Parole chiave:

Sistemi a Mediatore

Integrazione Dati

CereaLab

Wrapper

Estrazione Dati

Indice

Elenco delle figure

Introduzione

Negli ultimi anni i progressi nel campo della biologia molecolare hanno portato ad una crescita esponenziale delle informazioni di carattere biologico a disposizione dei ricercatori. Il grande problema che ci si trova ora a dover affrontare è come rendere facilmente accessibile questa mole di dati a coloro che svolgono ricerca nel campo delle biotecnologie. Questa tesi nasce con questo obbiettivo, in particolare quello di realizzare un database destinato ad assistere il processo di costituzione varietale di quattro specie di piante di cereali, l'orzo, il grano, il riso ed il mais, processo per il quale è necessario avere accesso ad una grande quantità di dati sui caratteri fenotipici e genotipici delle piante in esame. Lo scopo particolare è quello di mettere a disposizione dei breeder, o coltivatori, presenti nella regione Emilia Romagna informazioni che consentano il miglioramento di queste specie di cereali.

Il problema iniziale che si è incontrato è stato quello del reperimento di queste informazioni. Sono numerose infatti le fonti presenti sul Web che mettono a disposizione della comunità scientifica dati molecolari riguardanti le specie in esame. Ottenere questi dati in modo manuale risulta però un processo alquanto laborioso, ma che soprattutto richiede molto tempo. Per questo motivo, nell'ambito del progetto CereaLab, finanziato dalla Regione

Emilia Romagna, in collaborazione con il DataBase Group dell'Università degli Studi di Modena e Reggio Emilia, si è pensato di realizzare il database CereaLab provando ad integrare le informazioni già presenti sul Web utilizzando il sistema a mediatore Momis, sviluppato dal DataBase Group. A questo progetto ha inoltre collaborato Salvatore Curia durante lo svolgimento del tirocinio universitario per la preparazione della sua tesi di Laurea.

Nel capitolo 1 viene descritto il progetto CereaLab e viene data una breve descrizione del dominio applicativo; nel capitolo 2 viene presentato il progetto del database che si voleva ottenere, che è stato condotto in collaborazione con il gruppo di ricerca del Prof.Pecchioni, responsabile del progetto CereaLab. Nel capitolo 3 viene presentato il sistema a mediatore Momis e il processo di creazione dello schema globale, la Global Virtual View (GVV) del database realizzato con tale strumento. Nel capitolo 4 vengono inoltre presentate alcune tecniche ulteriori che sono state prese in esame, in quanto di possibile applicazione per questo progetto, che consentono l'estrazione dei dati in modo automatico dal Web e la progettazione di interfacce user-friendly per aiutare i ricercatori nella fase di interrogazione dei database nel dominio delle biotecnologie. Infine nel capitolo 5 vengono tratte le conclusioni sul lavoro svolto, e vengono suggerite alcune soluzioni per completare la realizzazione del database CereaLab.

Capitolo 1

Descrizione del Progetto

L'obiettivo di questo lavoro di tesi è quello di realizzare il database relazionale CereaLab su richiesta del gruppo di ricerca del Prof.Pecchioni, all'interno di un progetto di ricerca più ampio finanziato dalla Regione Emilia Romagna. Obiettivo principale del laboratorio CereaLab è lo sviluppo di Ricerca ed il Trasferimento di conoscenze e nuove metodologie nel settore delle Biotecnologie non-OGM delle piante dalle Università ed Enti di Ricerca alle Ditte Sementiere attive in Regione che operano nel settore dei cereali. Questo progetto nasce dalla consapevolezza che in questo settore della Ricerca pubblica, cioè quello delle Biotecnologie applicate alla costituzione varietale, esiste in Emilia Romagna una eccellenza a livello nazionale, e dalla consapevolezza che l'Industria Sementiera nazionale, pur di dimensioni ridotte rispetto alle multinazionali del seme, ha una sua eccellenza in Regione. Nel contempo nasce da una esigenza, che è quella di far passare alla Industria Sementiera regionale le conoscenze e l'uso dei marcatori molecolari, al fine di innovare il processo di costituzione varietale. CereaLab riguarda le principali specie ce-

realicole coltivate in Regione. E' articolato in modo orizzontale per problemi biologici (resistenza a stress biotici, efficienza nell'uso dell'azoto e caratteri di qualità). L'innovazione riguarderà tre approcci, che corrispondono ad altrettanti sottoprogetti:

1. Laboratorio Informativo (LabIn): organizzazione e trasferimento di informazioni genetiche e molecolari già disponibili ed acquisite durante il progetto;

2. Laboratorio Genotyping (LabGen): monitoraggio della diversità genetico/molecolare per i caratteri ci cui sopra tramite l'uso di piattaforme degli Istituti di Ricerca e marcatori noti. Monitoraggio della diversità fenotipica per unire fenotipo a genotipo. Trasferimento di tali risultati, tramite il sottoprogetto LabIn alla industria;

3. Laboratorio Discovery (LabDis): individuazione di Geni/Qtl di caratteri utili nei cereali, e sviluppo di nuovi marcatori per l'industria.

In particolare questa tesi costituisce parte integrante dell'attività di trasferimento. Tale attività di trasferimento ha infatti l'obiettivo di selezionare, organizzare ed orientare le informazioni disponibili a livello Internazionale nel settore pubblico (public domain) utili ad assistere il processo di costituzione varietale. Con un progetto pilota a tutt'oggi mai realizzato in regione e nella nazione, si intende infatti realizzare un database orientato per rendere tali informazioni fruibili dal comparto produttivo e dal settore della ricerca.

In questa tesi è stato quindi progettato e realizzato un database finalizzato al miglioramento assistito dei cereali. Tale esperienza risulta, al contrario di quanto accaduto in altri paesi europei, una realizzazione preliminare per il primo database pubblico di questo tipo nella regione Emilia Romagna e in

Italia.

Il primo problema che è stato riscontrato durante la fase di studio di fattibilità di questo progetto è stato quello del reperimento dei dati per popolare il database. Inizialmente l'idea del Prof.Pecchioni era quella di ottenerli manualmente andando a interrogare alcuni database presenti su Internet, che mettono a disposizione della comunità scientifica i loro dati, per poi andarli a inserire nel database che sarebbe stato realizzato. Questa modalità tuttavia sarebbe risultata troppo laboriosa data la vasta mole di dati da reperire. Pur consentendo infatti di popolare il database esclusivamente con i dati di interesse per questo progetto e quindi di scremare fortemente la grande quantità di dati disponibili su Internet, avrebbe comunque richiesto uno sforzo non indifferente, in quanto spesso i database da cui ottenere le informazioni non risultano particolarmente facili da consultare, soprattutto se l'utente non dispone di specifiche competenze sia di carattere biologico che informatico.

Alcuni di questi database però consentono di effettuare il download dell'intero database relazionale. In questo modo è quindi possibile crearsi una copia locale del database originale. In particolare due delle sorgenti dati presenti sul Web che contengono dati utili per popolare il nostro database offrono questa possibilità, ovvero le banche dati Gramene, riguardante riso e mais, e Graingenes, che riguarda le specie orzo e grano.

Avendo quindi la possibilità di ottenere una copia di due database già esistenti, riportanti una grande quantità di dati utili per il nostro fine, si è pensato di utilizzare queste sorgenti per realizzare una prima versione del database, utilizzando il sistema Momis[1], che consente di integrare e interrogare sorgenti dati eterogenee.

1.1 Descrizione del Dominio Applicativo

Viene ora presentata una breve descrizione del dominio applicativo per specificare il significato dei termini di carattere biologico che verranno usati in seguito.

Per ogni specie di pianta si possono avere più varietà, solitamente indicate con il termine cultivar. In botanica col termine cultivar (dall'inglese *culti*vated *var*iety) si intende il sistema di classificazione usato per designare le diverse varietà ottenute da una pianta coltivata. Il termine è stato ufficialmente adottato durante il XIII Congresso di orticultura tenutosi a Londra nel 1952 al fine di distinguerle dalla classificazione usata per le varietà ottenute invece da piante allo stato spontaneo. La parola si può intendere come derivata dall'espressione "varietas culta".

Di particolare interesse per una cultivar sono i caratteri o "traits". Un trait, o carattere, è una caratteristica ereditata geneticamente di un organismo. Il termine fenotipo viene a volte usato come sinonimo, anche se rigorosamente per fenotipo non si intende il trait, ma la possibile espressione che uno specifico carattere può assumere. Per trait si intende quindi ogni singola caratteristica o misura quantificabile di un organismo. Tuttavia, i caratteri più utili a livello di analisi genetica sono quelli presenti sotto forme differenti in individui differenti. Un tratto visibile è il prodotto finale di numerosi processi molecolari e biochimici. Nella maggior parte dei casi, questa informazione ha origine dal DNA che viene trascritto nell'RNA per la produzione di proteine, che influiscono sulla struttura e sulle funzioni di un organismo. Inoltre l'ambiente ed i fattori ambientali giocano un ruolo

fondamentale nella determinazione di un trait.

L'unità ereditabile che può influire su di un trait viene chiamato gene. Un gene è una sequenza di acido desossiribonucleico (DNA) che fa parte di una struttura molto più lunga di DNA chiamata cromosoma. Il gene è situato in un locus cromosomico ben preciso, è in grado di autoduplicarsi, di cambiare e di trasmettersi indefinitamente attraverso il processo ereditario. È un'unità fondamentale del sistema genetico di tutti gli individui che ha la possibilità di "dirigere" caratteri somatici, reazioni biochimiche e caratteri psichici. Quando il gene responsabile di un carattere non è ben identificato, ma viene riconosciuta una regione cromosomica determinante nel controllare quel trait, questa viene definita Qtl.

Un Qtl (dall'inglese Quantitative Trait Locus) è una regione di DNA associata ad un particolare carattere quantitativo. Il Qtl è la regione genomica (locus) nella quale risiede almeno un gene che determina il fenotipo in questione o partecipa nella determinazione. Normalmente infatti un carattere quantitativo è determinato da più geni ad effetti minori e sommabili (un fenomeno detto additività), e per questo è anche indicato come carattere poligenico. Di conseguenza più Qtl, che possono trovarsi anche su diversi cromosomi, sono associati ad un singolo carattere. Appropriati algoritmi (e.g. maximum likelihood) utilizzati da software cosiddetti di Qtl mapping sono in grado di individuare i Qtl in una mappa genetica, e quindi nel genoma di una specie. Il numero di Qtl coinvolti in un carattere fornisce informazioni sulla sua architettura genetica. Per esempio tale numero potrebbe indicare se l'altezza di una specie sia determinata da molti geni (Qtl), l'effetto di ognuno dei quali è di portata limitata, oppure da pochi geni (Qtl) ciascuno

dei quali con un effetto più marcato. Gli algoritmi di analisi Qtl riescono infatti anche a stimare gli effetti dei singoli Qtl sul carattere, attraverso la stima del parametro R^2 (varianza spiegata).

Un allele è invece una forma alternativa di un gene, responsabile della particolare modalità con cui si manifesta il carattere ereditario controllato da quel gene. Un individuo diploide porta due alleli per singolo gene, ciascuno dei quali ereditati per via sessuale da un parentale. Per fare un esempio di facile comprensione, il gene che controlla il carattere "colore degli occhi" esiste in due forme, o alleli, alternative: l'allele "occhio chiaro" e l'allele "occhio scuro". Appropriati algoritmi (e.g. maximum likelihood) utilizzati da software cosiddetti di Qtl mapping sono in grado di individuare i Qtl in una mappa genetica, e quindi nel genoma di una specie.

Secondo la genetica formale classica, per ciascun carattere, ovvero per ciascun gene, ognuno dei due alleli è presente ad uno stesso locus su ciascuno dei due cromosomi che costituiscono, nella cellula, una coppia di omologhi.

Ogni carattere, però, all'interno di una popolazione di individui, può essere rappresentato anche da molti alleli. L'insieme degli alleli che, in una popolazione, controllano (o "codificano per") tutti i caratteri degli individui viene detto pool genico.

Non tutti gli alleli determinano un effetto visibile nell'individuo che ne è portatore; essi si dicono pertanto dominanti, se il carattere da essi controllato si manifesta anche allo stato eterozigote, e recessivi se il carattere non si manifesta in fase eterozigote poiché coperto negli effetti dall'altro allele. Si parlerà invece di allele semidominante se questo si manifesterà con intensità doppia nel suo omozigote rispetto a quella riscontrata nell'eterozigote, che

possiede soltanto una copia dello stesso.

Quindi l'insieme dei caratteri visibili in un organismo prende il nome di fenotipo, mentre l'insieme del suo corredo di geni (comprendente quindi alleli dominanti, recessivi e semidominanti) è detto genotipo. Con il termine germoplasma o "Germplasm" viene indicato il materiale genetico, in particolare assieme alla sua specifica caratterizzazione fenotipica, talvolta anche costituzione molecolare e chimica.

Un marcatore molecolare (marker) è una specifica sequenza di DNA con una posizione nota nel genoma e viene utilizzato per identificare un determinato gene o Qtl. E' noto infatti che parti di DNA che risiedono vicino ad altre tendono ad essere ereditate insieme. Questa proprietà consente l'utilizzo dei marcatori, in quanto essi possono essere utilizzati per determinare la specifica sequenza ereditaria di un gene che sottintenda un fenotipo e che non sia ancora stato localizzato con precisione. I marcatori genetici, per essere utili, devono quindi essere facilmente identificabili e associati ad uno specifico locus.

Capitolo 2

Progettazione del Database

2.1 Progetto Concettuale

Seguendo le indicazioni del gruppo di ricerca del Prof. Pecchioni è stato realizzato uno schema concettuale utilizzando il modello Entity Relationship per capire quali fossero le entità principali da modellare nel database da realizzare. Queste entità sono cultivar, che rappresenta le varietà delle quattro specie di cereali, i geni, i Qtl, i marker e i trait.

In figura 2.1 è riportato lo schema ottenuto con raffigurate per ogni entità le relazioni che le coinvolgono e, per ragioni di spazio e di chiarezza, solamente l'attributo scelto come chiave.

L'entità Cultivar rappresenta le varietà di pianta modellate nel nostro database. Questa entità è in associazione di cardinalità (0,N) con l'entità Marker, in quanto una varietà di pianta può essere stata usata per testare nessuno, uno o più marker. E' invece in associazione di cardinalità (1,N) con l'entità GQ. Gli attributi dell'entità Cultivar sono riportati in figura 2.2.

