura del codice, una ridondanza non necessaria. Inoltre gli oggetti modificati a tale scopo diventano più complessi e quindi diminuisce la manutenibilità del programma.

L'aspect oriented programming è nato con lo scopo di risolvere problemi di questo tipo. Gli aspetti modellano cioè le problematiche trasversali agli oggetti stessi, ossia compiti (quali ad esempio l'accounting) che nell' OOP tradizionale sono difficilmente modellabili.

Gli aspetti sono quindi delle entità esterne agli oggetti che osservano il flusso del programma generato dalle interazioni tra oggetti, modificandolo quando opportuno.

B.2.3 Accesso ai dati

Spring prevede il supporto dei framework più popolari per l'accesso ai dati in ambiente Java: JDBC, iBatis, Hibernate, JDO, JPA, Oracle TopLink, Apache OJB e Apache Cayenne.

Per tutti questi frameworks Spring fornisce le seguenti funzionalità:

- gestione delle risorse: permette l'acquisizione e il rilascio in modo automatico delle risorse del database
- gestione delle eccezioni: gestisce tutte le eccezioni relative all'accesso ai dati
- transaction participation: partecipa in modo trasparente alle transazioni in corso

Tutte queste funzionalità diventano disponibili utilizzando le classi Template fornite da *Spring* per ogni framework supportato.

B.2.4 Gestione delle transazioni

Spring prevede un meccanismo di astrazione per la piattaforma Java in grado di gestire transazioni locali e globali, semplici o nidificate in quasi tutti gli ambienti della piattaforma Java. Spring implementa un *PlatformTransactionManager* che si occupa delle strategie di gestione delle transazioni:

- Transazioni gestite su una connessione JDBC
- Transazioni gestite tramite il TransactionManager JTA e UserTransaction

Accanto a questo meccanismo di astrazione Spring prevede inoltre due modi di aggiungere alle applicazioni dei moduli per la gestione delle transazioni:

- tramite programmazione, utilizzando il TransactionTemplate
- tramite configurazione, usando metadati come XML o Java annotations

B.2.5 Model-View-Controller

Il Model-View-Controller (MVC, Modello-Vista-Controllore) è un pattern architetturale molto diffuso nello sviluppo di interfacce grafiche di sistemi software object-oriented.

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- il model fornisce i metodi per accedere ai dati utili all'applicazione;
- il view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Ne consegue la separazione fra la logica applicativa (o logica di business), a carico del controller e del model, e l'interfaccia utente a carico del view.

I dettagli delle interazioni fra questi tre oggetti software dipendono molto dalle tecnologie usate (linguaggio di programmazione, eventuali librerie, middleware...) e dal tipo di applicazione (per esempio se si tratta di un'applicazione web o di un'applicazione desktop).

