UNIVERSITA' DEGLI STUDI DI MODENA
E REGGIO EMILIA

Facoltà di Ingegneria
Sede di Modena

Corso di Laurea Specialistica in Ingegneria Informatica

INTEGRAZIONE MULTILINGUA DI SORGENTI DATI POSTGRESQL DELLA PUBBLICA AMMINISTRAZIONE: ACCOPPIAMENTO DEI SISTEMI MOMIS E SIAM

Relatore:
Chiar.mo Prof. Sonia Bergamaschi

Correlatori:
PhD. Ing. Mirko Orsini
PhD. Ing. Laura Po

Candidato:
Riccardo Saponi

Anno Accademico 2008 - 2009

A Lello, Rosaria e Rossella

PAROLE CHIAVE:
MOMIS
MultiWordNet
SIAM
PostgreSQL
Intelligent Information Integration

INDICE

INDICE DELLE FIGURE

INDICE DELLE TABELLE

Introduzione

L’informatizzazione della Pubblica Amministrazione è certamente una delle principali sfide che Stato, Regioni ed Enti Locali si trovano ad affrontare in questo periodo storico. L’impatto della tecnologia sull’amministrazione pubblica ed i servizi ai cittadini è di enorme portata, ma per risultare veramente efficace il processo di informatizzazione necessita di un gran numero di strumenti normativi, tecnici e organizzativi. Uno dei principali aspetti critici da considerare nelle attuali procedure di informatizzazione dei servizi pubblici è proprio l’aspetto tecnico. Si è verificato negli ultimi anni un aumento esponenziale del numero e della tipologia di fonti di informazione. L’eterogeneità del patrimonio informativo si manifesta sia nella natura degli stessi dati (testi, dati strutturati, immagini, record..), sia nei diversi tipi di sorgente nelle quelle tali dati sono contenuti (DBMS, pagine HTML, file system..). Cosi, se da un lato si ha a disposizione un numero sempre maggiore di informazioni con la naturale conseguenza di poter eventualmente aumentare i servizi erogati, dall’altro non risulta essere affatto semplice reperire e strutturare queste informazioni. Attualmente gli standard di maggior utilizzo nei Sistemi Informativi riescono solo in parte a risolvere o addirittura non trattano assolutamente i problemi relativi alla modellazione delle informazioni, fra cui quelli relativi alla *semantica*, ovvero alla *qualità dell’informazione* e quelli relativi all' *information overloading*, ovvero alla *quantità effettiva dell’informazione utile*. Ciò costituisce un evidente impedimento del processo di informatizzazione della Pubblica Amministrazione ed è in questo contesto che si concretizzano gli obiettivi di questa Tesi.

Attraverso l’utilizzo di **MOMIS** (**M**ediator envir**O**nment for **M**ultiple **I**nformation **S**ources), un *data integration system*, si analizzeranno diverse problematiche che riguardano il reperimento e l’interrogazione di dati da parte del **SIAM** (**S**istema **I**nformativo **AM**bientale), una Web Application in uso in diverse province di Italia, sviluppata dall’azienda **Quix s.r.l**. Tale Web Application si occupa di gestire l’iter procedurale delle Autorizzazioni dell’Ufficio Ambiente delle Province e ha come caratteristica fondamentale quella di poter interagire con diversi modelli di informazioni e l’utilizzo di tecnologie open – source.

Nell’ipotesi di poter avere una “vista” sui dati di diverse province e aumentare le potenzialità della Web Application è nata l’idea di un accoppiamento fra questi due programmi. L’analisi per la realizzazione di tale accoppiamento ha messo in luce una serie di fattori ai quali è stato necessario fornire una soluzione per poter proseguire con lo studio.

L’installazione del SIAM presa in considerazione è quella relativa alla **Provincia di Ancona**. Si sono utilizzati i dati di produzione presenti su un database **PostgreSQL**, un RDBMS open – source.

Non essendo presente in MOMIS una "risorsa" adatta per l'interfacciamento con i dati di questo tipo di database si è proposto uno studio per la realizzazione di un Wrapper che potesse provvedere a tale mancanza. All'interno di MOMIS un Wrapper è un modulo software che si occupa di descrivere in un linguaggio comune le sorgenti alle quali MOMIS è connesso e di permettere l'esecuzione di query locali e di presentare i risultati al mediatore.

Altro aspetto critico ha riguardato il database lessicale attualmente presente in MOMIS. Come anticipato MOMIS è un *data integration system* che offre ottimi risultati durante l'integrazione di database o fonti di informazione implementati in lingua inglese grazie all'utilizzo del database lessicale **Princeton WordNet**. Nonostante in precedenza fosse già stato effettuato uno studio su alcuni database lessicali multilingua non era stata ancora implementata una soluzione che ne garantisse l'utilizzo all'interno di MOMIS. Il SIAM si basa però su fonti in lingua italiana, sia per scelte in fase di sviluppo, sia per consentire un uso più agevole ai diversi operatori delle province o alle aziende che si interfacciano con la parte pubblica del programma. E' in questo contesto che si è collocato lo studio delle risorse multilingua **MultiWordNet** ed **EuroWordNet** e la creazione di un database lessicale multilingua.

In stretta relazione con il punto precedente si collocano le modifiche apportate a MOMIS riguardo alcuni metodi per l'annotazione automatica o manuale delle sorgenti locali, step fondamentale per il processo di integrazione. Uno dei vantaggi nell'utilizzo di MOMIS è infatti la possibilità di avere a disposizione alcune funzionalità automatiche o semi – automatiche che riguardano la costruzione di una Vista Globale mediante il database lessicale. Essendo funzionalità strettamente legate al database lessicale, e di conseguenza alle lingue implementate, si è resa necessaria apportare delle variazioni mediante una logica basata sulla possibilità di scelta della lingua con cui annotare una sorgente. Verrà inoltre proposto un primo studio per una funzione di *stemming* in lingua italiana.