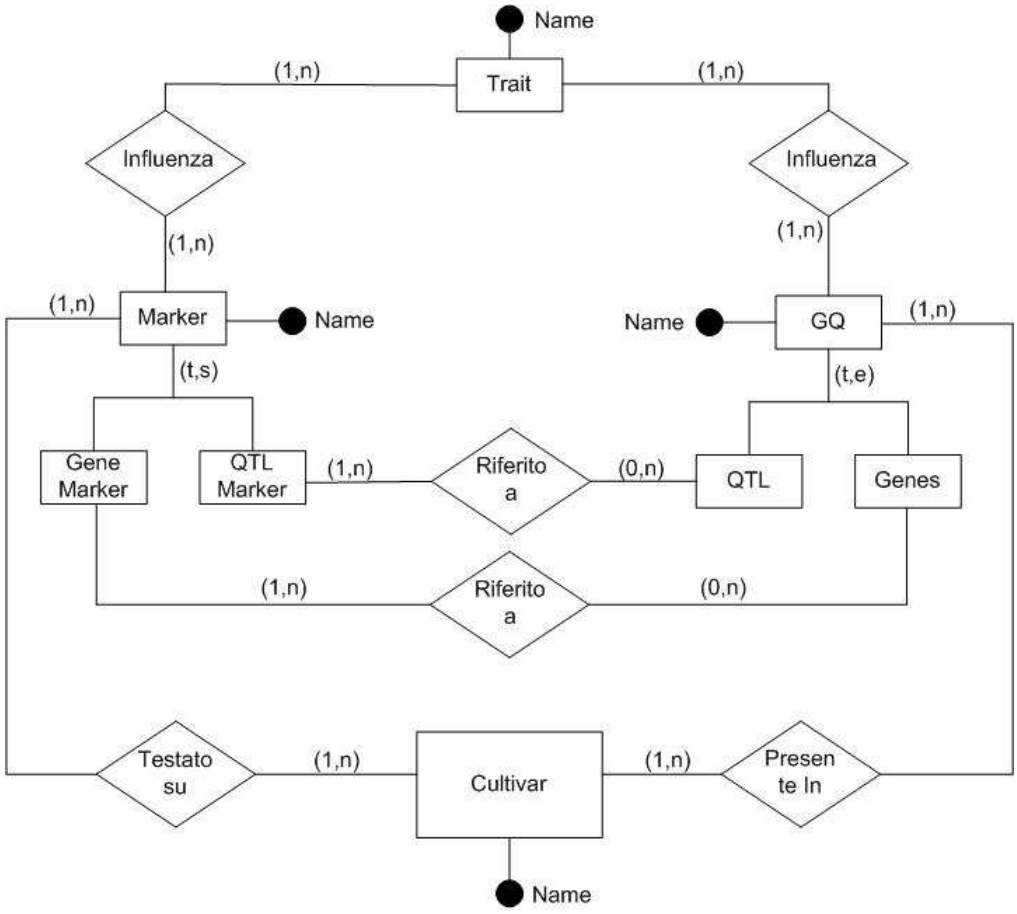


Figura 2.1: Schema E/R

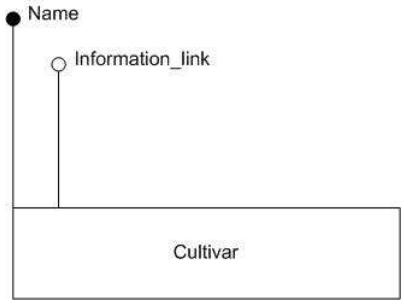


Figura 2.2: Attributi dell'entità Cultivar

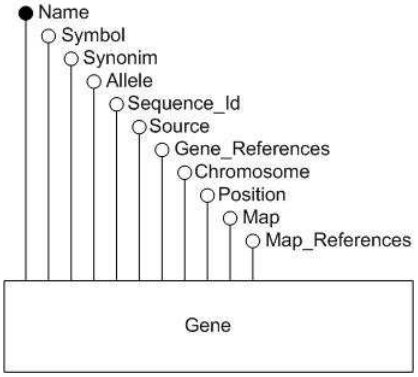


Figura 2.3: Attributi dell'entità Gene

L'entità GQ è una generalizzazione totale ed esclusiva delle entità Qtl e Gene. Infatti, anche se concettualmente molto simili, geni e qtl sono differenziati in quanto un qtl è una regione di DNA che influenza un carattere (trait) di tipo quantitativo, mentre un gene influenza caratteri di tipo qualitativo. Questa entità generalizzata GQ è in associazione di cardinalità (1,N) con l'entità Trait in quanto un gene o un qtl influenza uno o più caratteri di una pianta, e in associazione di cardinalità (1,N) con l'entità Cultivar perchè geni e qtl possono essere presenti in una o più varietà di pianta. Le entità generalizzate Gene e Qtl sono in associazione (0,N) rispettivamente con le entità Gene Marker e Qtl Marker, in quanto rappresentano i marker associati ai rispettivi geni e qtl. Gli attributi delle entità generalizzate Gene e Qtl sono riportati nelle figure 2.3 e 2.4.

L'entità Marker è invece una generalizzazione totale e sovrapposta delle sopracitate entità più specifiche Gene Marker e Qtl Marker. La generalizzazione è sovrapposta in quanto uno stesso marker può essere riferito sia ad un gene che ad un Qtl. Le entità generalizzate rappresentano Marker riferiti a Geni oppure riferiti a Qtl. Le entità generalizzate sono quindi in associazio-

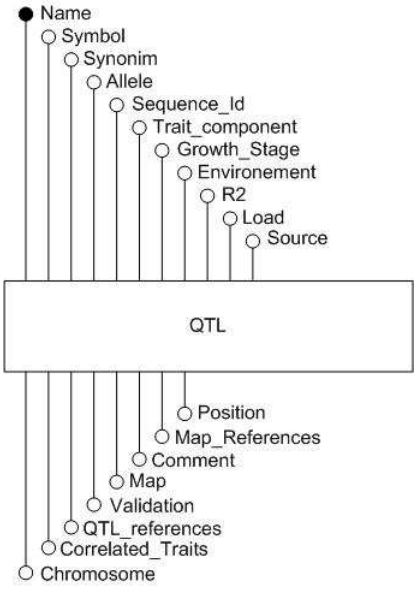


Figura 2.4: Attributi dell'entità Qtl

ne (1,N) rispettivamente con l'entità Gene e Qtl in quanto un marker può essere riferito a più di un gene o qtl. L'entità generica è in associazione di cardinalità (1,N) con l'entità Cultivar. Infatti un marker, sia esso riferito ad un gene o ad un qtl, può essere stato testato su una o più varietà di pianta. Gli attributi dell'entità Marker, comuni alle entità generalizzate Gene Maker e Qtl Marker, sono riportate in figura2.5.

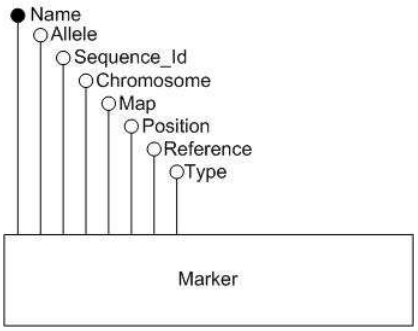


Figura 2.5: Attributi dell'entità Marker

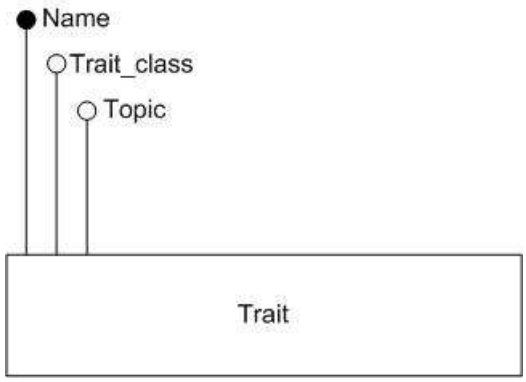


Figura 2.6: Attributi dell'entità Trait

Infine l'entità Trait rappresenta i caratteri fenotipici che può avere una pianta. Questa entità è in assciazione di cardinalità (1,N) con l'entità GQ, in quanto un trait può essere influenzato da uno o più geni o Qtl. Gli attributi delle'entità trait sono riportati in figura2.6.

2.2 Progetto Logico

Una volta realizzato lo schema concettuale secondo il modello Entity Relationship, si è proceduto alla progettazione logica del database.

Per prima cosa si è effettuata una ristrutturazione dello schema E/R per eliminare le due gerarchie presenti, GQ e Marker.

Per entrambe è stato scelto il collasso verso il basso in quanto solitamente tutte le entità generalizzate presenti verranno accedute separatamente. Il collasso verso il basso è possibile in quanto entrambe le gerarchie presenti hanno copertura totale.

L'entità generica GQ è stata quindi collassata verso il basso nelle due entità figlie Gene e Qtl, mentre l'entità generica Marker è stata collassata

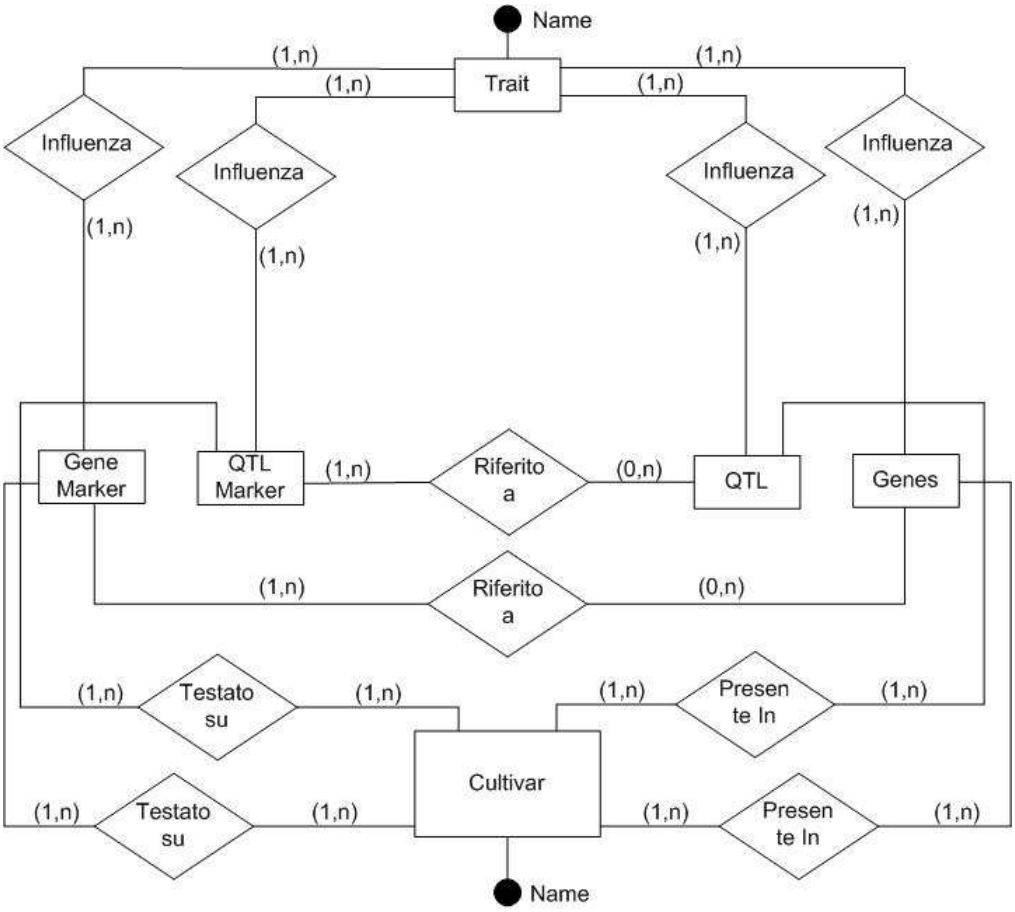


Figura 2.7: Schema E/R semplificato

verso il basso nelle due entità figlie Gene Marker e Qtl Marker. In entrambi i casi tutti gli attributi e le associazioni dell'entità padre vengono trasferiti sulle entità figlie. Nel caso dell'entità Marker, però, il collasso verso il basso introdurrà ridondanza dei dati in quanto la copertura non è esclusiva.

Lo schema E/R semplificato che si è ottenuto viene mostrato in figura 2.7.

La fase di progettazione logica è terminata con la semplificazione dello schema E/R. Non è stata effettuata una vera e propria traduzione di questo

schema in modello relazionale in quanto il database rappresentato nello schema non è stato realizzato fisicamente, ma per il momento è stata costruita una vista virtuale ottenuta dall'integrazione di database già esistenti, che approssima il database richiesto precedentemente descritto. La realizzazione di questa vista virtuale viene descritta con maggiore dettaglio nel capitolo 3.

Capitolo 3

Integrazione delle Sorgenti con Momis

Come accennato precedentemente, la realizzazione del database CereaLab è stata effettuata integrando due database sorgente, Gramene e Graingenes, già presenti sul Web. Per compiere questo processo di integrazione è stato utilizzato il sistema a mediatore Momis [1], realizzato dal DataBase Group dell'Università di Modena e Reggio Emilia. Tramite Momis è stato possibile creare una Vista Virtuale Globale (GVV) delle due sorgenti, che approssimasse lo schema del database progettato al capitolo 2. Vengono ora descritti brevemente l'architettura ed il funzionamento di Momis, per poi spiegare in dettaglio come è stata realizzata l'integrazione delle sorgenti dati.

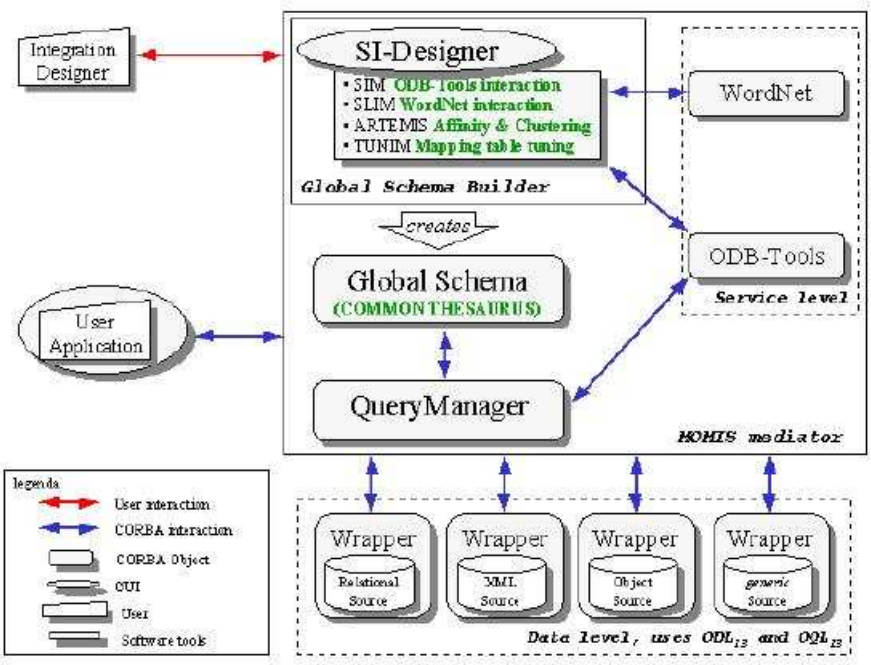


Figura 3.1: Architettura di MOMIS

3.1 Momis

Il Mediator EnvirOnment for Multiple Information Sources (MOMIS)[1], è un sistema per l'estrazione e l'integrazione intelligente di dati strutturati o semi-strutturati.

L'architettura del sistema viene presentata in figura 3.1.

Al livello più basso abbiamo i wrapper, che svolgono la funzione di interfaccia tra il mediatore e le sorgenti. Le operazioni fondamentali che vengono implementate da un oggetto wrapper sono due:

1. fornire la descrizione di una sorgente di dati e descriverne, quindi, la struttura;

2. porre una query alla sorgente nel suo linguaggio nativo e restituire i dati reperiti al sistema MOMIS.

A livello superiore troviamo invece il Mediatore di MOMIS, costitutito da due moduli:

- Il Global Schema Builder, per l'integrazione delle sorgenti e la generazione della Global Virtual View (GVV);
- il Query Manager che si occupa della scomposizione e ottimizzazione delle query da fornire ai wrapper che le applicheranno alle singole sorgenti.

MOMIS utilizza inoltre il tool WordNet, un database lessicale realizzato dal Cognitive Science Laboratory di Princeton. In WordNet sono raccolti sostantivi, aggettivi, avverbi e forme verbali inglesi, organizzati per gruppi di sinonimi, detti synset, che rappresentano determinati concetti lessicali.

L'idea su cui è costruita la semantica lessicale è che esiste una associazione tra la forma e delle parole e il loro significato, associazione che è di tipo molti a molti e che dà origine a due proprietà:

- sinonimia: proprietà di un significato di avere più parole che lo possano esprimere;
- polisemia: proprietà di una parola di poter esprimere molteplici significati.

WordNet collega i termini in base a relazioni semantiche tra synset. Le relazioni lessicali principali sono Sinonimia, Ipernimia, Meronimia e Correlazione.

La sinonimia è la relazione che stabilisce che due termini possono essere scambiati tra loro senza cambiare il significato di ciò che viene espresso.

L'ipernimia è una relazione di specializzazione tra due concetti, esprime un rapporto di tipo isa e gode delle proprietà tipiche dell'ereditarietà. Il suo inverso è l'ipernimia.

La meronimia è una relazione semantica fra due concetti x e y (x è meronimo di y) tali che x è una parte di y. La sua relazione duale è l'olonimia.

L'ultima relazione da considerare è la correlazione che si stabilisce tra due termini che condividono lo stesso ipernimo; se ne deduce quindi che è una relazione derivabile dalle altre.

3.1.1 Il Processo d'Integrazione

La prima parte del processo di integrazione tramite il sistema MOMIS consiste nell'annotazione. In questa fase viene assegnato alle classi e agli attributi che compaiono nelle sorgenti un significato rispetto ad una ontologia lessicale comune (WordNet).

Questa annotazione porta alla generazione del Thesaurus Comune in cui viene raccolta la conoscenza delle informazioni semantiche relative al contesto e alla struttura dei vari schemi sorgente. Per la costruzione del Thesaurus vengono utilizzate in primo luogo le relazioni tra le classi e i nomi degli attributi ottenute sfruttando le relazioni semantiche di WordNet. La sinonimia diviene quindi una relazione SYN (Synonym), l'iponimia diviene una relazione NT (Narrower Term), la meronimia e la correlazione diventato relazioni RT (Related Term).