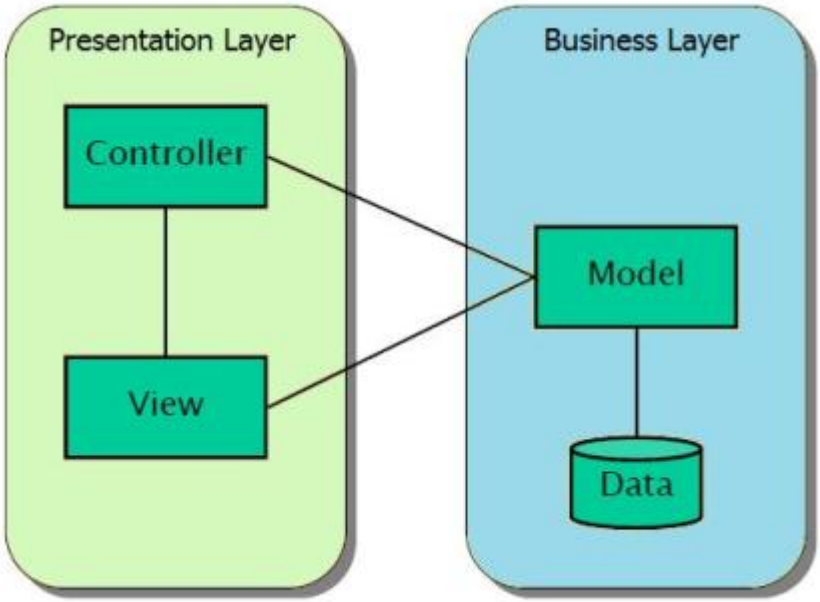


Fig. B.2: Model View Controller

B.2.6 Testing

Un'altra delle caratteristiche per il quale *Spring* si contraddistingue è il supporto al testing. Questo livello mette a disposizione un ambiente molto potente per il test delle componenti *Spring*, grazie anche alla sua integrazione con JUnit e TestNG e alla presenza di Mock objects per il testing del codice in isolamento.

B.3 Struts 2

Apache Struts 2 è un framework estensibile per creare Applicazioni web java.

Il framework è stato creato per garantire l'intero ciclo di sviluppo: dalla costruzione, all'installazione su server, fino alla manutenzione dell'applicazione nel tempo.

Struts 2 estende le Java Servlet, incoraggiando gli sviluppatori all'utilizzo del pattern Model-View-Controller:

- **Model**: implementa la logica applicativa, ed è costituito da un insieme di classi java, tipicamente Java Bean;

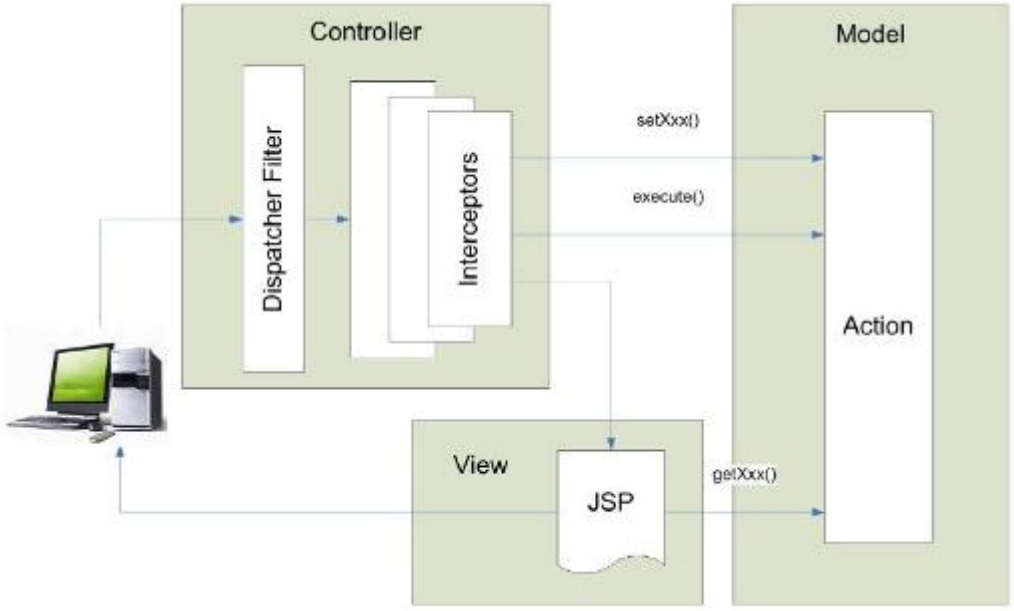


Fig. B.3: MVC pattern in *Struts 2*

- **View**: insieme di pagine JSP costruite mediante l'ausilio di particolari tag offerti da *Struts 2*;
- **Controller**: Il controllo viene affidato ad un filtro (*Dispatcher Filter*) che gestisce le classi Action ed eventuali classi Helper; le classi Action rispecchiano una struttura flessibile e possono estendere o implementare elementi messi a servizio dal framework. In base delle richieste dell'utente il controller decide quale Action eseguire per interagire con il modello. Queste informazioni sono elencate e gestibili dal file di configurazione Struts.xml che si trova nel classpath. Sulla base del risultato ritornato dalla Action eseguita, il controller decide a quale pagina Jsp affidare la gestione della risposta.