Infine vi è una profonda differenza fra i programmi SIAM e MOMIS, ovviamente dovuta alle diverse *mission* che si prefiggono. Il SIAM è una Web Application che interagisce con delle fonti fisiche localizzate all'interno di un file system, di un DBMS o della rete e ciò costituisce anche un vincolo obbligatorio per poterne usufruire. MOMIS invece, essendo un data integration system, ha la possibilità di interagire velocemente con diverse sorgenti di dati per la costruzione di uno schema globale virtuale, ma non ha ancora la possibilità di poter materializzare questo schema in un database relazionale. Per poter proseguire e concludere l'accoppiamento fra i due sistemi è stata realizzata una funzione che consente la materializzazione totale o parziale dello schema globale generato da MOMIS in un database relazionale. Tale funzione è stata sviluppata in modo tale da

sfruttare i Wrapper presenti in MOMIS per la costruzione di eventuali tabelle e per il popolamento delle stesse.

La Tesi risulta quindi essere strutturata in questo modo:

- **Capitolo 1: il programma I^3 e MOMIS.** Si presenta il Programma I^3 e il *data integration system* MOMIS, fornendo alcune informazioni sulla nascita del progetto, sull'architettura e sul processo di integrazione;
- **Capitolo 2: Wrapper JDBC / PostgreSQL.** Viene approfondita la definizione di Wrapper all'interno di MOMIS e viene descritto il RDBMS PostgreSQL, delineando alcune caratteristiche fondamentali. Infine viene data una descrizione dell'implementazione del Wrapper per PostgreSQL;
- **Capitolo 3: Sviluppo di un database multilingua per l'utilizzo in MOMIS.** Viene introdotto il database lessicale WordNet utilizzato da MOMIS; in seguito ad un'analisi dei database lessicali MultiWordNet ed EuroWordNet si procede con la descrizione dell'integrazione di MultiWordNet all'interno di WordNet per poter disporre all'interno del processo di integrazione della lingua italiana oltre che alla lingua inglese;
- **Capitolo 4: MOMIS "DataRiver": creazione della vista materializzata dello Schema Globale.** Viene descritta la realizzazione della funzione di materializzazione dello schema globale all'interno della versione "**DataRiver**" di MOMIS;
- **Capitolo 5: Un caso di studio: SIAM (Sistema Informativo Ambientale)**. Viene descritto l'applicativo SIAM e vengono presentati una serie di test su un ambiente di prova ricavato da un'installazione reale per l'eventuale realizzazione di un accoppiamento SIAM – MOMIS;

1 Il programma I^3 e MOMIS

In questo capitolo verrà offerta una breve descrizione delle problematiche legate all'Integrazione di Informazioni e all' Integrazione Intelligente delle Informazioni e del ruolo che assumono all'interno di queste problematiche i "mediatori". Verranno analizzate e descritte le peculiarità di differenti versioni del mediatore MOMIS, **M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources, sviluppato in seguito a una collaborazione fra l'Università di Modena e Reggio Emilia e l'Università di Milano ed oggi rimodernato nella versione opensource Datariver.

1.1 L'Integrazione Intelligente delle Informazioni

Come citato da Wiederhold [1], l' Integrazione delle Informazioni, $\mathbf{I}^2$, differisce dall'integrazione di dati e dei database in quanto cerca di collegare i risultati ottenuti da esse invece che limitarsi al collegare semplicemente le sorgenti stesse [2]. Il desiderio, attualmente diventato una necessità, di poter raccogliere questo tipo di informazioni e disporre quindi di una selezione "*intelligente*" dei dati prelevati da diverse sorgenti pone dei punti cardine sugli sviluppi e sulla ricerca. A questo proposito nasce l'esigenza di creare delle metodologie che siano flessibili e modulari, per un eventuale riutilizzo in base all'evoluzione delle tecnologie.

1.2 Il programma I^3

L' **Advanced Research Projects Agency**, **ARPA**, agenzia che fa a capo al **Dipartimento di Difesa degli Stati Uniti d' America** ha condotto un'importante ricerca basata sull' individuazione di un architettura di riferimento che possa consentire di realizzare l'integrazione di sorgenti eterogenee in maniera automatica o semi-automatica: il programma $\mathbf{I}^3$**,** Integrazione Intelligente delle Informazioni. Per quanto non sia necessario addentrarsi dettagliatamente nella descrizione del programma I^3, è doveroso dare delle semplici indicazioni per consentire una maggiore comprensione delle problematiche affrontate da questa tesi.

Partendo dal fatto che realizzare un sistema in grado di utilizzare diverse sorgenti, semanticamente differenti, risulta essere particolarmente oneroso e che il sistema stesso presenterebbe ovvi problemi di manutenibilità o adattabilità, il programma I^3 pone come principale

punto di riferimento la definizione di un'architettura modulare ampiamente sviluppabile secondo uno standard che ponga le basi dei servizi necessari all'integrazione. Un notevole contributo alla definizione di questo tipo di architettura è dato dall'Intelligenza Artificiale, la quale, essendo in grado di dedurre dagli schemi informazioni utili, può essere considerata un valido strumento per fornire soluzioni flessibili e riusabili.

All'interno del progetto I^3 , la suddivisione dei servizi e delle risorse di cui il sistema viene dotato è basata su due partizionamenti fondamentali:

- *orizzontale:* composto da 3 livelli: livello utente, moduli interni intermedi basati su tecnologie di IA, risorse di dati;
- *verticale:* che distingue i domini in cui raggruppare le sorgenti;

In particolar modo I^3 si concentra sul livello intermedio del partizionamento orizzontale proponendo un ulteriore suddivisone dei moduli in:

- **Mediator**: ricercano le fonti e combinano o dati da esse ottenuti;
- **Query Processor**: riformula le query per ottimizzarle;
- **Data Miner**: analizza i dati per informazioni implicite;

1.2.1 Architettura

L'architettura di un sistema I^3, definita dall'ARPA [3], deriva la dal tentativo di porre soluzione a una serie di problemi complessi:

- *Eterogeneità delle sorgenti*: differenza fra strutture dei dati, interfacce d'accesso..
- *Evoluzione delle sorgenti di dati*: esiste la possibilità che vengano create nuove tipologie di fonti e che ne vengano soppresse altre.
- *Dimensione delle fonti*: porre soluzione all'aumento della quantità di informazione all'interno di una sorgente e all'aumento del tempo di risposta delle sorgenti stesse.
- *Semantica nascosta*: interpretazione dei dati con cui interagire mediante deduzioni su regole e schemi.
- *Necessità di sistemi modulari*: punto di fondamentale importanza per la riduzione di tempi e costi di sviluppo delle singole applicazioni.