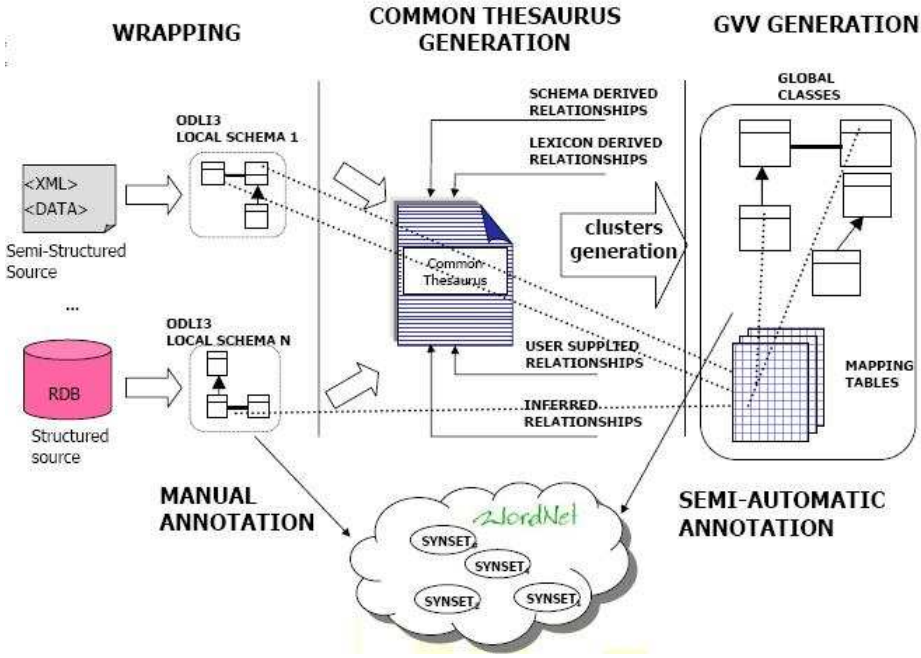


Figura 3.2: Il processo di Integrazione

Ulteriori relazioni vengono ricavate dallo schema delle sorgenti. Nel caso, per esempio, di sorgenti relazionali le foreign key diventano relazioni di tipo RT. Utilizzando inoltre tecniche di logica descrittiva vengono inferite ulteriori relazioni tra le classi locali.

Il Thesaurus Comune può essere inoltre arricchito direttamente dal progettista. In questa fase il progettista stesso può inserire nuove relazioni, non ricavate precedentemente, e migliorare il Thesaurus sulla base della sua personale conoscenza.

Terminata la costruzione del Thesaurus Comune si prosegue con il Calcolo delle affinità tra classi e termini, il cui risultato viene sfruttato per la formazione dei cluster in cui vengono raggruppate le classi; tutte quelle che hanno una affinità superiore a una soglia prestabilita vengono riunite in gruppo dal

mediatore, secondo tecniche di clustering.

Una volta che sono stati realizzati i cluster in cui sono state raccolte le classi locali, si passa alla fase di integrazione vera e propria degli schemi. Per ogni cluster realizzato si crea una classe globale caratterizzata da:

- un nome che fa da identificatore
- un insieme di attributi
- una mapping table che gestisce la corrispondenza tra gli attributi globali e i dati delle sorgenti locali

Dopo l'implementazione delle classi globali, il processo di integrazione prosegue con altre due fasi: la fusione degli attributi e la creazione della mapping table.

La fusione degli attributi nasce dall'esigenza di eliminare le ridondanze e di integrare completamente gli schemi. La modalità di fusione degli attributi all'interno di una classe globale è dipendente dal tipo di relazioni che li legano, e dal fatto che queste siano o meno validate.

Le mapping table sono le tabelle degli schemi globali che conservano le informazioni necessarie per passare dagli attributi globali a quelli locali, cioè ai dati veri e propri. Sono evidentemente uno strumento fondamentale soprattutto in fase di query processing. In una mapping table si ha una riga per ogni classe locale e una colonna per ogni attributo globale; gli elementi della tabella sono gli attributi locali che sono stati mappati in quelli globali.

3.2 Acquisizione e Ripristino delle Sorgenti

Il database CereaLab è stato ottenuto utilizzando due database sorgente disponibili in Internet. I due database sono Graingenes `http://wheat.pw.usda.gov/GG2/ggdb.shtml` riguardante orzo e grano, e Gramene `http://www.gramene.org` riguardante riso e mais.

Il sistema Momis permette di ottenere gli schemi delle sorgenti da integrare e di utilizzarli per creare una Global Virtual View risultante dall'integrazione delle sorgenti. Attraverso Momis è quindi possibile interrogare questa Vista Virtuale Globale. Il sistema si occupa di interrogare le sorgenti locali traducendo le query poste sulla GVV, e una volta ottenuti li integra per restituire una risposta conforme alla Vista Virtuale Globale.

Nel nostro caso però è stato necessario ottenere una copia delle sorgenti, in quanto attraverso il Web i due database non risultano interrogabili direttamente. In questa tesi viene descritto come è stato effettuato il ripristino del database Gramene, mentre il ripristino del database Graingenes viene descritto in [2].

Il sito Web di Gramene consente il download via ftp dei file di dump per il database managment system MySql all'indirizzo `//ftp.gramene.org/pub/gramene`. A questo indirizzo sono disponibili le diverse versioni del database, che vengono rilasciate a cadenza semestrale, sia sotto forma di file dump per MySql, sia sotto forma di file di testo contenenti gli script sql per la realizzazione del database e i dati su file separati. Per la realizzazione di questa tesi è stata utilizzata la versione 19 del database rilasciata a dicembre 2005.

Il sito mette a disposizione le diverse sezioni del database in file separati. Considerando le nostre necessità, sono stati quindi scaricati i file relativi alle sezioni riguardanti i fenotipi, i marker, i Qtl e le proteine. Il ripristino dei file dump riguardanti i fenotipi, i Qtl e le proteine non ha dato difficoltà utilizzando la versione 5.0 di MySql. Per quanto riguarda il ripristino della sezione del database riguardante i marker, invece, si sono riscontrati alcuni problemi a causa della grande dimensione di questo file. Questo file di dump è infatti di dimensione superiore ai 6 GB, pertanto il ripristino automatico è risultato irrealizzabile sulla macchina che è stata usata per questo lavoro di tesi. Per ricreare questa sezione del database è stato quindi utilizzato lo script sql disponibile sul sito di Gramene per la creazione delle tabelle, e successivamente sono state popolate le tabelle utilizzando i file dati sotto forma di file di testo.

Una volta ricreata la copia del database è stata esportata sul Dbms Sql Server 2000, che offre prestazioni migliori, data soprattutto la grande mole di dati contenuti. Esportando il database da MySql a Sql Server 2000 è stato necessario ricrostruire lo schema del database, ricreando tutte le chiavi e le foreign key del database. MySql infatti non supporta l'integrità referenziale, quindi tutte le foreign key mancavano nel database originale.

Come accennato precedentemente, il database è suddiviso in diverse sezioni. Queste sezioni risultano però tra loro disgiunte a livello relazionale. Per renderne possibile il collegamento, nel database presente sul Web è stata utilizzata una soluzione particolare, evidentemente dettata da altre scelte implementative. Questo collegamento viene infatti realizzato a livello applicativo in fase di interrogazione, sfruttando alcune particolari tabelle presenti

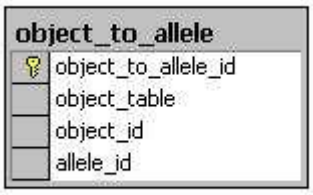


Figura 3.3: Tabella object_to_allele

nel database. Queste tabelle sono riconoscibili dal loro nome, che presenta un prefisso comune "object_to_". Un esempio di queste tabelle viene mostrato in figura 3.3.

Queste tabelle sono tabelle che esprimono associazioni tra una tabella data (nell'esempio allele) e un'altra tabella. Presentano un identificatore proprio (nell'esempio in figura: object_to_allele_id) , e altri 3 attributi: il primo, denominato "object_table", contiene il nome di una tabella del database; il secondo attributo, denominato "object_id", contiene la primary key della tabella indicata in "object_table"; l'ultimo attributo invece contiene la primary key della tabella a cui questa tabella referenzia. In questo modo queste tabelle vengono utilizzate per collegare più di una tabella (quelle indicate nell'attributo "object_table") ad un'altra singola tabella (tramite l'ultimo attributo, che contiene l'identificatore di questa tabella singola). Questo tipo di tabelle viene interrogato a livello applicativo per collegare sezioni altrimenti separate del database. Facendo riferimento all'esempio in figura, la tabella object_to_allele viene utilizzata per collegare le tabelle indicate nell'attributo object_table alla tabella allele.

Una soluzione di questo genere, che fa uso di tabelle di questo tipo, divie-

ne però inutilizzabile a livello relazionale, in quanto utilizzabile solo a livello applicativo. Per ovviare a questo problema è stata scritta un "stored procedure" per creare delle tabelle di associazione semplici e utilizzabili a livello relazionale eliminando l'attributo object_table. In questo modo ogni volta che viene fatto un update del database, per esempio nel caso di rilascio di una nuova versione, è sufficiente eseguire questa stored procedure per ripetere la trasformazione di queste tabelle. Viene ora riportata questa "stored procedure":

```
CREATE PROCEDURE dbo.set_object_to AS

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[allele_to_germplasm]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[allele_to_germplasm]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[allele_to_phenotype_expression]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[allele_to_phenotype_expression]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[allele_to_study]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1)
```

```
drop table [dbo].[allele_to_study]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[gene_product_to_cultivar]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[gene_product_to_cultivar]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[gene_product_to_embl_accession]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[gene_product_to_embl_accession]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[gene_product_to_gi_number]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[gene_product_to_gi_number]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[gene_product_to_keyword]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
```

```
drop table [dbo].[gene_product_to_keyword]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[gene_product_to_pid]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[gene_product_to_pid]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[gene_product_to_species]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[gene_product_to_species]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[map_position_to_evidence_code]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[map_position_to_evidence_code]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[mutant_to_allele]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1)
```

```
drop table [dbo].[mutant_to_allele]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[mutant_to_evidence_code]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[mutant_to_evidence_code]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[mutant_to_gene_product]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[mutant_to_gene_product]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[mutant_to_germplasm_info]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)

drop table [dbo].[mutant_to_germplasm_info]

GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[mutant_to_phenotype_expression]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
```

```
drop table [dbo].[mutant_to_phenotype_expression]

GO

CREATE TABLE [dbo].[allele_to_germplasm] (

[allele_to_germplasm_info_id] [int] NOT NULL ,

[allele_id] [int] NULL ,

[germpalsm_info_id] [int] NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[allele_to_phenotype_expression] (

[allele_to_phenotype_expression_id] [int] NOT NULL ,

[allele_id] [int] NULL ,

[phenotype_expression_id] [int] NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[allele_to_study] (

[allele_to_study_id] [int] NOT NULL ,

[allele_id] [int] NULL ,
```

```
[study_id] [int] NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[gene_product_to_cultivar] (

[gene_product_id] [int] NOT NULL ,

[cultivar_id] [int] NOT NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[gene_product_to_embl_accession] (

[gene_product_id] [int] NOT NULL ,

[embl_accession] [varchar] (30) COLLATE
SQL_Latin1_General_CP1_CI_AS NOT NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[gene_product_to_gi_number] (

[gene_product_id] [int] NOT NULL ,

[gi_number] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS
NOT NULL
```

```
) ON [PRIMARY]

GO

CREATE TABLE [dbo].[gene_product_to_keyword] (

[gene_product_id] [int] NOT NULL ,

[keyword_id] [int] NOT NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[gene_product_to_pid] (

[gene_product_id] [int] NOT NULL ,

[pid] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NOT
NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[gene_product_to_species] (

[gene_product_id] [int] NOT NULL ,

[species_id] [numeric](10, 0) NOT NULL

) ON [PRIMARY]
```

```
GO

CREATE TABLE [dbo].[map_position_to_evidence_code] (

[map_position_to_evidence_code_id] [int] NOT NULL ,

[map_id] [numeric](10, 0) NULL ,

[evidence_code] [varchar] (64) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[mutant_to_allele] (

[mutant_to_allele_id] [int] NOT NULL ,

[mutant_id] [int] NULL ,

[allele_id] [int] NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[mutant_to_evidence_code] (

[mutant_to_evidence_code_id] [int] NOT NULL ,

[mutant_id] [int] NULL ,
```

```
[evidence_code] [varchar] (64) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[mutant_to_gene_product] (

[mutant_to_gene_product_id] [int] NOT NULL ,

[mutant_id] [int] NULL ,

[gene_product_accession] [varchar] (32) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL

) ON [PRIMARY]

GO

CREATE TABLE [dbo].[mutant_to_germplasm_info] (

[mutant_to_germplasm_info_id] [int] NOT NULL ,

[mutant_id] [int] NULL ,

[germplasm_info_id] [int] NULL

) ON [PRIMARY]

GO
```

```
CREATE TABLE [dbo].[mutant_to_phenotype_expression] (

[mutant_to_phenotype_expression_id] [int] NOT NULL ,

[mutant_id] [int] NULL ,

[phenotype_expression_id] [int] NULL

) ON [PRIMARY]

GO

insert into allele_to_germplasm

select object_to_germplasm_info_id, object_id,
germplasm_info_id

from dbo.object_to_germplasm_info

where object_tadi relazione ble='allele'

GO

insert into allele_to_phenotype_expression

select ot_phenotype_expression_id, object_id,
phenotype_expression_id

from dbo.object_to_phenotype_expression

where object_table='allele'
```

```
GO

insert into allele_to_study

select object_to_study_id, object_id, study_id

from dbo.object_to_study

where object_table='allele'

GO

insert into map_position_to_evidence_code

select object_to_evidence_code_id, object_id, evidence_code

from dbo.object_to_evidence_code

where object_table='map_position'

GO

insert into mutant_to_allele

select object_to_allele_id, object_id, allele_id

from dbo.object_to_allele

where object_table='mutant'

GO

insert into mutant_to_evidence_code
```

```
select object_to_evidence_code_id, object_id, evidence_code

from dbo.object_to_evidence_code

where object_table='mutant'

GO

insert into mutant_to_gene_product

select object_to_gene_product_id, object_id,
gene_product_accession

from dbo.object_to_gene_product

where object_table='mutant'

GO

insert into mutant_to_germplasm_info

select object_to_germplasm_info_id, object_id,
germplasm_info_id

from dbo.object_to_germplasm_info

where object_table='mutant'

GO

insert into mutant_to_phenotype_expression
```

```
select ot_phenotype_expression_id, object_id,
phenotype_expression_id

from dbo.object_to_phenotype_expression

where object_table='mutant'

GO
```

Lo schema del database che è stato ottenuto viene riportato nelle figure 3.4,3.5 e 3.6.

Lo schema di questo database, insieme allo schema del database Graingenes, sono stati quindi usati quali schemi locali per la creazione della Vista Virtuale Globale del database CereaLab, come viene ora descritto.

3.3 Integrazione delle Sorgenti

Una volta ripristinate le copie locali e ricreati gli schemi dei database sorgente, si è potuto utilizzare il sistema Momis per creare la Vista Virtuale Globale che, sfruttando i dati contenuti nei due database esistenti, ricreasse un database avente le stesse classi e attributi di quello che è stato descritto nel capitolo 2. Vengono ora descritti in dettaglio i passi che hanno portato alla realizzazione di questa Global Virtual View.