Uno dei concetti nuovi di *Struts 2* è rappresentato dagli Interceptor, classi stateless (che non mantengono uno stato tra invocazioni successive) che possono essere invocate automaticamente prima e dopo una Action. Di default, *Struts 2*, prevede un gruppo di interceptor, che vengono richiamati prima di invocare qualsiasi action. Il cosiddetto stack di default, prevede ben 17 interceptor che lavorano dietro le quinte per offrire vari servizi. I principali sono i seguenti:

- Exception: permette di mappare una particolare eccezione ad una vista;

- Prepare: permette di richiamamare un metodo di inizializzazione della Action;
- I18n: gestisce la memorizzazione del locale per l'utente corrente;
- Debugging: permette di attivare il debug delle viste;
- FileUpload: permette di gestire l'upload dei file;
- Validation: permette di eseguire la validazione dei dati forniti nella form, congruentemente al contenuto dei relativi file xml di definizione dei controlli.

Oltre agli interceptor inclusi nello stack di default, *Struts 2* permette di configurare ulteriori interceptor disponibili, utili soltanto in situazioni particolari.

B.4 JavaScript, Ajax, Ext-Js

B.4.1 JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti comunemente usato nei siti web.

La caratteristica principale di JavaScript è quella di essere un linguaggio interpretato. Il codice quindi non viene compilato bensì c'è un interprete (lato client, incluso nel browser) che esegue riga per riga, a tempo di esecuzione, quanto trascritto nello script. JavaScript presenta quindi tutte le caratteristiche di un normale linguaggio interpretato (e di conseguenza i suoi vantaggi e svantaggi) con una sintassi analoga a quella di un linguaggio compilato (essa è relativamente simile a quella del C, del C++ e del Java), quindi con la possibilità di utilizzare funzionalità tipiche dei linguaggi di programmazione ad alto livello (strutture di controllo, cicli, etc.) e con in più anche la potenzialità di definire strutture più complesse, vicine a quelle adottate nei normali linguaggi object oriented (creazione di prototipi, istanziazione di oggetti, costruttori).

Un'altra caratteristica importante di JavaScript consiste nel suo essere un linguaggio debolmente tipizzato; quindi il tipo delle variabili può non essere assegnato in fase di dichiarazione e le variabili stesse vengono convertite in maniera automatica dall'interprete.

Inoltre JavaScript è un linguaggio debolmente orientato agli oggetti. Ad esempio, il meccanismo dell'ereditarietà è più simile a quello del Self e del NewtonScript che a quello del linguaggio Java (che è un linguaggio fortemente

orientato agli oggetti). Gli oggetti stessi ricordano più gli array associativi del Perl che gli oggetti di Java o del C++.

Altri aspetti di interesse: il codice viene eseguito sul client, quindi il server non viene sollecitato. Ciò risulta essere un vantaggio in quanto con la presenza di script particolarmente complessi il server non verrebbe sovraccaricato. Di contro, nel caso di script che presentino una considerevole mole di dati, il tempo di attesa potrebbe diventare molto lungo. Inoltre, lavorando solamente sul client, ogni informazione che presuppone un accesso a dati memorizzati in un database deve essere rimandata ad un linguaggio che effettua esplicitamente la transazione per poi restituire i risultati ad una o più variabili JavaScript; operazioni del genere richiedono il caricamento della pagina stessa. Con l'avvento di AJAX però tutti questi limiti sono stati superati.

B.4.2 Ajax

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application).

Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente.

AJAX è asincrono nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente.

Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è necessario che le richieste di caricamento debbano essere necessariamente asincrone.