In Figura 1 è riportato uno schema dell'architettura di un sistema I^3. Si evince immediatamente come sia organizzato in diversi "moduli", ognuno adibito allo svolgimento di un particolare *servizio*:

- *Servizi di Amministrazione;*
- *Servizi di Coordinamento;*
- *Servizi di Integrazione e Trasformazione Semantica;*
- *Servizi di Wrapping;*
- *Servizi Ausiliari;*

L'architettura di un sistema I^3 fornisce particolare rilevanza ai **Servizi di Coordinamento,** i quali hanno come obiettivo quello di coordinare le operazioni attuate dai vari servizi sia in fase di progettazione che in fase di esecuzione su specifiche richieste degli utenti.

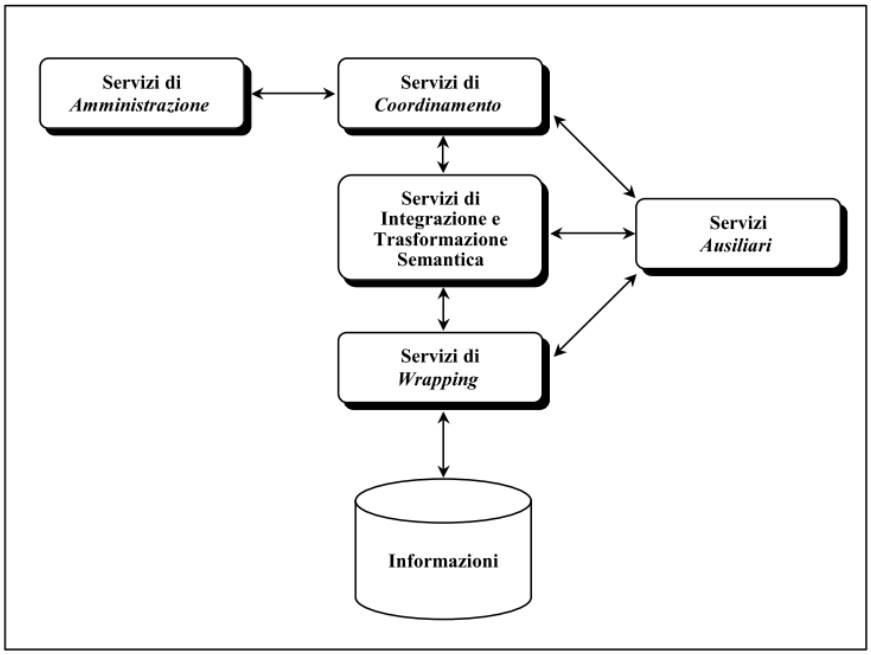

Figura 1 Struttura di un architettura I^3

Di seguito vengono analizzate in dettaglio le diverse tipologie di servizio.

Servizi di Coordinamento

Sono servizi di alto livello che svolgono operazioni di supporto e che permettono l'individuazione delle diverse sorgenti di dati che possono essere utili per risolvere le richieste dell'utente, presentando all'utente stesso l'intero sistema come un blocco unico ed omogeneo piuttosto che diviso in diversi moduli.

Proprio in base alla tipologia delle richieste dell'utente sono suddivisi in :

- **Facilitation** o **Brokering Service**: l'utente invia una determinata richiesta al sistema e questi si occupa di individuare un insieme di moduli che possono trattare la richiesta direttamente. Nel caso il modulo necessario sia solo uno si parla di ***brokering***, mentre in caso contrario di ***facilitatori*** o ***mediatori*** dove la richiesta viene inviata singolarmente ai moduli che gestiscono le diverse sorgenti per poi ripresentare i dati come provenienti da una sola sorgente. Al utente non viene richiesta la conoscenza delle diverse sorgenti e risulta evidente la diminuzione di complessità del sistema stesso.
- **Matchmaking**: l'utente configura il sistema "*a mano*" in fase di inizializzazione. In seguito ogni richiesta verrà trattata secondo tale configurazione.

Servizi di Amministrazione

Servizi utilizzati dai Servizi di Coordinamento per localizzare le sorgenti utili, determinare le loro capacità, creare i *Template*, ovvero strutture dati atte a descrivere i servizi , le fonti e i moduli da utilizzare per realizzare un particolare task. In tale maniera vengono definite a priori le azioni da eseguire a fronte di una richiesta. In alternativa ai Template possono essere usate le *Yellow pages*, servizi di directory che mantengono le informazioni sul contenuto e sullo stato di una sorgente. Consentono al Mediatore di inviare la richiesta di informazioni alla sorgente giusta a ad una equivalente qualora non fosse disponibile.

Servizi di Integrazione e Trasformazione Semantica.

Sono quei servizi che supportano le trasformazioni semantiche necessarie per l'integrazione delle informazioni. I servizi di integrazione ricevono in input le sorgenti dati tradotte dai servizi di Wrapping e restituiscono in output una "*vista*" integrata o trasformata di queste informazioni. Sono spesso indicati anche come servizi di mediazione e sono divisi in:

- **Servizi di integrazione degli schemi**: supportano la trasformazione e l'integrazione degli schemi e delle informazioni derivate da fonti eterogenee; creano un vocabolario e le ontologie condivise dalle sorgenti; integrano gli schemi in una vista globale; mantengono il *mapping* tra vista globale e schemi locali.
- **Servizi di integrazione dei dati**: aggregano ed estraggono i dati per consentire una presentazione degli stessi significativa ed esplicita.
- **Servizi di supporto al processo di integrazione**: utilizzati ad esempio quando di fronte ad

una query complessa ci si trova a doverla scomporre in subquery da inviare a fonti differenti con la necessità di integrare successivamente i dati.

Servizi di Wrapping

Sono i servizi che fungono da traduttori dei sistemi locali ai servizi di alto livello e viceversa. Consentono ai servizi di Coordinamento e di Mediazione di poter manipolare le sorgenti locali in modo uniforme e permettono di accedere alle sorgenti estratte dal maggior numero di moduli attraverso un interfaccia che segue gli standard più diffusi: **SQL** per l'interrogazione, **CORBA** come protocollo di scambio. Il processo di realizzazione di questi servizi dovrebbe essere standardizzato in modo da poter essere riutilizzato per altre fonti.