3.3.1 Caricamento delle Sorgenti

In figura viene mostrata la schermata iniziale di Momis, dove è possibile selezionare le sorgenti da caricare e soprattutto è necessario indicare il wrapper

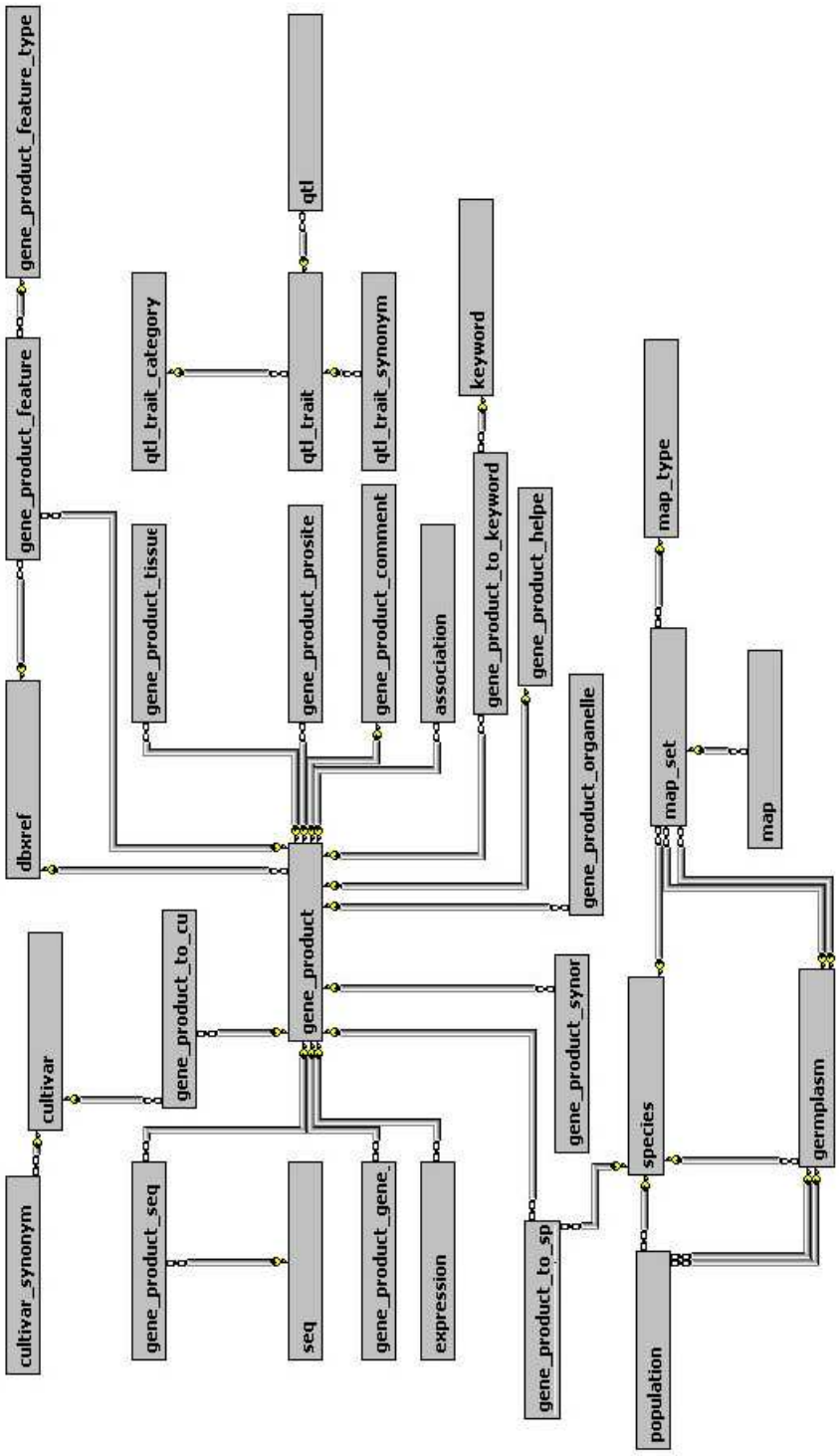


Figura 3.4: Schema sezione Gene_Product e Qtl

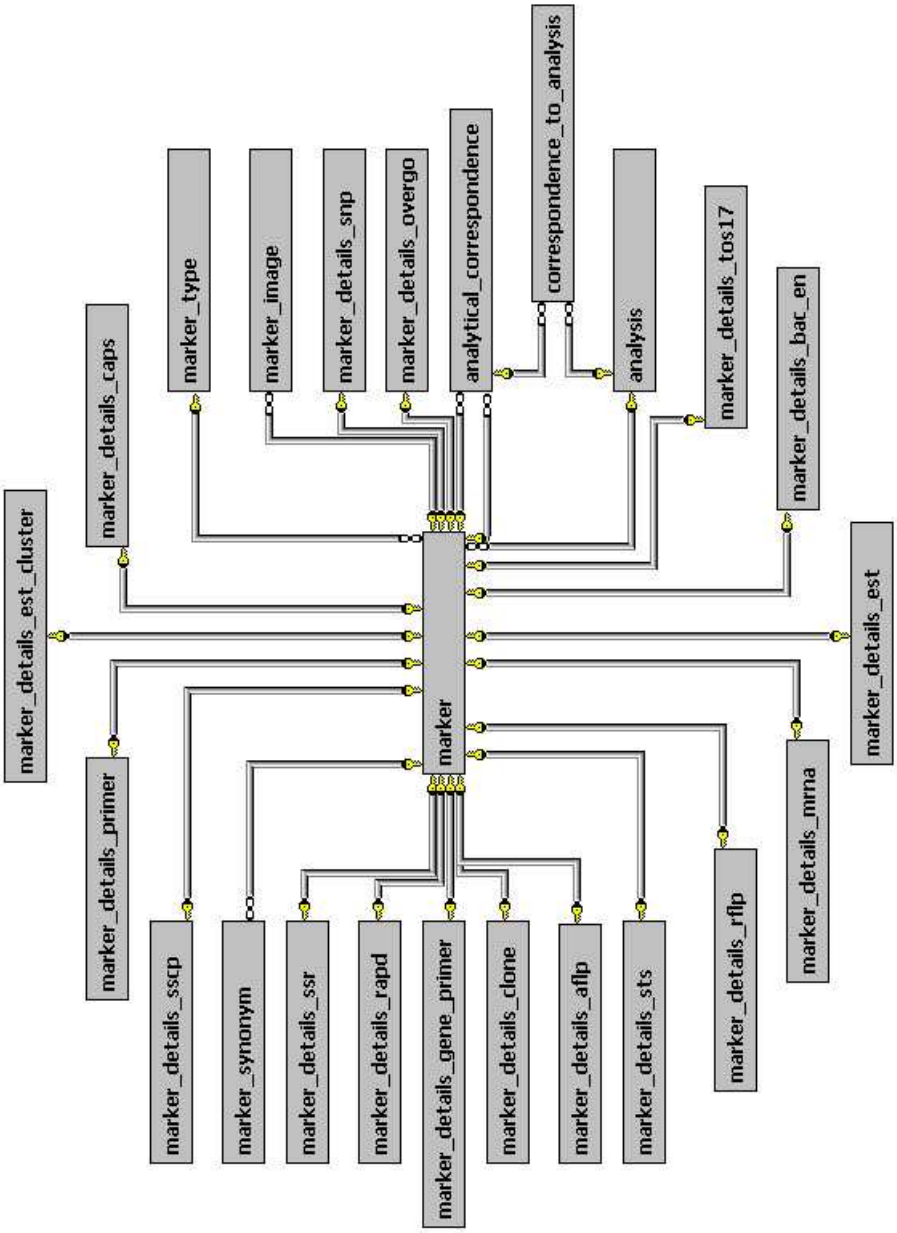


Figura 3.5: Schema sezione Marker

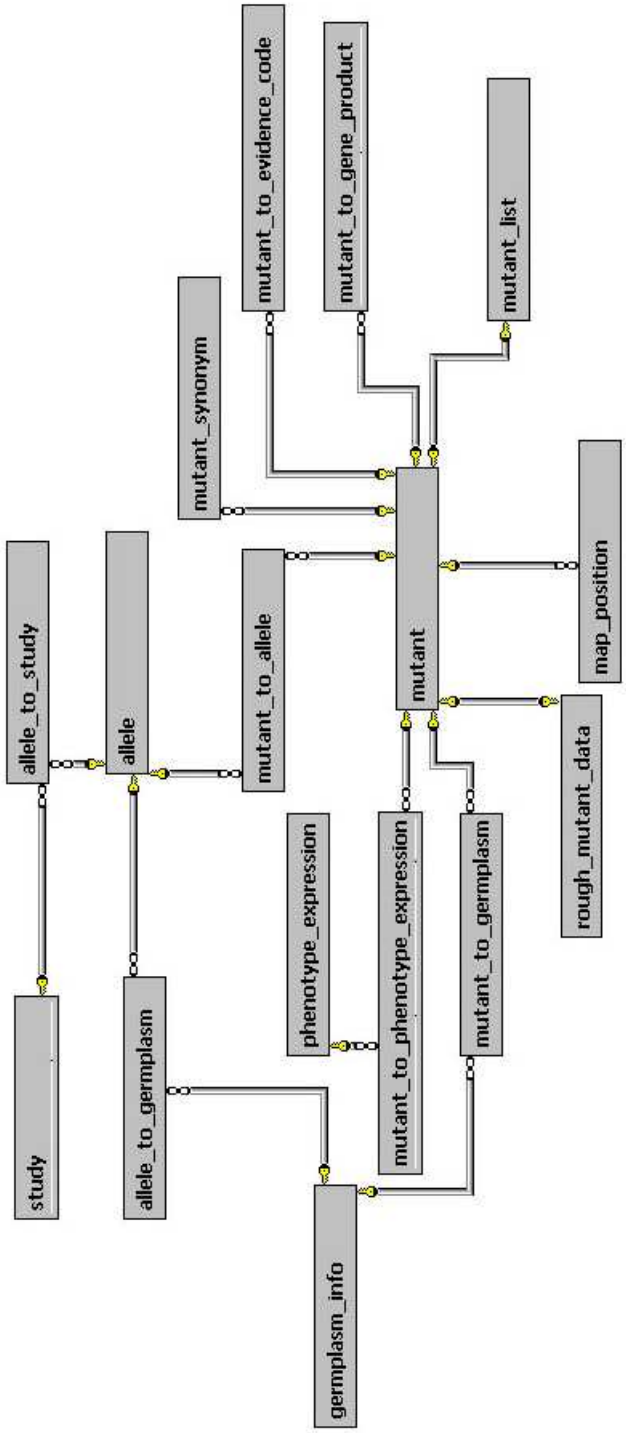


Figura 3.6: Schema sezione Phenotype

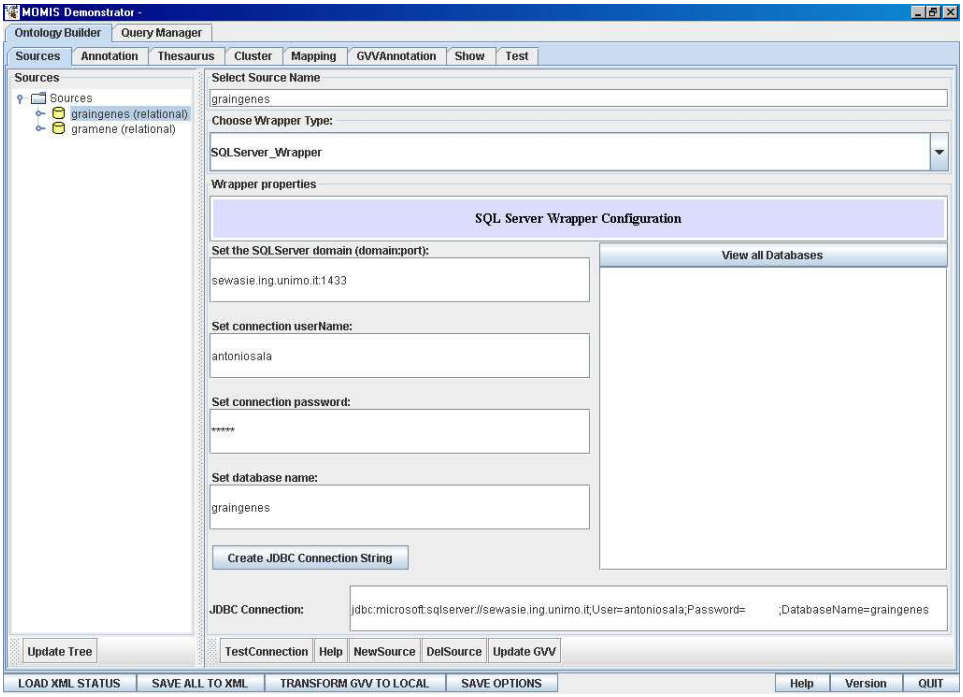


Figura 3.7: Interfaccia di Momis

da utilizzare per ottenere le informazioni sulle sorgenti. Nel nostro caso è stato utilizzato per entrambe le sorgenti il wrapper per Sql Server. Le altre informazioni da fornire sono il nome del server su cui è presente la sorgente, la relativa porta e i parametri per connettervisi. Inoltre viene richiesto di fornire un nome alla sorgente.

3.3.2 Annotazione e Generazione del Thesaurus

Una volta caricati gli schemi relativi alle due sorgenti, la prima fase consiste nell'annotazione dei nomi delle classi e degli attributi che compaiono nei database sorgente. Momis consente di realizzare questa annotazione in modo semi-automatico, sfruttando il database WordNet. Nel nostro caso però la

fase di annotazione è risultata abbastanza laboriosa in quanto il dominio su cui si lavorava è un dominio molto specifico e quindi in WordNet non erano presenti la maggior parte dei termini coinvolti. In questa fase quindi si è dovuto arricchire ampiamente il database lessicale WordNet, con l'aiuto del gruppo del Prof.Pecchioni che ci ha fornito indicazioni e spiegazioni specifiche riguardati il dominio in questione, utilizzando il WordNet Editor messo a disposizione da Momis.

Sfruttando le informazioni che vengono fornite in fase di annotazione, oltre a quelle che vengono ricavate dallo schema del database, Momis genera un Thesaurus comune ricavando quindi relazioni lessicali e relazioni derivate dallo schema delle sorgenti. In figura 3.8 viene mostrato un esempio delle relazioni ricavate da Momis per le classi Trait e Gene delle due sorgenti Gramene e Graingenes. Come si può vedere nella parte centrale vengono indicati i termini che partecipano alla relazione nelle colonne Source e Destination, separati dalla colonna Type che indica il tipo di relazione che li lega: SYN, BT, NT, RT. Dopo questi elementi si trova un'ulteriore colonna in cui viene indicato, per mezzo di un codice e di un colore, da cosa è stata ricavata la relazione:in grigio vengono indicate le relazioni ricavate dallo schema, in blu quelle lessicali, in rosso quelle inferite mediante tecniche di logica descrittiva. Infine sulla destra viene riportata una colonna che indica se la relazione risulta o meno valida (se i tipi degli elementi sono compatibili).

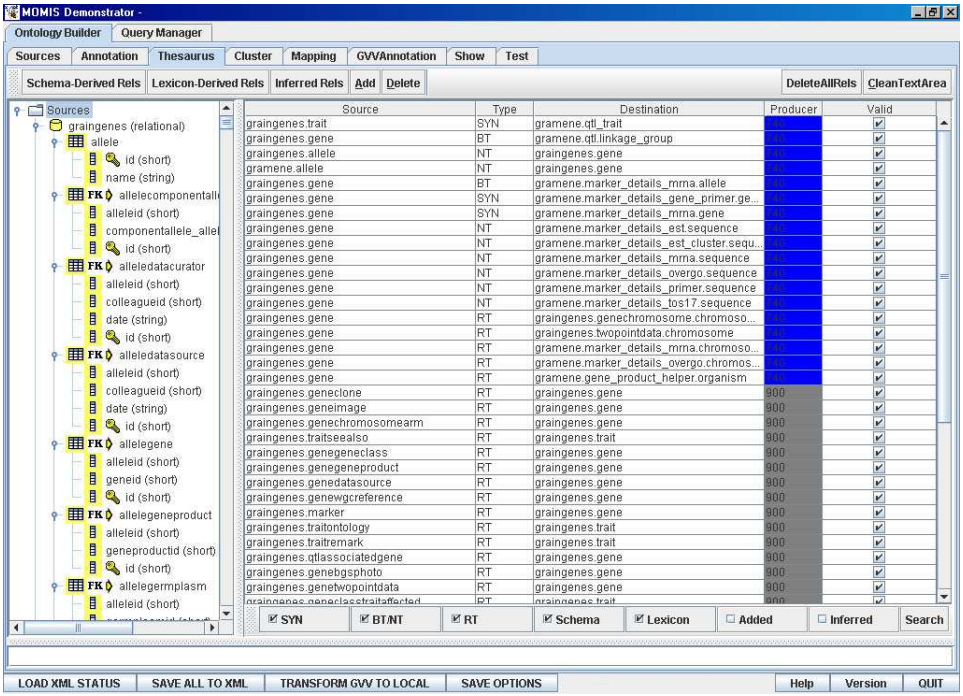


Figura 3.8: Thesaurus: relazioni rigurdanti Gene e Trait

3.3.3 Clustering

Una volta generato il Thesaurus, le relazioni ivi contenute vengono sfruttate da Momis per la realizzazione dei cluster in cui raggruppare le classi locali dei vari schemi che devono essere integrati.

Nel nostro caso si è dovuto in parte adattare i cluster che erano stati generati automaticamente per ricreare le classi globali che rappresentassero le entità desiderate descritte nel capitolo 2. Inoltre alcune informazioni erano presenti eslusivamente nel database Graingenes, mentre altre erano presenti solamente in Gramene. Sono state quindi generate le classi globali `Gene`, `Germplasm`, `Qtl`, `Trait` e `Marker`. Le informazioni sui geni vengono ottenute esclusivamente dalla sorgente Graingenes, mentre le informazioni sui marker provengono esclusivamente da Gramene. Per quanto riguarda i germplasm, sono state create due classi globali distinte ciascuna contenente le informazioni di una sola sorgente, in quanto i due database contengono dati di specie differenti, quindi l'integrazione dei dati sui germplasm non avrebbe avuto senso, anche se logicamente simili. Lo stesso discorso vale per i qtl ad essi associati. Sono state quindi create le classi globali `germplasm_gramene` e `germplasm_graingenes`, e le classi `qtl_graingenes` e `qtl_gramene`. La classe globale trait invece è il risultato dell'integrazione delle classi locali delle due sorgenti, in quanto i trait che vengono definiti sono comuni alle quattro specie trattate dai due database. In figura 3.9 viene mostrata la finestra di Momis per la manipolazione dei Cluster dove vengono mostrati i cluster riguardanti i qtl e i trait.