AJAX è una tecnica multi-piattaforma utilizzabile su molti sistemi operativi, architetture informatiche e browser web, ed esistono numerose implementazioni open source di librerie e framework.

La tecnica Ajax utilizza una combinazione di:

- HTML (o XHTML) e CSS per il markup e lo stile;
- DOM (Document Object Model) manipolato attraverso un linguaggio ECMAScript come JavaScript o JScript per mostrare le informazioni ed interagirvi;
- l'oggetto XMLHttpRequest per l'interscambio asincrono dei dati tra il browser dell'utente e il web server.

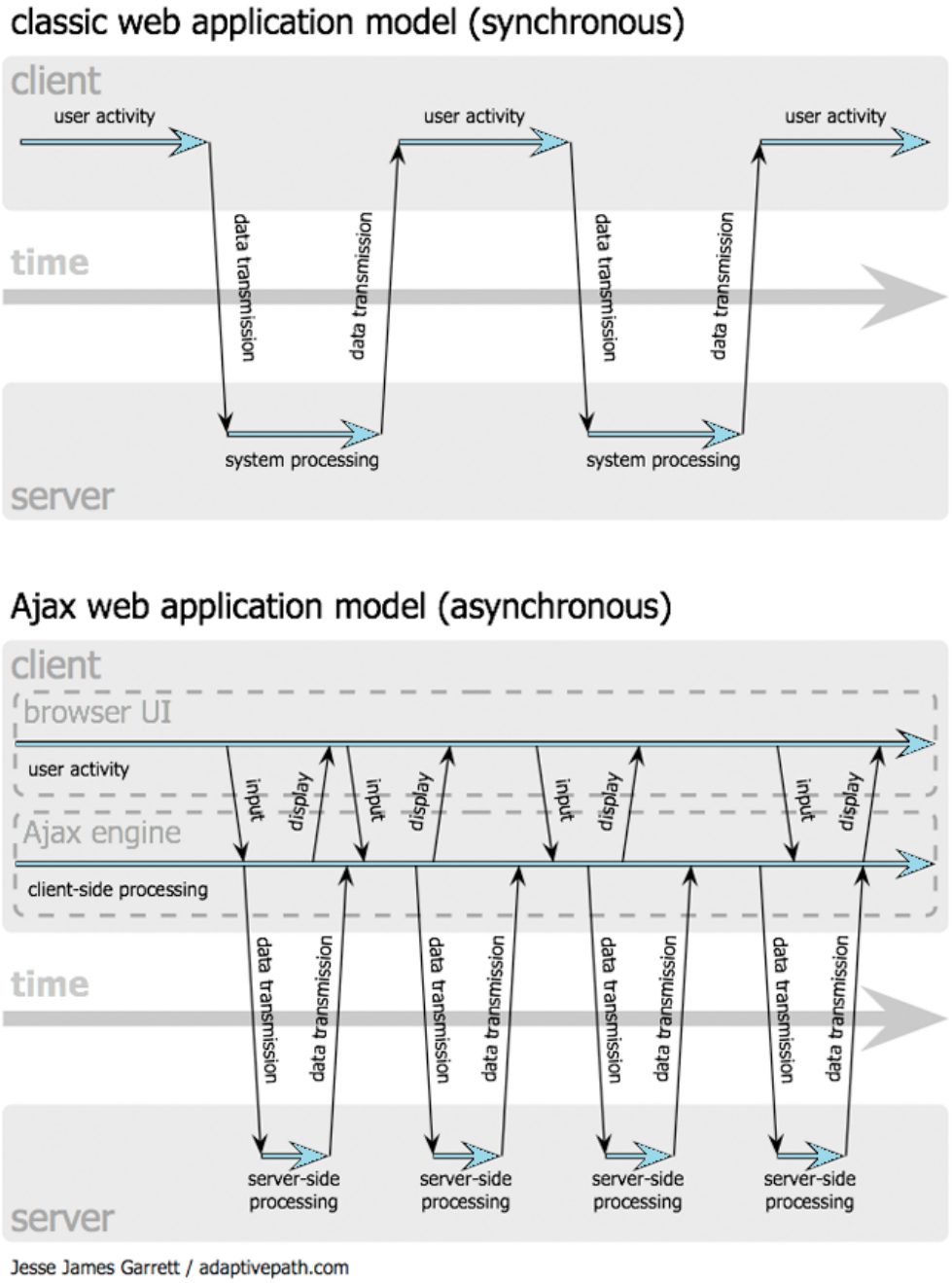


Fig. B.4: Ajax, interazione asicrona client-server

In genere viene usato XML come formato di scambio dei dati, anche se di fatto qualunque formato può essere utilizzato, incluso testo semplice, HTML preformattato, JSON.

Ajax quindi non è una tecnologia individuale, è un gruppo di tecnologie utilizzate insieme.

Il vantaggio di usare AJAX è la grande velocità alla quale un'applicazione risponde agli input dell'utente, mentre lo svantaggio più degno di nota è che, senza l'adozione di adeguate contromisure, le applicazioni AJAX possono rendere non utilizzabile il tasto "indietro" del browser: con questo tipo di applicazioni, infatti, non si naviga da una pagina all'altra, ma si aggiorna di volta in volta una singola parte del medesimo documento. Proprio per questo i browser, che sono programmi orientati alla pagina, non hanno possibilità di risalire ad alcuna di tali versioni "intermedie".

B.4.3 Ext-Js

Ext-JS è un prodotto JavaScript catalogabile all'interno di queste due categorie di software:

- libreria grafica: in quanto presenta notevoli componenti grafici (griglie di dati, finestre, alberi, bottoni, form) che permettono di arricchire sia dal punto di vista delle funzionalità offerte che dal punto di vista grafico le applicazioni web;