Servizi Ausiliari

I servizi ausiliari aumentano le funzionalità degli altri servizi descritti precedentemente e svolgono azioni di monitoraggio del sistema, propagazione degli aggiornamenti, ottimizzazione, etc. Sono utilizzati prevalentemente dai moduli che agiscono direttamente sulle informazioni.

1.2.2 Mediatore

Un mediatore è un modulo software che sfrutta la conoscenza su un certo insieme di dati per creare informazioni per un'applicazione di livello superiore. Dovrebbe essere piccolo e semplice, così da poter essere amministrato da uno o al più, da pochi esperti. [1]

Un mediatore è in grado di:

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

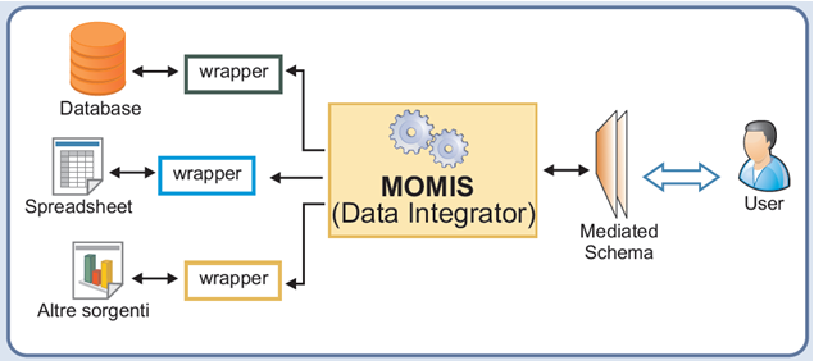

Figura 2 Il ruolo di un Sistema Informativo mediatore

Tra gli obiettivi del progetto MOMIS c'è la realizzazione di un modulo mediatore e l'architettura scelta sviluppa il sistema su 3 livelli:

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

Nei confronti dell'architettura precedentemente descritta esistono due approcci fondamentali:

- *Approccio strutturale*: caratterizzato dall'uso di *self-describing model* per rappresentare gli oggetti da integrare, limitando così l'uso delle informazioni semantiche a delle regole predefinite dall'operatore. Nella pratica, il sistema non conosce a priori la semantica di un oggetto che va a recuperare da una sorgente, bensì è l'oggetto stesso che, attraverso delle *label*, si descrive, specificando tutte le volte, per ogni suo singolo campo, il significato associato.
- *Approccio semantico*: è l'approccio effettivamente utilizzato in MOMIS, ed è caratterizzato dal fatto che il mediatore deve conoscere per ogni sorgente, lo schema concettuale; le informazioni semantiche sono codificate in questi schemi, per cui deve essere disponibile un modello comune per descrivere le informazioni da condividere, e deve esserci un'integrazione delle sorgenti di dati. In questo modo il Mediatore può individuare i concetti comuni a più sorgenti e relazioni che li legano.

A questo punto, pur avendo a disposizione gli schemi concettuali delle varie sorgenti, la risoluzione dei concetti comuni, delle relazioni che li legano non è qualcosa di immediata risoluzione in quanto, anche tralasciando le differenze fisiche dei dati(sono i wrapper che si occupano di analizzare tali peculiarità), si devono risolvere due tipologie di problemi:

- ***Problemi ontologici***: Definita una Ontologia come "*insieme dei termini e delle relazioni usate in un dominio, per indicare oggetti e concetti*",e quindi come a quell'insieme di termini in grado di individuare una determinata conoscenza in un certo dominio condiviso da tutti gli utenti, possono essere definite una gerarchia dei diversi livelli e le problematiche relative:
  - *top-level ontology*: descrizione di concetti generali(spazio,tempo), indipendenti da un particolare problema o da un dominio. E' ragionevole supporre che anche in comunità separate gli utenti condividano la stessa top-level ontology;
  - *domain and task ontology*: descrizione del vocabolario relativo ad un certo dominio o a un generico obiettivo dando una specializzazione dei termini introdotti nella top-level ontology;
  - *application ontology*: descrizione di concetti che dipendono da un particolare obiettivo o da un particolare dominio.
- ***Problemi semantici:*** sotto l'ipotesi che sorgenti diverse mantengano una visione simile del problema da modellare, e quindi un insieme di concetti comuni, è altamente improbabile che utilizzino la stessa semantica, cioè gli stessi termini, per rappresentare questi concetti[1]. Un esempio pratico sono i diversi **DBMS** che possono portare all'uso di differenti modelli per la rappresentazione dell'informazione. Compito dell'integratore sarà quindi non solo individuare l'insieme di concetti comuni ma anche risolvere le differenze semantiche presenti in un insieme di sorgenti. E' possibile classificare queste differenze in 3 macro-gruppi:
  - ***eterogeneità tra le classi di oggetti***: nonostante due classi in due differenti sorgenti rappresentino lo stesso concetto nello stesso contesto, vengono usati nomi differenti per gli stessi attributi, gli stessi metodi, oppure hanno gli stessi attributi ma domini con valori diversi o ancora regole diverse su questi valori;
  - ***eterogeneità tra le strutture delle classi***: differenze nei criteri di specializzazione e discrepanze semantiche. Un esempio è dato dall'attributo SESSO in uno schema mentre

[1] "*the probability of two person using the same term in describing the same thing is less than 20%*" Bates.

in un altro è presente implicitamente come suddivisone della classe PERSONE in MASCHI e FEMMINE;

- ***eterogeneità nelle istanze delle classi***: uso di diverse unità di misura per i domini di un attributo o presenza/assenza di valori nulli.

1.3 MOMIS

Il progetto **MOMIS**, **M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources, è un sistema intelligente di integrazione di informazioni da sorgenti di dati strutturati e semi-strutturati. MOMIS nasce all'interno del progetto **MURST 40% - INTERDATA** dalla collaborazione tra i gruppi di ricerca dell'**Università di Modena e Reggio Emilia** e dell'**Università di Milano.**

L'innovazione di questo strumento rispetto ad altri simili risiede nelle fasi di analisi ed integrazione degli schemi sorgente realizzata in modo semi-automatico, e nella fase di *query processing*, cioè nel processo che dalla query generata sullo schema globale provvede a generare in maniera automatica le subquery da inviare alle sorgenti e ad integrare i risultati.