In tutti i cluster sono state inoltre unite alcune classi locali della stessa

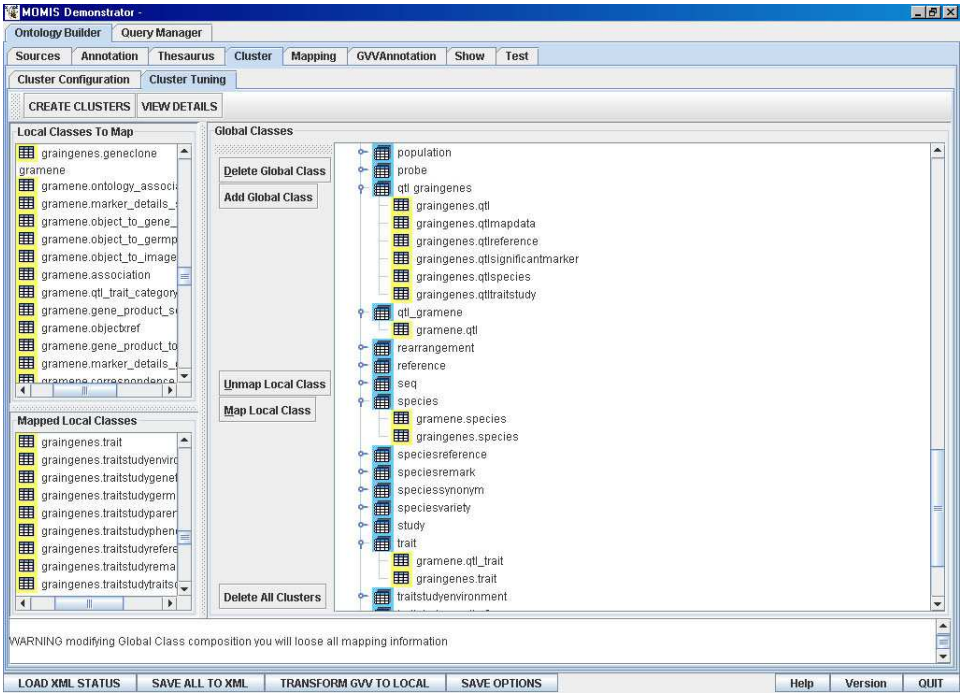


Figura 3.9: Cluster

sorgente per ricavare degli attributi che nelle sorgenti sono distribuiti su più classi, mentre nel nostro database si voleva che fossero raggruppati nella stessa classe. Come si può vedere nella figura 3.9 le classi `qtl_graingenes` e `qtl_gramene` contengono al loro interno diverse classi locali della stessa sorgente per questo motivo.

La selezione degli attributi di ogni classe globale viene invece realizzata nella fase di mapping, descritta nella parte seguente.

3.3.4 Mapping

Nella fase di mapping vengono create le mapping table che consentono di controllare le relazioni tra gli attributi globali delle classi e gli attributi locali

delle sorgenti, contenenti i dati.

In questa fase sono stati rimossi dalle classi globali sopracitate gli attributi che erano presenti sui database sorgente che riportavano informazioni non richieste. In figura 3.10 viene presentata la finestra di Momis nella quale è possibile controllare e modificare il mapping. Nella parte in basso viene visualizzata la mapping table che mostra le corrispondenze tra gli attributi globali e gli attributi locali. Più precisamente nella figura viene mostrato il mapping realizzato (in questo caso in modo completamente automatico) per la classe globale `Trait`.

Vengono ora riportati gli attributi globali dei principali cluster ottenuti:

- `Interface [globalSource.gene]`

```
chromosome

fullname
geneid
locusid
name
pathologyid
referenceid
referenceid_1
remark
sequenceid
type
```

- `Interface [globalSource.germplasm_graingenes]`

```
bandsize
chromosomeconfiguration
chromosomedonor
```

```
chromosomenumber
cytoplasm
germplasmid
libraryid
mapdataid
name
pathologyid
polymorphismid
referenceid
remark
remark_type
species_type
speciesid
traitstudyid
```

- Interface [globalSource.germplasm_gramene]

```
description
genus
germplasm_id
germplasm_info_id
germplasm_name
mutagenesis_id
species_id
subspecies_id
wild_type
```

- Interface [globalSource.qtl_graingenes]

```
chromosomearm
locusid
mapdataid
maplabel
name
nearestmarker_locusid
qtlid
referenceid
significancelevel
```

speciesid
traitaffected_traitid
traitstudyid

- Interface [globalSource.qtl_gramene]

comments

linkage_group
published_symbol
Qtl_accession_id
Qtl_id
Qtl_trait_id
species
start_position
stop_position

- Interface [globalSource.trait]

graingenes_trait_id
gramene_trait_id
Qtl_trait_category_id
to_accession
trait_name
trait_symbol

- Interface [globalSource.marker]

EC_number

chromosome
comment
description
dev_stage
gene
germplasm_id
keyword

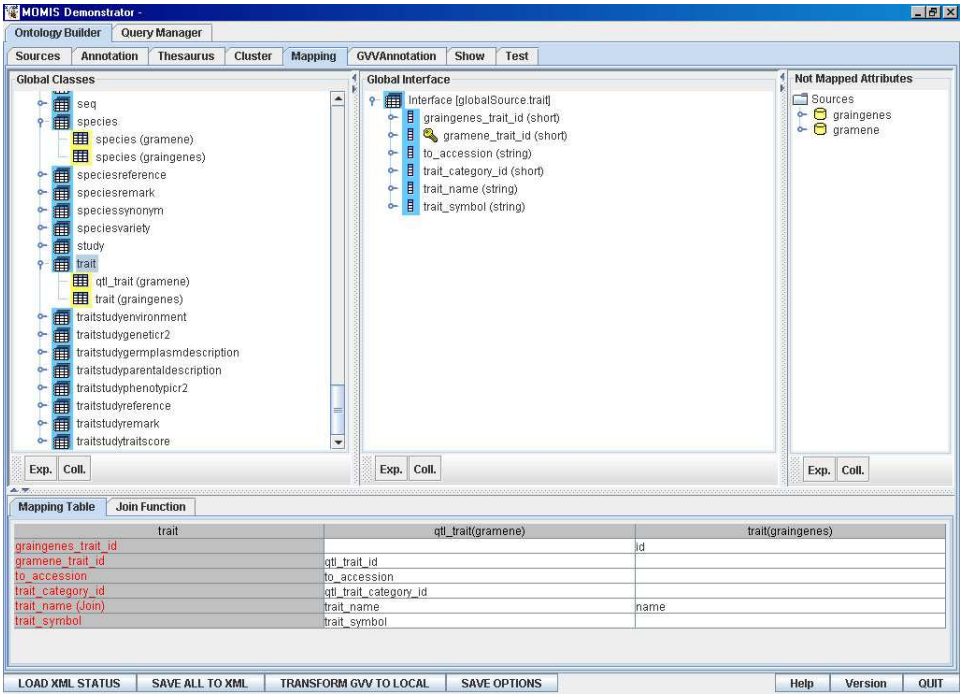


Figura 3.10: Mapping della classe `Trait`

```
label
map
marker_id
marker_type_id
note
phenotype
rearranged
ref_authors
ref_location
ref_title
ref_year
sequence
sequence_length
sex
standard_name
tissue_type
```

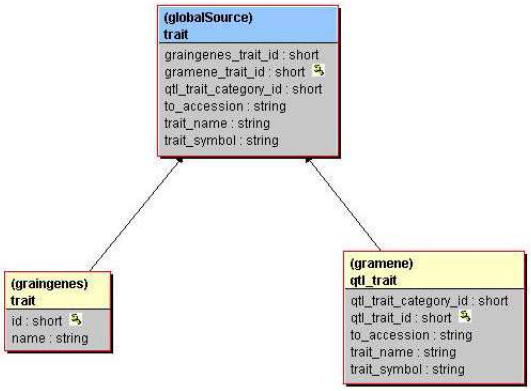


Figura 3.11: Visualizzazione della Classe Globale `Trait`

3.3.5 Global Virtual View

Una volta effettuato il mapping, le classi globali ottenute realizzano una astrazione delle sorgenti locali che compongono la nostra Gloabal Virtual View.

Momis prevede anche una fase di annotazione della GVV, in cui è possibile assegnare, in modo semi-automatico, ad ogni classe globale e a ciascun attributo globale un nome ed un significato rispetto all'ontologia lessicale comune. Nel nostro caso specifico l'annotazione effettuata in modo automatico dal sistema è risultata coerente e non ha richiesto correzioni.

Momis mette a disposizione inoltre una finestra nella quale è possibile visualizzare il mapping che è stato realizzato delle singole classi locali nelle classi globali, per averne una comprensione più immediata. Un esempio, sempre raffigurante la classe globale `Trait`, viene presentato in figura 3.11.

Queste classi globali possono essere interrogate utilizzando l'interfaccia

Query Manager di Momis. Il sistema si occupa di tradurre le query formulate sulla Global Virtual View in query da sottoporre alle relative classi locali, e di integrare i risultati ottenuti in modo dalle sorgenti.

3.4 Considerazioni sulla Vista Virtuale Ottenuta

La Global Virtual View ottenuta risulta una buona approssimazione del database che era stato progettato nel capitolo 2. Il mapping che è stato realizzato consente infatti di ottenere molte informazioni che nelle sorgenti sono distribuite su diverse classi locali interrogando una sola classe globale. Le classi globali corrispondono infatti alle entità che erano state modellate nello schema E/R, se consideriamo che le informazioni relative alle cultivar vengono trovate nella classi globali relative ai germplasm. L'unica differenza sostanziale consiste nelle relazioni tra le diverse entità, in quanto nella nostra Global Virtual View non risultano immediate, ma bisogna utilizzare le classi locali presenti nelle sorgenti per realizzare queste relazioni. Essendo però le classi locali molto numerose, è necessaria una buona conoscenza delle sorgenti stesse per ricavarne le relazioni tra le diverse classi. Per evitare l'eccessiva parcellizzazione presente negli schemi delle sorgenti, basterebbe poter realizzare delle viste su questi schemi, per poter rendere più agevole l'interrogazione dei database. Le viste però non sono supportate da Momis, quindi questa soluzione non è al momento realizzabile.

Alcune informazioni invece, come ad esempio l'associazione tra i geni

ed i trait influenzati da questi geni, non sono invece ottenibili in quanto non presenti nei database sorgente. Queste informazioni verranno aggiunte in seguito quando saranno ottenute in modo sperimentale dal laboratorio CereaLab.

Capitolo 4

Estrazione Automatica dei Dati dal Web

4.1 Utilizzo di Wrapper per l'Estrazione Dati

Un metodo alternativo per il reperimento dei dati per popolare il database potrebbe essere l'utilizzo di wrapper per l'estrazione dei dati direttamente da pagine Web contenenti informazioni di interesse. Un wrapper è infatti un modulo software in grado di compiere l'estrazione dei dati da una sorgente Html.

All'interno del dominio di cui ci stiamo occupando sono infatti numerosi i dati disponibili sul Web. La maggior parte di queste fonti sono però state progettate per essere consultate dagli utenti, e risulta quindi difficile riuscire ad estrane in maniera automatica i dati. Solitamente questa estrazione può essere compiuta dai wrapper, i quali però devono essere progettati ad hoc per una particolare risorsa Web. La specificità dei wrapper per una sola sorgente

dati che deriva dalla progettazione ad hoc risulta però poco vantaggiosa, in quanto sia la sua realizzazione manuale sia il suo mantenimento sono operazione particolarmente laboriose.

Per semplificare l'estrazione dei dati da siti Web definiti data-intensive, ovvero siti HTML che contengono grandi quantità di dati con una struttura abbastanza regolare, si stanno studiando diverse tecniche per la generazione automatica di wrapper. Uno strumento del genere risulta molto utile nel nostro caso, in quanto consentirebbe di estrarre le grandi quantità di dati presenti sul Web per popolare il nostro database e facilitarne quindi la consultazione.

Solitamente però queste tecniche per la generazione automatica dei wrapper presentano alcune caratteristiche comuni:

1. La generazione del wrapper richiede di fornire informazioni aggiuntive, solitamente un insieme di esempi realizzati dall'utente o da qualche tool esterno; il wrapper viene quindi inferito utilizzando questi esempi e cercando di generalizzarli.

2. Il sistema per la generazione del wrapper è a conoscenza della organizzazione dei dati nella pagina; la maggior parte delle tecniche assume che le pagine da cui verrà effettuata l'estrazione presentano i dati in forma non innestata; in altri casi, qualora il sistema sia in grado di gestire dati innestati, richiede di indicare quali attributi devono essere estratti e come sono innestati.

3. Infine, il wrapper viene generato esaminando una pagina HTML alla volta.

Di seguito viene invece presentato strumento, RoadRunner, con il quale è possibile generare in modo completamente automatico un wrapper a partire da un insieme di pagine HTML.

4.2 RoadRunner

RoadRunner[3] è uno strumento progettato per automatizzare la generazione di un wrapper ed il processo di estrazione dei dati da una sorgete HTML. La tecnica utilizzata consiste nel confrontare un insieme di pagine HTML per generare un wrapper sulla base delle similarità e delle differenze presenti in queste pagine.

Il processo di generazione di un wrapper per un insieme di pagine HTML corrisponde ad inferire una grammatica regolare per il codice HTML, e quindi all'utilizzo di questa grammatica per compiere l'analisi logica dela pagina sorgente e quindi estrarne i dati contenuti.

Le pagine dei siti Web data-intensive sono solitamente generate automaticamente utilizzando i dati presenti in un Database Managment System (DBMS) sottostante. La generazione di una pagina è quindi il risultato di due attività: in primo luogo, vengono eseguite una serie di query sul database per generare un dataset contenente i dati che sono stati richiesti, i quali vengono in un secondo momento inseriti nel codice HTML che produrrà la pagina, eventualmente aggiungendo link URL, immagini o altro. Si può quindi generalizzare che le pagine generate dallo stesso codice HTML presentino le stesse caratteristiche.

RoadRunner sfrutta quindi il fatto che le diverse pagine abbiano la stessa

stessa struttura, per confrontare il codice HTML di queste pagine ed inferirne quindi la struttura comune ed il wrapper da utilizzare per estrarne i dati, o dataset.

Il processo di generazione della pagina di un sito Web data-intensive può essere visto come un processo di codifica dei dati contenuti nel database sorgente all'interno di stringhe HTML; l'estrazione dei dati da queste pagine può quindi essere visto come il processo inverso di decodifica. Tale problema può essere quindi formalizzato come il problema di identificare i tipi di istanze innestate, codificate come stringhe al fine di essere presentate nella pagina, e la conseguente decodifica delle istanze originali dalla loro versione codificata.

La soluzione di questo problema è basata sulla corrispondenza tra i dati innestati e le espressioni regolari union-free. Utilizzando uno speciale simbolo $\#PCDATA$, ed un alfabeto di simboli Σ, una espressione regolare union-free ($UFRE$) su Σ è una stringa costruita sull'alfabeto $\Sigma \cup \{\#PCDATA, \cdot, +, ?, (,)\}$ definita come segue: per prima cosa, la stringa vuota, ϵ, e tutti gli elementi di $\Sigma \cup \{\#PCDATA\}$ sono espressioni regolari union-free. Se a e b sono $UFRE$, allora anche a·b, $(a)^+$, e (a)? sono $UFRE$. La semantica di queste espressioni viene definita intendendo + come iteratore e (a)? come abbreviazione di $(a|\epsilon)$ per indicare una sequenza opzionale. L'insieme delle espressioni regolari union-free si presta bene quindi in questo ambito in quanto può essere utilizzata per rappresentare i dati innestati, dove con il simbolo $\#PCDATA$si possono indicare i campi dati, con + si possono indicare le liste, eventualmente innestate, ? può invece rappresentare i campi che possono assumere valore NULL. In [9] viene dimostrato che, data una UFRE$UFRE$ σ, il corrispondente tipo innestato, τ, può essere ricavato in

un tempo lineare.