- framework: in quanto sono presenti una moltitudine di funzionalità che arricchiscono Javascript offrendo sia un supporto cross-browser sia l'utilizzo di tecniche di programmazione altrimenti impensabili (programmazione ad oggetti e programmazione ad eventi).

ExtJS è senza dubbio una delle più conosciute librerie grafiche presenti nel panorama Web; presenta un'interfaccia gradevole alla vista, molto usabile e personalizzabile (sono molti i temi di terze parti scaricabili),inoltre è una libreria completamente orientata agli oggetti. Tutto in *ExtJS* è un oggetto con propri attributi e metodi richiamabili come qualsiasi altro linguaggio con queste caratteristiche. Questo permette di avere codice ordinato e di non perdersi tra il disordine che Javascript spesso genera data la sua alta flessibilità.

Un ultimo punto di forza di *Ext-Js* è la gestione degli eventi: grazie alla presenza di eventi personalizzati, si ha la possibilità di avere componenti grafici tra loro indipendenti, ma che riescono ad interagire in modo semplice con il resto dell'applicazione permettendo un'integrazione efficace dei dati e delle informazioni.

Ringraziamenti

Ringrazio tutta la mia famiglia, per il sostegno morale ed economico di cui non mi ha mai privato.

Ringrazio la Professoressa Sonia Bergamaschi per avere seguito il mio lavoro in questi mesi.

Ringrazio la Quix S.r.l in tutti i suoi componenti per l'accoglienza e il tempo dedicatomi, in particolare Andrea Prandini e Daniele Miselli che si sono personalmente occupati della mia formazione.

Ringrazio il team di DataRiver per avermi "ospitata" tra loro nell'ultimo mese di tirocinio, li ringrazio per la disponibilità dimostrata e per la piacevole compagnia.

E infine ringrazio tutti i miei amici, che in questo periodo mi hanno ascoltata, appoggiata e divertita.

Ringrazio tutti veramente di cuore, il merito di questo mio grande traguardo è tutto vostro.