MOMIS utilizza un approccio di integrazione delle sorgenti *semantico* e *virtuale* [4]. La parola *semantico* è riferita al fatto che l'integrazione assume come cardine del processo il significato dei termini del contesto analizzato, mentre per virtuale si intende che la vista integrata delle sorgenti non viene materializzata. In fase di interrogazione dello schema globale, il sistema si basa sulla decomposizione delle query e sull'individuazione delle sorgenti da interrogare per generare delle subquery eseguibili localmente [5];

Un sistema di Data Integration è una tripla $\langle G, S, M \rangle$, dove **G** è lo schema globale, **S** è lo schema locale o un insieme di schemi locali, e **M** rappresenta il mapping fra G ed S. In base a questa struttura possono essere individuati due possibili approcci attraverso i quali affrontare dei problemi di Integrazione Dati:

- **Global – as – View** (GaV): lo schema globale G è una "*vista*" dell' insieme delle sorgenti locali S. La query sullo schema globale vengono scomposte in sub-queries mediante la sostituzione della definizione della *vista* riportata nel mapping M. Presenta svantaggi quando vengono aggiunte nuove sorgenti locali, in quanto lo schema globale deve essere ridefinito;
- **Local – as – View** (LaV): Le sorgenti locali S sono "*viste*" dello schema globale G. Una opportuna query su G definisce una particolare sorgente e il mapping M riporta esattamente queste definizioni. L'aggiunto di una nuova sorgente locale non comporta la riscrittura dello schema globale.

In MOMIS l'approccio scelto è il **GaV** e diverse sono le motivazioni che hanno portato all'adozione di un approccio come questo:

- attraverso lo schema locale l'utente può formulare qualsiasi interrogazione che sia coerente

con esso;

- le informazioni semantiche che comprende possono contribuire all'ottimizzazione delle interrogazioni;
- la vista virtuale rende il sistema estremamente flessibile, in grado cioè di sopportare frequenti cambiamenti sia nel numero che nel tipo delle sorgenti, ed anche nei loro contenuti (non occorre quindi prevedere onerose procedure di allineamento).

La vista virtuale rende il sistema estremamente flessibile, in grado di sopportare numerosi cambiamenti, come ad esempio, l'aggiunta di una nuova sorgente o le modifiche al contenuto di alcune sorgenti. Per la rappresentazione degli schemi e per la formulazione delle interrogazioni è stato adottato un unico modello di dati basato sul paradigma ad oggetti. Viene utilizzato **ODM_I^3** [6], modello di dati ad alto livello e conveniente durante le fasi di comunicazione fra mediatore e Wrapper. Per quanto possibile si è cercato di seguire le proposte di standardizzazione per i linguaggi di mediazione in ambito I^3: un mediatore deve poter essere in grado di gestire sorgenti dotate di formalismi complessi ed altre più semplici, cercando di adottare un formalismo il più completo possibile.

Per la descrizione degli schemi si arrivati ad estendere il linguaggio **ODL**, definendo il linguaggio **ODL_I^3** [7], posposto dal gruppo di standardizzazione **ODMG-93**.

Il linguaggio di interrogazione utilizzato è l'**OQL_I^3**, che adotta la sintassi di **OQL** [8] senza discostarsi dallo standard e risulta versatile ed espressivo nello sfruttare le informazioni contenute nello schema globale.

Per la comunicazione fra moduli si è scelto lo standard **CORBA** (**C**ommon **ORB A**rchitecture) [9]. CORBA è una tecnologia di integrazione, è ad oggetti e permette di ridurre la complessità di MOMIS.

1.3.1 Architettura

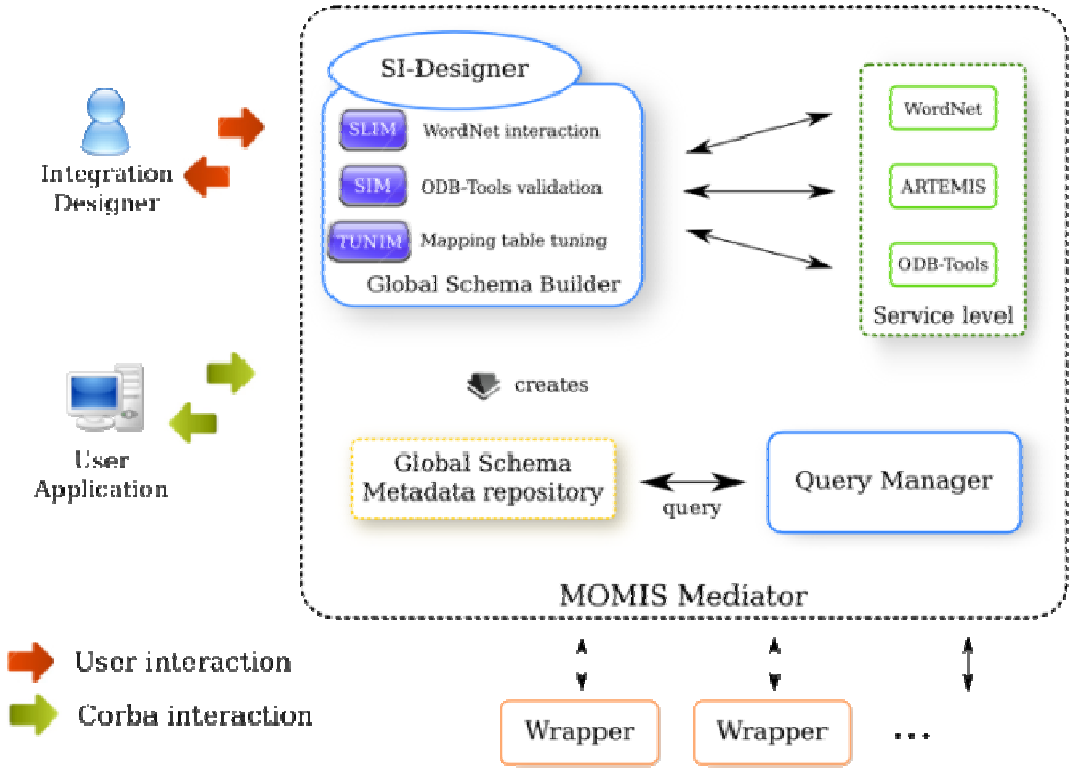

Figura 3 Architettura del Sistema Informativo MOMIS

In Figura 3 è illustrata dettagliatamente l'architettura di MOMIS e si evidenzia in particolar modo la struttura a 3 livelli:

- **Livello Dati**: posti al di sopra delle sorgenti si trovano i **Wrapper** che rappresentano i moduli di interfaccia tra il mediatore e le sorgenti locali. I wrapper hanno una duplice funzione:
  - in fase di integrazione forniscono una descrizione delle informazioni contenute nelle sorgenti, attraverso l'utilizzo del linguaggio ODL_I^3;
  - in fase di *Query Processing* traducono l'interrogazione ricevuta dal mediatore, espressa in linguaggio OQL_I^3 in una interrogazione espressa in un linguaggio comprensibile alla sorgente locale. Inoltre si occupano dell'esportazione dei dati verso il mediatore attraverso il modello comune dei dati utilizzato dal sistema.
- **Livello Mediatore**: cuore del sistema, presiede all'esecuzione di diverse operazioni assegnate a loro volta a 3 diversi moduli:
  - **Global Schema Builder(GSB)**: modulo adibito alla costruzione dello *Schema Globale*.

Il modulo riceve in input dai Wrapper le descrizioni delle sorgenti locali espresse in linguaggio ODL_I^3. Attraverso altri moduli ausiliari come **ODB-Tools**, **WordNet**, il modulo costruisce la vista virtuale integrata, **Global Virtual View(GVV)**, utilizzando tecniche di *clustering* e di Intelligenza Artificiale. Inoltre attraverso il tool di ausilio **SI-Designer** è prevista l'interazione con l'utente, il quale inserisce regole per il *mapping* automatico e interviene laddove esistano processi non eseguibili automaticamente, come ad esempio l'assegnazione del nome ad una classe globale o la modifica delle annotazioni lessicali;

- **Query Manager**: modulo adibito al *query processing* . Esso genera le interrogazioni in linguaggio OQL_I^3 da inviare ai Wrapper partendo dalla singola query formulata dall'utente sullo schema globale. Avvalendosi di tecniche di *Descriptions Logic* di ODB-Tools, il Query Manager scompone automaticamente la query nelle relative sottoquery da inviare alle sorgenti locali;
- **SI-Designer**: interfaccia grafica (GUI) per l'interazione con l'utente durante le fasi dell'integrazione. Diviso anch'esso in moduli in base alla funzionalità:
  - **SIM**(**S**ource **I**ntegrator **M**odule): estrae le relazioni inter-schema sulla base della struttura delle classi ODL_I^3 e delle sorgenti relazionali usando **ODB-Tools**;
  - **SLIM**(**S**ource **L**exical **I**ntegrator **M**odule): estrae le relazioni inter-schema tra nomi di attributi e classi di ODL_I^3 sfruttando il database lessicale **WordNet**;
  - **TUNIM**(**Tuni**ng of the **M**apping Table): modulo che gestisce la creazione dello Schema Globale;

- **Livello Utente**: livello che consente all'utente di poter interagire con lo Schema Globale. L'accesso ai dati risulta trasparente all'utente, è il sistema ad occuparsi delle operazioni necessarie a recuperare le informazioni e combinare i risultati di una query in una risposta corretta. Fa uso di tools di ausilio:
  - **ODB-Tools**: strumento software sviluppato presso l'**Università di Modena e Reggio Emilia**. Si occupa della validazione degli schemi dell'ottimizzazione semantica di interrogazioni rivolte a sistemi **OODB**(**O**bject-**O**riented ODL_I^3 **d**ata**b**ase);
  - **WordNet**: database lessicale descritto nel capitolo 3.1;
  - **ARTEMIS**: tool basato su tecniche di *clustering affinity-based*. Compie l'analisi e il *clustering* sulle classi ODL_I^3 .

1.3.2 Il processo di Integrazione

L'integrazione di sorgenti d'informazione eterogenee strutturate o semi-strutturate è compiuta all'interno di MOMIS in maniera semi-automatica [10], utilizzando gli schemi locali in linguaggio ODL_I^3 e combinando tecniche di *Description Logic* e di *clustering*.

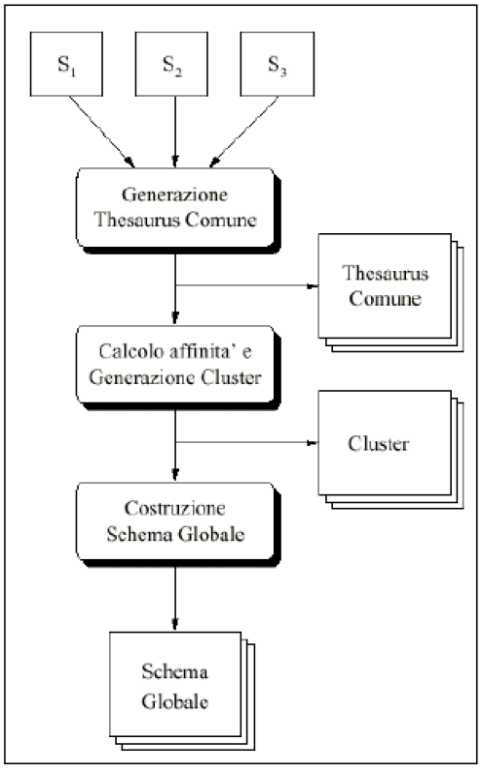

Figura 4 Processo di Integrazione

Come mostrato in Figura 4, i passi compiuti durante il processo di integrazione sono i seguenti:

- *Generazione del Common Thesaurus*: attraverso il supporto di ODB-Tools e di WordNet in questa fase vengono identificate una serie di relazioni terminologiche che esprimono la conoscenza inter-schema e intra-schema su sorgenti eterogenee. Sono derivate in maniera semi-automatica a partire dalle descrizioni dagli schemi in ODL_I^3, attraverso l'analisi strutturale (utilizzando ODB-Tools e le tecniche di *Description Logic*) e di contesto (attraverso l'utilizzo del database lessicale WordNet) delle classi coinvolte.
- *Generazione dei cluster di classi* ODL_I^3: con il supporto di **ARTEMIS**. Le relazioni terminologiche del Theaurus sono utilizzate per valutare il livello di similarità tra le classi di ODL_I^3 in modo da identificare informazioni che devono essere integrate a livello globale. A questo proposito ARTEMIS calcola i coefficienti che misurano il grado di affinità tra le

classi, basandosi sia sui nomi delle stesse sia sugli attributi che contengono. Attraverso tecniche di *clustering* le classi che presentano una elevata affinità vengono raggruppate [11].