Si può quindi dimostrare che dato un insieme di stringhe HTML, corrispondenti ad un dataset sorgente, cioè un insieme di istanze di un tipo innestato τ, si può ricavare il tipo τ inferendo l'espressione regolare union-free minima σ il cui linguaggio $L(\sigma)$contiene le stringhe HTML originali. Inoltre, si può usare σ come wrapper per effettuare il parsing di queste stringhe e ricavare quindi il dataset. Quindi il processo di estrazione dei dati corrisponde a trovare la minima $UFRE$ il cui linguaggio comprenda le stringhe HTML in input. Se condideriamo una serie di $UFRE$ tra loro in relazione per cui $\sigma_1 \leq \sigma_2$ se e solo se $L(\sigma_1) \leq L(\sigma_2)$, allora la $UFRE$ che cerchiamo è il Least Upper Bound delle stringhe in input. Dato che l'operatore LUB è associativo, quanto detto prima corrisponde quindi a ricavare il least upper bound delle espressioni regolari union-free σ_1 e σ_2.

Il processo di estrazione dei dati può quindi essere risolto calcolando iterativamente il least upper bound del reticolo per generare un wrapper comune per la pagina HTML in input. Risulta quindi chiaro che risolvere il problema dell'estrazione dei dati consiste nel trovare un algoritmo per il calcolo del least upper bound di due $UFRE$.

Viene ora presentato l'algoritmo di matching implementato da RoadRunner per il calcolo del least upper bound di due $UFRE$.

4.2.1 Tecnica per la Generazione del Wrapper in RoadRunner

Per generare un wrapper sulla base di un set di pagine HTML sorgente, RoadRunner si basa su di un algoritmo di match per il calcolo del Least Upper Bound di due espressioni regolari union-free.

Per evitare errori o tag mancanti nella pagina in input, si presuppone che il codice HTML della pagina sia conforme alle specifiche XHTML, una variante restrittiva dell'HTML nella quale i tag devono necessariamente essere chiusi e innestati appropriatamente. Tale assunzione può essere fatta senza perdere di generalità in quanto sono disponibili numerosi tool per converti una qualunque pagina HTML in una conforme alle specifiche XHTML. Si assume anche che le sorgenti siano state preprocessate e trasformate in liste di token; ogni token può quindi essere un tag HTML oppure un valore stringa.

L'algoritmo di matching processa due oggetti alla volta: la lista di token, che viene chiamato sample; e un wrapper, ovvero una espressione regolare union-free. Per iniziare, date due pagine HTML in input, una di queste viene considerata come versione iniziale del wrapper; tale wrapper viene quindi progressivamente raffinato cercando di trovare una espressione regolare comune alle due pagine. Questa fase avviene risolvendo le differenze, o mismatch, che vengono trovate tra il wrapper ed il sample.

L'algoritmo di match consiste nel compiere il parsing del sample utilizzando il wrapper. Si ha un mismatch quando qualche token del sample non risulta conforme alla grammatica specificata nel wrapper. I mismatch sono quindi molto importanti in quanto permettono di ricavare informazioni mol-

to importanti per il wrapper. Ogni volta che viene incontrato un mismatch infatti, si cerca di risolverlo generalizzando il wrapper. Il processo di generazione del wrapper ha quindi successo se si riesce a generare un wrapper comune risolvendo tutti i mismatch incontrati durante il parsing.

Particolare attenzione va quindi dedicata ai mismatch. Vengono riconosciuti essenzialmente due tipi di mismatch: lo string mismatch, ovvero il mismatch che avviene quando si incontrano stringhe differenti nella medesima posizione del wrapper e del sample; il tag mismatch, ovvero i mismatch tra tag diversi nel wrapper e nel sample, oppure tra un tag e una stringa.

Lo string mismatch viene generato solamente da valori differenti di un campo dati tra due pagine appartenenti alla stessa classe. Questi mismatch vengono quindi usati per riconoscere i campi, ovvero i $\#PCDATA$. Per risolvere questo tipo di mismatch viene semplicemente gerealizzato il wrapper per identificare il campo appena identificato. La stringa dati viene quindi sostituita nel wrapper dal termine$\#PCDATA$. Da notare il fatto che stringhe costanti nelle due pagine non danno origine a campi dati nel wrapper, ma vengono semplicemente considerate come informazioni addizionali che non portano contenuto informativo utile.

Il tag mismatch può invece essere utilizzato per identificare iteratori o parti opzionali. In presenza di mismatch di questo tipo, l'algoritmo di match di RoadRunner ricerca in prima analisi la presenza di pattern ripetuti e, qualora questa ricerca non vada a buon fine, prova ad identificare pattern opzionali.

I pattern ripetuti, chiamati square, sono riconducibili alla diversa cardinalità dei campi dati presenti nelle diverse pagine. Risolvere questi mismatch

significa quindi identificare i pattern che riportano tali dati con cardinalità maggiore di uno, e quindi generalizzare il wrapper di conseguenza. Il processo di identificazione di questi pattern viene portato a termine in tre passi principali:

1. In seguito ad un tag mismatch potremmo essere di fronte ad un iteratore (+), quindi sia il wrapper che il sample dovrebbero contenere almeno una occorrenza di un pattern potenzialmente ripetuto. Chiamando o_W e o_S il numero di occorrenze di un campo dati iterato nel wrapper e nel sample, rispettivamente, si può concludere che prima di incontrare il mismatch la prima occorrenza $\min(o_W,o_S)$ è stata identifcata. Di conseguenza possiamo identificare l'ultimo token dello square cercando il token immediatamente precedente alla posizione dove avviene il mismatch. Questo tag viene definito tag terminale. Inoltre, dal momento che il mismatch corrisponde alla fine della lista su un sample e l'inizio di una nuova occorrenza dello square su un altro, abbiamo anche un indizio sul tag iniziale dello square. In ogni modo, non sappiamo esattamente dove si trovi la lista con cardinalità maggiore, se sul sample o sul wrapper; per questo motivo non sappiamo quale tra i token mismatch corrisponde al tag iniziale. Dobbiamo quindi valutare due possibilità: potenziali square sul wrapper che non sono square reali; oppure potenziali square nel sample. Vengono quindi valutate entrambe le possibilità prima sul wrapper e dopo sul sample.

2. Per verificare che l'occorrenza che potenzialmente rappresenta uno square lo idetifichi realmente, si cerca di abbinare lo square candidato con

una qualche porzione precedente del sample. La ricerca ha successo se si riesce a trovare un match per l'intero square.

3. A questo punto è quindi possibile generalizzare il wrapper. Se indichiamo lo square appena trovato con s, si cercano nel wrapper le occorrenze contigue di s intorno alla posizione che ha originato il mismatch e le si sostituiscono con $(s)^+$.

Uno volta che si è esclusa la presenza di un iteratore, si prova a risolvere il tag mismatch cercando di identificare un pattern opzionale. In questo caso sul wrapper oppure sul sample si è in presenza di una parte di codice HTML che non è presente nell'altra sorgente, e pertanto evitando di considerare quella parte di codice dovremmo essere in grado di riprendere il parsing. Questo processo viene effettuato in due passi:

1. Si procede ad una ricerca incrociata tra le due sorgenti per identificare il pattern opzionale: assumendo per esempio che il pattern sia nel wrapper, saltando quel pattern sul wrapper dovrebbe essere possibile associare il tag presente sul sample con il tag successivo al tag ritenuto opzionale nel wrapper. La stessa valutazione può essere fatta assumendo che il pattern opzionale sia quello incontrato sul sample. Una semplice ricerca incrociata di questo genere può quindi permettere di identificare il pattern opzionale.

2. Una volta che il pattern opzionale è stato identificato si può procedere alla conseguente generalizzazione del wrapper e riprendere il parsing.

Il numero di mismatch da risolvere potrebbe essere molto grande, e possono pertanto presentarsi alcuni problemi, soprattutto riguardo ai tag mismatch.

L'algoritmo per la risoluzione dei mismatch infatti risulta intrinsecamente ricorsivo, in quanto dalla soluzione di un mismatch possono venire generati altri mismatch. Per esempio quando si cerca di risolvere un tag mismatch cercando un possibile iteratore, prima si prova a localizzare una potenziale occorrenza dello square nel wrapper, poi si cerca un possibile match per questo pattern nella porzione precedente del wrapper. La ricerca viene fatta all'indietro, iniziando a confrontare le occorrenze dei due tag terminali. Questa ricerca dà quindi origine a nuovi mismatch, chiamati mismatch interni dovuti alla struttura innestata della pagina. I mismatch interni vengono quindi affrontati nello stesso modo in cui vengono risolti i mismatch esterni, implicando quindi la natura ricorsiva dell'algoritmo di match. L'unica differenza consiste nel fatto che i mismatch interni non vengono risolti confrontando il wrapper con il sample, ma due porzioni differenti dello stesso oggetto.

Un altro problema da affrontare è causato dalla necessità di dover scegliere tra diverse possibili alternative per la risoluzione di un mismatch, le quali però non è sicuro che conducano ad una soluzione corretta. Quindi, quando proseguendo nel parsing le scelte fatte si rivelano errate, si rende necessario ripercorrere il procedimento a ritroso per poter riprendere il parsing scegliendo un'altra alternativa.

Tendendo in considerazione le riflessioni fatte fino ad ora, si può quindi provare a spiegare più in dettaglio l'algoritmo di match per la generazione del wrapper. Trovare una soluzione all'algoritmo match a cui sono state

sottoposte due pagine, il wrapper w e il sample s, corrisponde a trovare un percorso per visitare un albero AND-OR. Questo albero è una rappresentazione delle possibili scelte da fare per risolvere tutti i mismatch incontrati durante il parsing (nodi AND); risolvere ognuno di questi mismatch corrisponde a sua volta a visitare un sotto-albero AND-OR; infatti il mismatch può essere risolto introducendo un campo dati, oppure un iteratore, oppure un pattern opzionale (nodo OR); la ricerca può essere fatta sul wrapper oppure sul sample (OR); sia gli iteratori sia i pattern opzionali presentano solitamente diverse alternative da valutare (OR); per scoprire la presenza di iteratori può essere necessario risolvere iterativamente numerosi mismatch interni (AND), ognuno dei quali corrisponde ad un nuovo sottoalbero AND-OR. Se si riesce a risolvere questo albero AND-OR, si ottiene in output il wrapper w'. Altrimenti il risultato è ϵ.

Questo algoritmo match può essere di tipo esponenziale rispetto ai dati in ingresso a causa della dimensione esponenziale del problema di esplorare un albero AND-OR. Per limitare la complessità del match vengono quindi adottate alcune tecniche di pruning per evitare di visitare sottoalberi che molto probabilmente non porteranno a soluzioni significative. Per esempio viene limitato il fan-out dei nodi OR ad una costante k, in quanto sulla base degli esperimenti fatti si può affermare che, sia nel caso di square che nel caso di pattern opzionali, la soluzione si trova sempre molto in prossimità del punto dove si è riscontrato il mismatch. Vengono quindi ordinati i pattern potenziali sulla base della loro lunghezza, e vengono tenuti solamente i k pattern più corti.

Per limitare la quantità di memoria richiesta per memorizzare l'albero,

vengono scartate alcune porzioni dell'albero per le quali è già stata trovata una visita. Questi sotto-alberi corrispondono a mismatch esterni che portano a riconoscere un iteratore; fondamentalmente, dal momento che è molto improbabile che mismatch esterni portino a riconoscere iteratori falsi, ognuna di queste visite trovate viene considerata come un punto fisso dal quale non si renderà necessario tornare indietro.

Infine, vengono imposte alcune limitazioni sulla posizione dei pattern opzionali nel wrapper; come conseguenza, vengono scartati gli alberi sulla base delle rispettive posizioni degli iteratori e dei pattern opzionali nel wrapper: si scartano tutte le strategie di visita corrispondenti a wrapper nei quali un pattern (iteratore o opzionale) è delimitato da entrambi i lati da un pattern opzionale.

Queste scelte implementative riducono leggermente l'espressività del formalismo adottato, ma presentano il grande vantaggio di evitare la generazione di ricerche esponenziali.

4.2.2 Esempi

Vengono ora presentati due esempi di generazione del wrapper e del suo utilizzo per l'estrazione dei dati da pagine Web utilizzando RoadRunner.

Una prima sorgente nel dominio in esame su cui si è testato RoadRunner è rappresentata dal database Komugi disponibile all'indirizzo `http://www.shigen.nig.ac.jp/wheat/komugi/genes/symbolClassList.jsp` .

Questo database riporta dati su geni e marker del grano su pagine che presentano tra loro una struttura comune. Gli attributi riportati sono infatti

Figura 4.1: Una pagina del database Komugi

comuni tra geni e marker. Questo database si presta quindi molto bene all'utilizzo di RoadRunner.

In figura 4.1 viene riportata una pagina del database Komugi in cui vengono mostrate in una tabella alcune informazioni riguardanti geni del grano.

Per la generazione del wrapper sono state scelte come sample cinque pagine simili a quella mostrata in figura 4.1. Il wrapper che è stato inferito da RoadRunner viene riportato di seguito:

```
<?xml version='1.0' encoding="Cp1252"?>
<wrapper name="komugi" labels="_A_,_B_,_C_,_D_,_E_,_F_,_G_">
<tagname>table</tagname><keyword>Level1</keyword>
```

```
</locator>
<preferences>
<attributeValues value="id,invariant,dompath"/>
<skipAttributes value="selected,alt,bgcolor,class"/>
<evaluationSupport value="false"/>
<variantTags value="a,font,b,i,span,sup"/>
<skipTrees value="noscript,script,style"/>
<freetextTags value="a,font,b,i,span,sup,sub"/>
<keepTags value="false"/>
<separators value="-|/\()[]{}"/>
<allWhiteSpaces value="false"/>
<suppThreshold value="0.25"/>
<freetext value="true"/>
<segmentation value="true"/>
<skipTags value=""/>
<structuralSeg value="true"/>
<distillVariantTags value="true"/>
<disambiguate value="true"/>
<sizeThreshold value="2"/>
<textMarks value="false"/>
<useRoles value="true"/>
</preferences>
<preprocess>
<marks>
</marks>
```

```
<segments>
<segment id="4">
<mark id="59" type="S" ddepth="2"><pcdata depth="3"><![CDATA[Level1]]>
</pcdata></mark>
</segment>
</segments>
</preprocess>
<expression>
<and>
<tag element="table" depth="0" attrs=""/>
<tag element="segment" depth="1" attrs="invariant:RR4"/>
<tag element="tr" depth="2" attrs="align"/>
<tag element="td" depth="3" attrs="width,rowspan"/>
<pcdata depth="4"><![CDATA[Gene symbol]]></pcdata>
<tag element="/td" depth="3" attrs="width,rowspan"/>
<tag element="td" depth="3" attrs="colspan"/>
<pcdata depth="4"><![CDATA[Class]]></pcdata>
<tag element="/td" depth="3" attrs="colspan"/>
<tag element="td" depth="3" attrs="rowspan"/>
<pcdata depth="4"><![CDATA[Synonyms]]></pcdata>
<tag element="/td" depth="3" attrs="rowspan"/>
<tag element="td" depth="3" attrs="rowspan"/>
<pcdata depth="4"><![CDATA[Chromosomes]]></pcdata>
<tag element="/td" depth="3" attrs="rowspan"/>
<tag element="td" depth="3" attrs="rowspan"/>
```

```
<pcdata depth="4"><![CDATA[Germplasms]]></pcdata>
<tag element="/td" depth="3" attrs="rowspan"/>
<tag element="/tr" depth="2" attrs="align"/>
<tag element="/segment" depth="1" attrs="invariant:RR4"/>
<tag element="tr" depth="1" attrs="align"/>
<tag element="td" depth="2" attrs=""/>
<pcdata depth="3"><![CDATA[Level1]]></pcdata>
<tag element="/td" depth="2" attrs=""/>
<tag element="td" depth="2" attrs=""/>
<pcdata depth="3"><![CDATA[Level2]]></pcdata>
<tag element="/td" depth="2" attrs=""/>
<tag element="td" depth="2" attrs=""/>
<pcdata depth="3"><![CDATA[Level3]]></pcdata>
<tag element="/td" depth="2" attrs=""/>
<tag element="td" depth="2" attrs=""/>
<pcdata depth="3"><![CDATA[Level4]]></pcdata>
<tag element="/td" depth="2" attrs=""/>
<tag element="/tr" depth="1" attrs="align"/>
<tag element="segment" depth="1" attrs="invariant:RR4"/>
<plus>
<and>
<tag element="tr" depth="2" attrs="valign"/>
<tag element="td" depth="3" attrs="align"/>
<variant label="_A_"><pcdata depth="4">
<![CDATA[hd b1 b2]]>
```

```
</pcdata>
</variant>
<tag element="/td" depth="3" attrs="align"/>
<tag element="td" depth="3" attrs=""/>
<variant label="_B_"><pcdata depth="4"><![CDATA[Awnedness]]>
</pcdata>
</variant>
<tag element="/td" depth="3" attrs=""/>
<tag element="td" depth="3" attrs=""/>
<hook>
<and>
<variant label="_C_"><pcdata depth="4"><![CDATA[Smooth awns]]>
</pcdata>
</variant>
</and>
</hook>
<tag element="/td" depth="3" attrs=""/>
<tag element="td" depth="3" attrs=""/>
<hook>
<and>
<variant label="_D_"><pcdata depth="4"><![CDATA[Hooded]]>
</pcdata>
</variant>
</and>
</hook>
```