- *Costruzione dello Schema Globale*: I cluster ottenuti precedentemente e che risultano essere affini tra loro sono analizzati per costruire lo Schema Globale del Mediatore. Per ciascun di essi si definisce una Classe Globale che rappresenta tutte le classi locali riferite al cluster ed è caratterizzata dall'unione ragionata dei loro attributi e da una *mapping-table*. Questo insieme di classi globali costituisce lo Schema Globale che sarà utilizzato per porre le query alle differenti sorgenti locali.

2 Wrapper JDBC/PostgreSQL

2.1 I servizi di Wrapping in un'architettura I^3

Facendo riferimento all'architettura I^3 descritta nel capitolo 1.2.1, i servizi di Wrapping sono quei servizi che rendono le diverse sorgenti di informazione conformi ad uno standard interno o esterno. La loro funzionalità si può riassumere nel consentire la comunicazione fra i sistemi locali e i servizi di alto livello del mediatore. In particolare si concentrano su due obiettivi:

- Permettere ai servizi di coordinamento di trattare le diverse sorgenti locali in modo uniforme;
- Essere il più scalabili e modulari possibili, in modo da poterne permettere la riusabilità. Devono quindi fornire delle interfacce che seguano gli standard più diffusi(SQL per l'interrogazione dei dati, CORBA come protocollo di scambio oggetti). In questo modo le diverse sorgenti locali sarebbero accessibili da diversi moduli mediatori.

Un wrapper non è altro che un modulo che permette di standardizzare il processo di *wrapping* delle sorgenti, permettendo la creazione di una libreria di fonti accessibili. Prestando maggior attenzione ai servizi di Wrapper primari è possibile costruire una classificazione in base alla loro funzionalità:

- **Communication Wrapping Service**: insieme dei servizi di wrapping che si occupano di gestire la comunicazione fra oggetti, eventi, metodi e chiamate alle funzioni tra sorgenti locali e servizi di più alto livello;
- **Data Restructuring Wrapping Service**: insieme dei servizi di wrapping che gestiscono la trasformazione dei dati e dei metadati di sorgenti locali a livello macchina;
- **Behavioral Transformation Wrapping Service**: insieme dei servizi di wrapping che gestiscono le trasformazioni semantiche del programma, eventuali modifiche dei protocolli di sistema o la traslazione fra i diversi dialetti SQL.. Spesso fanno uso di moduli interni per "esporre" le interfacce per un uso indipendente o un'eventuale modifica.

Uno degli obiettivi di questa Tesi è stata la creazione di un Data Restructuring Wrapping Service che consentisse la connettività fra il sistema informativo MOMIS e diverse sorgenti situate su un DBMS PostgreSQL.

2.2 I Wrapper in Momis e il Wrapper PostgreSQL

Nel capitolo 1.2.1 si è brevemente accennato al fatto che in MOMIS sono presenti dei servizi di Wrapping implementati da diversi moduli Wrapper sviluppati ad hoc nel corso degli anni di vita del progetto. Poiché MOMIS si propone innanzitutto di offrire le funzionalità proprie di un software *mediatore* si è presentata in questi anni la necessità di creare i Wrapper dedicati alle sorgenti dati più comuni, SQL e non, in maniera tale da consentire a MOMIS di potersi interfacciare e di poter utilizzare queste sorgenti, ampliando in questo modo la gamma di operatività del progetto stesso. All'interno del sistema informativo MOMIS il Wrapper è un oggetto **CORBA** [9], connesso ad una sorgente, in grado di descriverne la struttura e di inviare interrogazioni mediante il linguaggio ODL_I^3.

Lo stato dell'arte dei Wrapper all'interno di MOMIS prima di questa Tesi era il seguente:

- Corba Wrapper Client;
- Wrapper JDBC generico;
- Wrapper Oracle;
- Wrapper Sql Server;
- Web Service Wrapper Client;
- Wrapper XML;
- Wrapper OWL;
- Wrapper JDBC/ODBC;
- Wrapper XSD;
- Wrapper MySql;
- Wrappers per file;

Uno degli obiettivi di questa Tesi è stata la creazione di un Wrapper per sorgenti dati PostgreSQL.

2.3 Il DBMS PostgreSQL

PostgreSQL[2] è un **object-relational database management system** (ORDBMS) open-source. Inizialmente sviluppato dalla University of California at Berkeley Computer Science Department e continuamente migliorato grazie al contributo di centinaia di sviluppatori.

Storicamente è una evoluzione di **Ingres**[3], progetto della University of California per un DBMS open-source e non a caso il nome deriva dalla contrazione di **post**-In**gres**, sviluppo nato in seguito alla commercializzazione da parte della società Relational Technologies del progetto padre. Il neonato progetto puntava a fornire un supporto completo ai tipi di dati standard del linguaggio SQL e a consentire all'utente di definire nuovi tipi di dati (UDT, User Defined Types). Il core centrale di Ingres assunse allora caratteristiche di un modello object-oriented.

E' possibile scaricare PostgreSQL direttamente dal sito ufficiale, www.postgresql.org, in diversi formati: file auto-installante disponibile per diversi Sistemi Operativi, file contenenti i sorgenti del codice del programma o Cd-Live. Inoltre è possibile scaricare la documentazione completa e aggregarsi alla Community di sviluppatori[4].

2.3.1 Descrizione

PostgreSQL è quindi un sistema per la gestione di database relazionali(**RDBMS**) che assume le caratteristiche tipiche di un linguaggio **object-oriented** ovvero presenta le seguenti caratteristiche:

- definizione di Oggetti;
- definizione di Classi;
- Ereditarietà;

Grazie a queste caratteristiche permette agli utenti la possibilità di estendere il sistema tramite la definizione di nuovi tipi di dati(**UDT**: User Defined Types), operatori e funzioni all'interno del database stesso. Questo approccio risulta particolarmente rilevante nell'offrire soluzioni valide anche in campi come i **Sistemi Informatici Territoriali**(**GIS**), per i quali si ha spesso la necessità di definire dei tipi di dato per la memorizzazione di Linee e Aree.