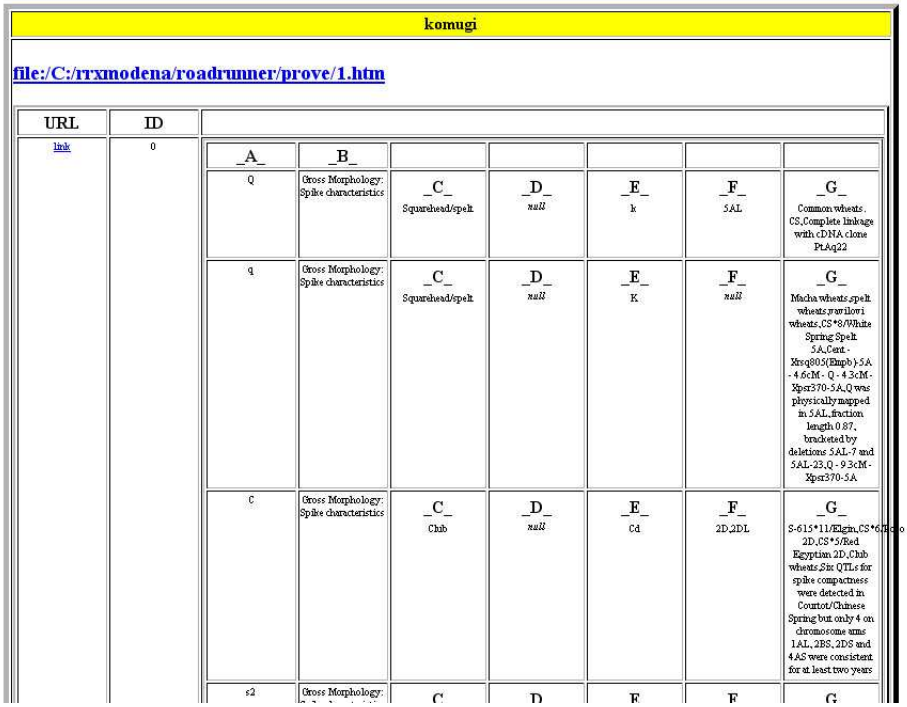
```
<tag element="/td" depth="3" attrs=""/>
<tag element="td" depth="3" attrs=""/>
<tag element="/td" depth="3" attrs=""/>
<tag element="td" depth="3" attrs=""/>
<hook>
<and>
<variant label="_E_"><pcdata depth="4"><![CDATA[Cd]]>
</pcdata>
</variant>
</and>
</hook>
<tag element="/td" depth="3" attrs=""/>
<tag element="td" depth="3" attrs=""/>
<hook>
<and>
<variant label="_F_"><pcdata depth="4"><![CDATA[2D,2DL]]>
</pcdata>
</variant>
</and>
</hook>
<tag element="/td" depth="3" attrs=""/>
<tag element="td" depth="3" attrs=""/>
<hook>
<and>
<variant label="_G_"><pcdata depth="4">
```

```
<![CDATA[S-615*11/Elgin,CS*6/Poso 2D,CS*5/Red Egyptian 2D,Club
wheats,Six QTLs for spike compactness were detected in
Courtot/Chinese Spring but only 4 on chromosome arms
1AL, 2BS, 2DS and 4AS were consistent for at least two years]]>
</pcdata>
</variant>
</and>
</hook>
<tag element="/td" depth="3" attrs=""/>
<tag element="/tr" depth="2" attrs="valign"/>
</and>
</plus>
<tag element="/segment" depth="1" attrs="invariant:RR4"/>
<tag element="/table" depth="0" attrs=""/>
</and>
</expression>
</wrapper>
```

Come si può notare il wrapper è un file XML che equivale ad una espressione che rappresenta la pagina da cui estrarre le informazioni.

Questo wrapper è stato successivamente utilizzato su un insieme di 20 pagine per estrarne il dataset, ottenendo il risultato mostrato in figura 4.2. Il processo di estrazione del dataset da un insieme di pagine HTML effettuato da RoadRunner genera in output un file Xml, che nella figura 4.2 viene presentato in forma tabellare ottenuta dal file di output. Questo file Xml riporta un albero Xml, nei cui nodi vengono memorizzati i dati estratti.

Extracted DataSet



komugi

file:/C:/rrxmodena/roadrunner/prove/1.htm

URL	ID	_A_	_B_					
link	0	Q	Gross Morphology: Spike characteristics	_C_ Squarehead/spelt	_D_ *null*	_E_ k	_F_ 5AL	_G_ Common wheats CS,Complete linkage with cDNA clone PtAq22
		q	Gross Morphology: Spike characteristics	_C_ Squarehead/spelt	_D_ *null*	_E_ K	_F_ *null*	_G_ Macha wheats,spelt wheats,vavilowi wheats,CS*8/White Spring Spelt 5A,Cont - Xbcd805(Bingh)-5A - 4.6cM - Q - 4.3cM - Xpsr370-5A,Q was physically mapped in 5AL,fraction length 0.87, bracketed by deletions 5AL-7 and 5AL-23,Q - 9.3cM - Xpsr370-5A
		C	Gross Morphology: Spike characteristics	_C_ Chib	_D_ *null*	_E_ Cd	_F_ 2D,2DL	_G_ S-615*11/Elgin,CS*6/Elgin 2D,CS*5/Red Egyptian 2D,Chib wheats,Six QTLs for spike compactness were detected in Courtot/Chinese Spring but only 4 on chromosome arms 1AL, 2BS, 2DS and 4AS were consistent for at least two years
		s2	Gross Morphology:	C	D	E	F	G

Figura 4.2: Dataset estratto dal Database Komugi

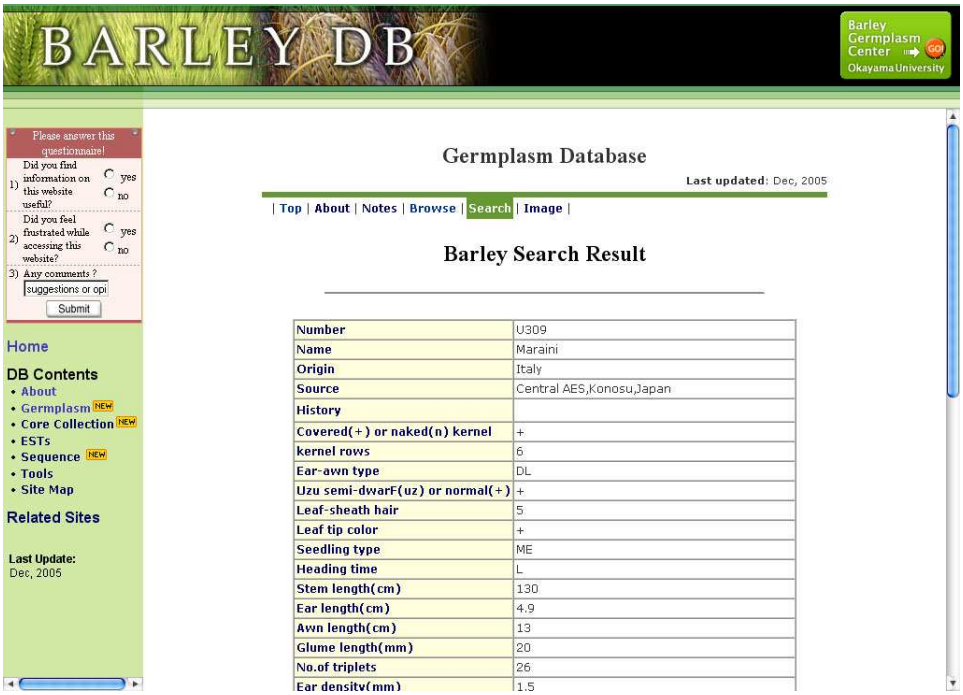
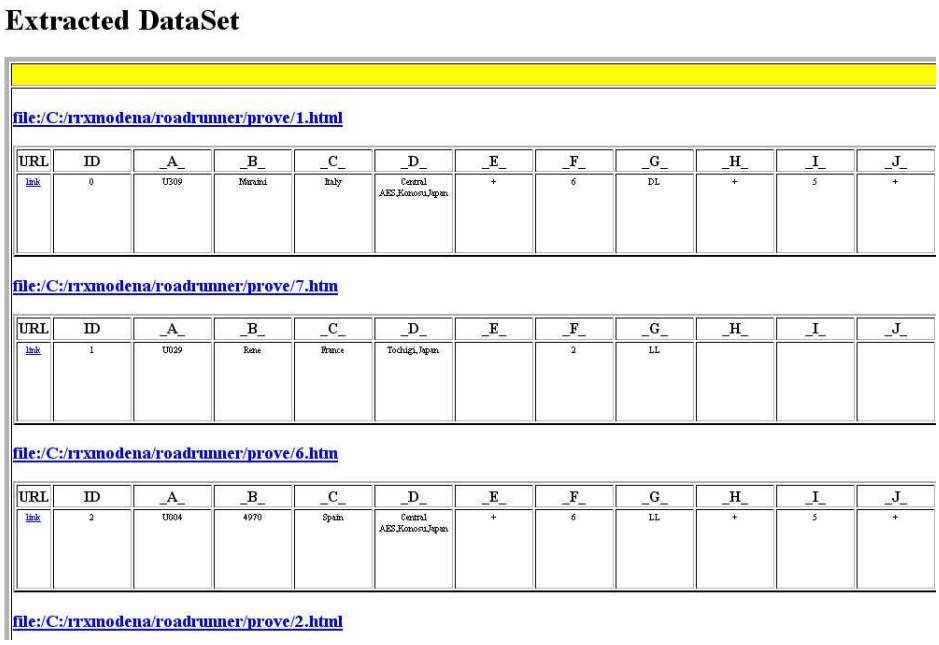


Figura 4.3: Pagina di Barley Db

Un altro database sul quale è stato provato RoadRunner è il Barley DB, consultabile all'indirizzo `http://www.shigen.nig.ac.jp/barley/index.html` nel quale è possibile trovare preziose informazioni riguardanti il germplasm dell'orzo.

Anche in questo caso la rappresentazione dei dati avviene in forma tabellare con attributi e struttura delle diverse tabelle sempre molto simile. Un esempio viene mostrato in figura 4.3.

In questo caso, a causa della scarsa variabilità degli attributi riportati nelle tabelle di Barley DB, è stato necessario selezionare accuratamente le pagine da utilizzare per la generazione del wrapper. Nel caso infatti che vengano sottoposte a RoadRunner pagine nelle quali un attributo assume



Extracted DataSet

file:/C:/rrxmodena/roadrunner/prove/1.html

URL	ID	_A_	_B_	_C_	_D_	_E_	_F_	_G_	_H_	_I_	_J_
link	0	U309	Maraini	Italy	Central AES,Konosu,Japan	+	6	DL	+	5	+

file:/C:/rrxmodena/roadrunner/prove/7.htm

URL	ID	_A_	_B_	_C_	_D_	_E_	_F_	_G_	_H_	_I_	_J_
link	1	U029	Rene	France	Tochigi, Japan		2	LL			

file:/C:/rrxmodena/roadrunner/prove/6.htm

URL	ID	_A_	_B_	_C_	_D_	_E_	_F_	_G_	_H_	_I_	_J_
link	2	U004	4970	Spain	Central AES,Konosu,Japan	+	6	LL	+	5	+

file:/C:/rrxmodena/roadrunner/prove/2.html

Figura 4.4: Dataset estratto da Barley Db

un valore costante, in questo caso questo valore non verrebbe estratto dal wrapper, in quanto in fase di generazione del wrapper non genera alcun mismatch tra il wrapper e il sample con cui viene confrontato.

Il wrapper è stato quindi generato utilizzando 8 pagine diverse, scelte in modo da presentare variabilità nei valori degli attributi.

Il dataset ottenuto utilizzando il wrapper così generato viene mostrato in figura4.4.

Le prove effettuate dimostrano una buona affidabilità di RoadRunner per la generazione del wrapper, anche se l'utente deve comunque porre attenzione alle pagine che vengono scelte per generare il wrapper, come si è visto nel secondo esempio.

Il dataset che viene ottenuto, sotto forma di file XML, può facilmente

essere utilizzato per popolare un database relazionale. Attualmente infatti molti DBMS supportano l'importazione dei dati da file in formato XML.

In questo modo è possibile quindi effettuare l'estrazione dei dati da un set di pagine presenti sul Web, generando in modo praticamente automatico un wrapper per questo scopo.

Le potenziali applicazioni di RoadRunner sono quindi molteplici, anche se il suo utilizzo risulta ancora macchinoso in quanto la fase di ricerca delle pagine da cui estrarre i dati per il momento deve essere compiuta manualmente dall'utente. L'obbiettivo è quello di riuscire ad automatizzare anche la fase di esplorazione del sito, anche se per ora le tecniche per effettuare il crawling della sorgente Web sono ancora allo studio [6, 15].

4.3 Altre Tecniche per la Generazione Automatica di Wrapper

Sono numerosi gli approcci diversi presenti in letteratura che sfruttano tecniche differenti per affrontare il problema della generazione di wrapper per sorgenti Web HTML, in manier automatica o semiautomatica.

Per esempio W4F (World Wide Web Wrapper Factory) [16] è un toolkit per la costruzione di wrapper. Esso divide il processo di sviluppo del wrapper i 3 fasi: per prima cosa, l'utente descrive come accedere al documento, quindi quali parti di dati vuole che siano estratte, e infine descrive la struttura che deve essere usata per memorizzare i dati estratti.

Un documento viene prima recuperato dal Web secondo certe regole spe-

cificate dall'utente, e una volta recuperato viene sottoposto ad un parser HTML che sviluppa un albero per il parsing del documento che ne riporta il Document Objct Model DOM [11]. In seguito, l'utente può scrivere alcune regole per localizzare i dati all'interno del parsing tree. I dati estratti vengono memorizzati usando un formato interno di W4F, chiamato NSL (Nested String List). Infine, le struttre NSL possono essere esportate in applicazioni di livello superiore, seguendo alcune specifiche regole di mapping. Il linguaggio usato da W4F per definire le regole di estrazione è chiamato HEL (HTML Extraction Language). Una regola per l'estrazione corrisponde ad un assegnamento di un nome di variabile ad una path-expression. W4F offre un wizard per aiutare l'utente nella scrittura di queste regole per l'estrazione, che vengono applicate ai nodi dell'albero per estrarre i dati. Per un documento Web, viene fornito all'utente lo stesso documento annotato con ulteriori informazioni. L'utente, cliccando sulle informazioni che gli interessano, ottiene la corrispondente regola per estrarre quei dati. Questa procedura però non gestisce insiemi di dati, quindi se l'utente è interessato a diverse istanze dello stesso dato deve essere aggiunte delle condizioni al path expression.

XWRAP [17] è un altro tool per generare wrapper che sfrutta una libreria di componenti la quale offre blocchi di base per costruire un wrapper e un'interfaccia user-friendly per facilitare il processo di sviluppo del wrapper. Prima di compiere l'estrazione il tool provvede a ripulire la sorgente HTML da tag errati o errori sintattici e trasforma il documento in un parsing tree. Il tool funziona guidando l'utente attraverso passi successivi durante i quali vengono selezionate le componenti appropriate della sua libreria. Alla fine di questi passi, XWRAP fornisce un wrapper (in Java) per una specifica

sorgente. Per l'estrazione, il tool sfrutta un insieme predefinito di metodi euristici appositi per le pagine HTML. L'utente può provare uno dei metodi disponibili, e se soddisfatto del risultato che ottiene, il processo di estrazione prosegue. L'utente può anche raffinare l'estrazione restingendo o rilassando il numero di componenti per oggetto o specificando il tipo di dato degli elementi. Quando il risultato dell'estrazione è soddisfacente, l'utente può inserire un nome per il tag di ogni elemento da estrarre e procedere alla generazione del codice del wrapper.

La differenza sostanziale di questi tool appena presentati rispetto a RoadRunner consiste nell'automazione del processo di generazione del wrapper. Utilizzando questi tool infatti il processo di generazione del wrapper è semiautomatico: l'utente deve intervenire fortemente nel processo suggerendo lo schema dei dati o fornendo esempi di dati da estrarre. RoadRunner invece, come detto in precedenza, è completamente automatico. In particolare, se le pagine per cui generare il wrapper presentano tutte la stessa struttura, viene ricostruita questa struttura a partire dal codice HTML di alcune pagine sample. In questo modo l'utente non deve intervenire in alcun modo.

4.4 Altre Tecniche per l'Estrazione dei Dati da Web

Il recupero di informazioni è un processo che a oggi si rileva strategico, soprattutto in certi campi come quello affrontato in questa tesi, ma al tempo stesso assai complesso e difficile da affrontare. L'estrazione dei dati dal Web

attraverso wrapper si rivela un approccio che può facilitare molto le cose, ma richiede pur sempre di sapere come e soprattutto dove ottenere i dati. Molto spesso però, soprattutto nel dominio di cui ci stiamo occupando, interrogare siti Web data-intensive per ottenere le informazioni che ci interessano può diventare alquanto difficile per un utente che non sia in possesso di conoscenze abbastanza approfonfite sia di natura biologica che di natura informatica. Un approccio per facilitare questa fase, ancora in fase di studio, è presentato in [4]. L'obbiettivo è quello di aiutare l'utente nella fase di interrogazione di quello che viene denominato il "deep Web", cioè tutta quella parte del Web generata dinamicamente, "on the fly", utilizzando le informazioni presenti su Database. Nel luglio 2000, infatti, è stato stimato [12] che esistessero 96,000 siti di ricerca che indirizzavano 550 miliardi di pagine di questo deep Web. Uno studio compiuto quattro anni più tardi nel 2004 stima che i database consultabili online siano 450,000 [13]. In questo deep Web, numerosi database online forniscono accesso dinamico ai loro dati basato su query fornite utilizzando i loro form, invece di un link URL statico. Per facilitare l'interazione dell'utente con questi form che devono essere utilizzati per formulare la query da sottoporre al database, viene proposto un "form assistant" per aiutare l'utente nel processo di formulazione della query sul Web. In particolare, col proliferare delle sorgenti dati disponibili per i vari domini di interesse, spesso risulta necessario interrogare sorgenti alternative nello stesso dominio. La soluzione proposta in [14] offre la possibilità di integrare sorgenti selezionate dinamicamente rilevanti per la query dell'utente nella quale però si rende necessario tradurre "on the fly" la query formulata dall'utente per queste sorgenti; oppure in [4] viene proposto lo sviluppo di un "form assistant toolkit"

per suggerire all'utente query ulteriori che potrebbero soddisfare la sua richiesta, eventualmente destinate ad altri siti che trattano lo stesso dominio. Il problema centrale rimane però la traduzione dinamica delle query tra form selezionati dinamicamente nello stesso dominio, quindi più specificatamente tradurre la query di un utente da un form sorgente ad un form target.

Un query translator dovrebbe avere due proprietà:

1. la tecnica di traduzione dovrebbe essere in grado di affrontare sorgenti nuove o mai affrontate prima;

2. deve poter essere facilmente adattato tramite conoscenze specifiche a qualunque nuovo dominio.

Per effettuare la traduzione di una query $Q_S$formulata sul form di una sorgente S in un form target T, è necessario riuscire a conciliare le eterogeneità della query a tre livelli distinti:

- livello di attributo: due sorgenti potrebbero non supportare richieste poste sugli stessi concetti o potrebbero consentire di porre query sugli stessi concetti utilizzando nomi differenti.

- livello di predicato: due sorgenti potrebbero usare predicati diversi per lo stesso concetto. Per esempio, un predicato sullo stesso concetto in T e in S può consentire di specificare un set di valori differenti. In questo caso la traduzione dovrà essere il più simile possibile. Si definisce la prossimità accettabile della traduzione come "minimal subsumption", deve ovvero contenere la query originale Q_S aggiungendo il minor numero di risposte ulteriori.

- livello di query: due sorgenti possono avere diverse possibilità di effettuare query su combinazioni valide di predicati.

Per realizzare il query translator è quindi necessario conciliare le eterogeneità presenti a questi tre livelli e generare un piano per la query espresso rispetto al form target T. Questo piano consta di due parti: una union query Q_t^*, che è l'unione delle query possibili sul form target per ottenere risposte rilevanti dal database target; e un filtro σ, che è una selezione per filtrare i falsi positivi dalla union query. Per limitare il costo del post-processing, ovvero dell'operazione di filtraggio, si cerca di avere una union query Q_t^* il più prossima possibile alla query sorgente Q_S in modo da ottenere il minor numero possibile di risposte extra.

Il problema della traduzione della query può quindi essere formulato propriamente in questo modo: data una query sorgente Q_S e un form target T, tra tutte le union query "valide" di T, scegliamo quella "semanticamente più vicina" a Q_S, come miglior traduzione. Bisogna però definire propriamente il concetto di union query valida e di vicinanza semantica.

Il query model di una sorgente descrive gli esempi di query accettabili. Quindi il query model di un form consiste di un vocabolario Σ e una sintassi $\mathcal{F}$. Il vocabolario Σ difinisce un insieme di modelli di predicati utilizzabili nel form. Un modello di predicato è una tupla composta dai tre elementi *[attributo; operatore; valore]*. Su questo vocabolario di modelli di predicati, la sintassi $\mathcal{F}$ specifica tutte le combinazioni valide di questi modelli nei confronti del form. La sintassi utilizzata in [4] si focalizza sulle "conjunctive query", ritenute sufficienti per esprimere le query possibili sulla maggior

parte delle sorgenti del deep Web. Sono stati inoltre riconosciuti due tipi di vincoli nella maggior parte delle sorgenti: in primo luogo alcuni modelli di predicato possono essere formulati in modo esclusivo, ovvero le selezioni su alcuni attributi possono comparire una sola volta in una query; inoltre un form può avere alcuni vincoli di legame che richiedono che un certo modello di predicato sia riempito obbligatoriamente. Viene quindi definita $\mathcal{F}$ come tutte le conjunctive query valide che soddisfano questi vincoli.

Sulla base di quanto appena detto, si può quindi definire cosa sia una union query valida. Si definisce un predicato $\mathrm{p_i}$come un'istanza di un modello di predicato $\mathrm{P_i}$, ovvero un valore concreto dei parametri di $\mathrm{P_i}$. Un form di query $\mathrm{f_i}$ è un istanza di un conjunctive form $\mathrm{F_i}$. Una union query valida su un form target è dunque $\mathrm{f_1} \vee \ldots \vee \mathrm{f_n}$, dove $\mathrm{f_i}$ è un form di query.

Vengono considerate esclusivamente le traduzioni delle unioni, in quanto l'operazione di unione non è bloccante, e consente quindi di costruire il risultato in modo incrementale a differenza dell'intersezione che risulta bloccante.

Per capire invece la vicinanza semantica tra una query valida e la query sorgente è necessario prima definire una metrica di vicinanza. In particolare, in questo caso viene utilizzata una metrica di inclusione (minimal subsuming) $\mathrm{C_{min}}$, ovvero data una query sorgente $\mathrm{Q_S}$ e un form target T, una query Q_t^* è una traduzione minimal subsuming rispetto a T se:

1. Q_t^* è una query valida rispetto a T;
2. Q_t^* include $\mathrm{Q_S}$, cioè per ogni istanza del database $\mathrm{D_i}$, $Q_s\left(D_i\right) \subseteq Q_t^*\left(D_i\right)$;

3. Q_t^* è minima, ovvero non esiste nessuna query $\mathrm{Q_t}$ tale che $\mathrm{Q_t}$ soddisfa (1) e (2) e Q_t^* include $\mathrm{Q_t}$.

$\mathrm{C_{min}}$ presenta numerosi vantaggi, in quanto non tralascia alcuna risposta corretta e contiene la minor quantità di risposte sbagliate. Di conseguenza, l'overhead dovuto al filtraggio dei falsi positivi risulta minimo. Inoltre risulta indipendente dal contenuto del database.

La realizzazione della query translation risulta quindi, concettualmente, nella ricerca tra tutte le query valide di quella minimal subsuming. Questo approccio è però sicuramente poco efficiente. Per evitare una ricerca così vasta, si cerca quindi di costruire fin dall'inizio la traduzione migliore. In particolare, dal momento che la traduzione consiste essenzialmente nel conciliare le eterogeneità che si presentano ai tre livelli precedentemente detti, risulta quindi più facile cercare di risolverli separatamente, e successivamento realizzare la traduzione unendoli. Pertanto per prima cosa vengono cercati i predicati corrispondenti tra loro, quindi viene mappata ciascuna coppia di questi predicati utilizzando la metrica $\mathrm{C_{min}}$, la quale dà luogo ad una "mapped query", come viene definita, e infine si cerca la riscrittura $\mathrm{C_{min}}$ della mapped query nel form target.

Per garantire la correttezza di questa separazione, viene richiesto che la mapped query, generata dai predicati di mapping individualmente, sia riscrivibile come traduzione minimal subsuming.

Questa separazione favorisce quindi una realizzazione modulare del form assistant, che consiste quindi di tre componenti: l'attribute matcher, il predicate mapper e il query rewriter.

L'attribute matcher si occupa di trovare le corrispondenze semantiche tra gli attributi. Per realizzare quello che viene solitamente chiamato schema matching, è possibile utilizzare thesaurus automatici o codificati manulamente, eventualmente con conoscenze specifiche per il dominio in esame.

Quindi, per ogni coppia di predicati corrispondenti, il predicate mapper associa il predicato sorgente al target utilizzando la metrica C_{min}. Dopo questa fase, accoppiando gli attributi e ogni predicato individualmente, si ottiene una mapped query. Infine, il query rewriter riscrive la mapped query in una query valida per il form target, ovvero una query che ne rispetti i vincoli sintattici.

Per il problema del predicate mapping viene presentata una soluzione che prende come input un predicato sorgente s accoppiato ad un modello di predicato target P, e restituisce come output la traduzione più vicina al target t per s. In particolare, il predicate mapper consiste di due componenti: un type recognizer e un type handler. Il primo riconosce il tipo di dato del predicato, in quanto i mapping hanno senso solo all'interno di dati dello stesso tipo, e lo distribuisce al rispettivo type handler. Quest'ultimo, realizzato *ad-hoc* per ogni tipo di dato, effettua il mapping tra i predicati di uno specifico tipo di dato con un approccio search driven. Invece di scrivere regole specifiche per realizzare il mapping per ogni coppia di modelli di predicato, viene codificata la conoscenza che si ha per effettuare il mapping in un metodo di valutazione di ogni modello. Questa valutazione consente di "materializzare" la semantica di una query rispetto ad uno specifico tipo di dato. I confronti semantici posssono quindi essere effettuati solo su questi risultati che si sono materializzati. Trovare il mapping più vicino si riduce

quindi ad un problema di ricerca, estensibile esplorando queste metriche di valutazione invece di regole statiche di confronto.

La tecnica presentata al momento è ancora in fase di studio e non sono ancora pronti tool per poterla sperimentare. Per il dominio di interesse di questa tesi potrebbe però rivelarsi un valido aiuto per interrogare i numerosi database presenti sul Web. Una volta selezionati alcuni di questi database infatti, potrebbe automatizzare la formulazione della stessa query su più sorgenti semplificando quindi il processo di interrogazione. Molto spesso infatti database contenenti lo stesso tipo di informazioni presentano form per effettuare le query molto diversi tra loro, rendendo quindi difficile l'interrogazione di più sorgenti per un utente che non abbia abbia discrete conoscenze informatiche, come ad esempio nel caso dell'utenza media dei database genetici.

Capitolo 5

Conclusioni e Sviluppi Futuri

In questo lavoro di tesi è stata presentata la progettazione del database CereaLab, destinato alla selezione varietale di piante di cereali. Per la realizzazione di questo database è stato utilizzato il sistema a mediatore Momis, per l'integrazione di sorgenti dati già esistenti.

Momis si è rivelato un valido strumento per la realizzazione del database che era stato progettato, in quanto ha consentito di integrare dati già presenti i sorgenti strutturate ottenute dal Web. Il processo di integrazione, gestito tramite il sistema Momis, è stato realizzato in modo quasi completamente automatico, impegnando poco il progettista se non nella scelta delle forme base e dei significati dei termini che identificavano le classi e gli attributi delle sorgenti. Risulta quindi chiaro l'aiuto che può essere offerto da uno strumento come MOMIS per gestire dati eterogenei in un dominio come quello in esame. L'unico problema incontrato è dovuto alla difficoltà nel ricostruire le relazioni tra le diverse classi locali delle sorgenti, ma tale problema è dovuto nel nostro caso all'elevato numero di classi locali, che richiede una conoscenza molto

approfondita delle sorgenti per poter accedere la grande mole di informazioni a disposizione.

Per ovviare a questo problema e per completare la realizzazione del database CereaLab si propone quindi di creare una terza sorgente dati, traducendo in un database relazionale il database progettato al capitolo 2, in modo da poter integrare le informazioni ottenute dalle sorgenti Gramene e Graingenes con i dati che verranno mano a mano ottenuti sperimentalmente dal Laboratorio CereaLab. In questo modo risulterà molto più facile interrogare il database sfruttando le relazioni presenti nell'ontologia progettata all'inizio di questo lavoro di tesi.

Inoltre la realizzazione del database CereaLab non può fermarsi alla realizzazione dello schema globale, ma richiede la realizzazione di una interfaccia Web di più alto livello rispetto al Query Manager. Per utilizzare il Query Manager è infatti necessario conoscere il linguaggio Sql, mentre per mettere a disposizione dei coltivatori le informazioni necessarie alla selezione varietale delle piante di cereali contenute nel database sarà necessario realizzare un'interfaccia Web user-friendly. A questo scopo sarebbe utile se Momis supportasse la creazione e l'utilizzo di viste sulle sorgenti locali e sulla Global Virtual View, in modo da semplificare le interrogazioni del database.

Sono state inoltre presentate nuove tecniche per estrarre dati dal Web per ovviare al problema iniziale del reperimento delle informazioni per popolare il database. Le tecniche presentate, in particolare lo strumento RoadRunner, sono risultate molto valide per questo fine. L'unico problema allo stato attuale dell'arte per poter sfruttare a pieno queste tecniche risiede nella possibilità di automatizzare il reperimento delle pagine sorgente da cui estrarre i

dati attraverso i wrapper generati da RoadRunner. Al momento infatti questa operazione deve essere effettuata manualmente, richiedendo un intenso lavoro di reperimento delle informazioni. Sono però allo studio tecniche per automatizzare l'esplorazione, o crawling, delle sorgenti sul web in modo da poter ottenere automaticamente le pagine sorgente.

Bibliografia

[1] S. Bergamaschi, S. Castano, D. Beneventano, M. Vincini: "Semantic Integration of Heterogeneous Information Sources", Special Issue on Intelligent Information Integration, Data & Knowledge Engineering, Vol.36, Num. 1, Pages 215-249, Elsevier Science B.V. 2001.

[2] Salvatore Curia: Studio e integrazione di sorgenti, per la realizzazione del database CEREALAB.

[3] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo: RoadRunner: Towards Automatic Data Extraction from Large Web Sites. VLDB 2001: 109-118.

[4] Zhen Zhang, Bin He, Kevin Chen-Chuan Chang: Light-weight Domain-based Form Assistant: Querying Web Databases On the Fly. VLDB 2005: 97-108.

[5] Bin He, Zhen Zhang, Kevin Chen-Chuan Chang: Towards Building a MetaQuerier: Extracting and Matching Web Query Interfaces. ICDE 2005: 1098-1099

[6] Luigi Arlotta, Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo: Automatic annotation of data extracted from large Web sites. WebDB 2003: 7-12.

[7] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, Paolo Missier: An Automatic Data Grabber for Large Web Sites. VLDB 2004: 1321-1324.

[8] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran Soares da Silva, Juliana S. Teixeira: A Brief Survey of Web Data Extraction Tools. SIGMOD Record 31(2): 84-93 (2002).

[9] Stéphane Grumbach, Giansalvatore Mecca: In Search of the Lost Schema. ICDT 1999: 314-331.

[10] Sriram Raghavan, Hector Garcia-Molina: Crawling the Hidden Web. VLDB 2001: 129-138.

[11] World Wide Web Consortium. W3C. The Docment Object Model. http://www.w3.org/DOM.

[12] BrightPlanet.com. The deep Web: Surfacing hidden value. Accessible at http://brightplanet.com

[13] Kevin Chen-Chuan Chang, Bin He, Chengkai Li, Mitesh Patel, Zhen Zhang: Structured Databases on the Web: Observations and Implications. SIGMOD Record 33(3): 61-70 (2004).

[14] Kevin Chen-Chuan Chang, Bin He, Zhen Zhang: Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. CIDR 2005: 44-55.

[15] Luigi Arlotta, Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo: Definition of Techniques for the Automatic Inferenence of the Schema of a Data-intensive Web Site.

[16] Arnaud Sahuguet, Fabien Azavant: Building intelligent Web applications using lightweight wrappers. Data Knowl. Eng. 36(3): 283-316 (2001).

[17] Ling Liu, Calton Pu, Wei Han: XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. ICDE 2000: 611-621.