Oltre a garantire una stabile integrità referenziale e una gestione avanzata delle transazioni, PostgreSQL mette a disposizione in modo nativo e maturo funzionalità come:

- viste

[2] Official Site http://www.postgresql.org/
[3] Official Site http://www.ingres.com/
[4] Community Site http://www.postgresql.org/community/

- schemi
- trigger
- tablespace
- stored procedure (in diversi linguaggi di programmazione)
- interfacce di connessione (in particolare ODBC e JDBC)
- UNICODE
- two-phase commit
- alta disponibilità

A partire dalla versione 8, PostgreSQL ha ampiamente migliorato la sua scalabilità e le sue performance, tanto da risultare adatto a data warehouse, data mining e a sistemi di supporto alle decisioni in genere.

Un primo approccio potrebbe far presagire che PostgreSQL sia uguale agli altri DBMS, in quanto mantiene le peculiarità fondamentali di questi sistemi. Usa il linguaggio SQL per eseguire query sui dati, i quali a loro volta sono memorizzati su una collezione di tabelle con chiavi interne ed esterne. Il punto di forza di PostgreSQL è rappresentato invece dall'estendibilità e dalla programmabilità.

Normalmente sviluppare un' applicazione multi-level con un linguaggio object-oriented e basata su un database presenta delle difficoltà dovute principalmente al fatto che i due contesti utilizzano modelli di organizzazione dei dati molto differenti. I database SQL conservano i dati in semplici "*flat-tables*", richiedendo che sia l'utente a prelevare e raggruppare le informazioni correlate utilizzando le query o semplici funzioni definite internamente, più o meno standard a seconda del DBMS utilizzato. Ciò entra in contrasto con il modo in cui sia le applicazioni e sia gli utenti utilizzano i dati, e cioè mediante delle strutture maggiormente complesse dove tutti i dati correlati operano come elementi completi , normalmente definiti Oggetti o Record. Si rende perciò necessario predisporre una mappatura fra i diversi modelli per permettere a questi due diverse strutture di poter comunicare.

Queste difficoltà, che nascono come detto a causa della differente organizzazione dei dati , vengono chiamate generalmente "*impedance mismatch*" [12](discrepanza di impedenza) e arrivano ad occupare anche il 35-40% del tempo di sviluppo di un progetto. Sebbene esistano nei DBMS di maggior utilizzo soluzioni riguardo questa mappatura, chiamate **object-relational mapping**, non sempre sono utili da un punto di vista prestazionale e economico.

PostgreSQL può risolvere molti di questi problemi direttamente nel database [13], in prima istanza permettendo all'utente di poter definire dei nuovi tipi di dati complessi basati sui tipi di dati SQL-standard, e successivamente al database stesso di comprendere questa nuova tipologia di dati.

Per esempio è possibile definire il tipo di dato "*Indirizzo*" come insieme di stringhe e numeri per rappresentare la Città, la Via e il Numero Civico e creare una tabella che contenga tutti i campi necessari per memorizzare l'indirizzo in una sola linea di codice, e quindi di poter utilizzarlo come *data type* di un attributo.

In PostgreSQL viene implementata anche una delle principali caratteristiche della programmazione orientata agli oggetti, l'Ereditarietà [14], applicabile sia ai tipi di dato definiti dall'utente, sia alle stesse tabelle del database. Partendo dalla definizione di un tipo di dato complesso e astratto (**ADT**: Abstract Data Type) è possibile implementare degli ulteriori tipi di dato che provvedono ad estendere l'ADT di partenza. Un esempio può essere mostrato con la definizione di un tipo di dato "*Codice Postale*" e dalle successive definizioni dei tipi di dato "*Cap*" e "*Us Zip Code*" che estendono il primo creato.

Rispetto ad altri DBMS PostgreSQL presenta anche un notevole vantaggio riguardo la programmabilità [15]. La maggior parte dei sistemi SQL permette agli utenti di scrivere delle istruzioni o delle funzioni richiamabili in fase di lettura o scrittura dei record. L'SQL Standard risulta però essere un linguaggio poco adatto alla programmazione e alla definizione di algoritmi, risultando lacunoso qualora si debbano costruire logiche complesse o più semplicemente quando si debba far uso degli operatori logici più comuni (*if..else, for, while..*), normalmente supportati dai linguaggi di programmazione più usati. E' per questo motivo che la maggior parte dei competitor presenti sul mercato dell'IT sceglie di estendere l'SQL secondo i propri standard. Queste soluzioni non risultino portabili su diverse piattaforme database e che quindi costituiscano dei limiti vincolanti nell'ideazione e nello sviluppo di una applicazione.

Al contrario, PostgreSQL consente invece agli sviluppatori di implementare funzioni secondo alcuni linguaggi supportati e secondo alcuni moduli software presente all'interno del sistema:

- **PL/pgSQL**, un linguaggio nativo, simile al linguaggio procedurale di **Oracle PL/SQL**, vantaggioso nelle procedure che fanno un uso intensivo di query;
- Wrapper per i principali linguaggi di scripting come **Perl**, **Python** e **Ruby**;
- **C** e **C++** per procedure con logica particolarmente complessa;

Queste caratteristiche, unite ai vantaggi che si hanno sulle prestazioni e sull'affidabilità rendono PostgreSQL uno dei più avanzati sistemi database dal punto di vista della programmabilità.

Per concludere è fondamentale dare qualche nozione sulla struttura e sulla memorizzazione fisica dei dati. In fase di creazione/modifica di un database, viene assegnato un "*User*", il quale assume quindi il ruolo di *Owner* ed è quindi autorizzato alla creazione e alla modifica dei diversi schemi. Come per altri DBMS, uno schema non è altro che una collezione di Tabelle o Viste, atte alla memorizzazione dei record. La gerarchia delle entità di PostgreSQL è quindi :

database :: schema :: tabella.

Fisicamente però la memorizzazione del database avviene mediante una variabile d'ambiente impostata automaticamente con l'installazione di PostgreSQL: **PG_DATA**. Questa definisce una cartella all'interno del folder principale di PostgreSQL all'interno della quale vengono memorizzati i dati. E' possibile tuttavia definire dei **TABLESPACES**, che consentono un'allocazione fisica diversa da quella che propone PostgreSQL e associata allo USER stesso scelto durante la creazione/modifica del Database. In Figura 5 è mostrato un semplice schema che riassume la struttura di un Database PostgreSQL.

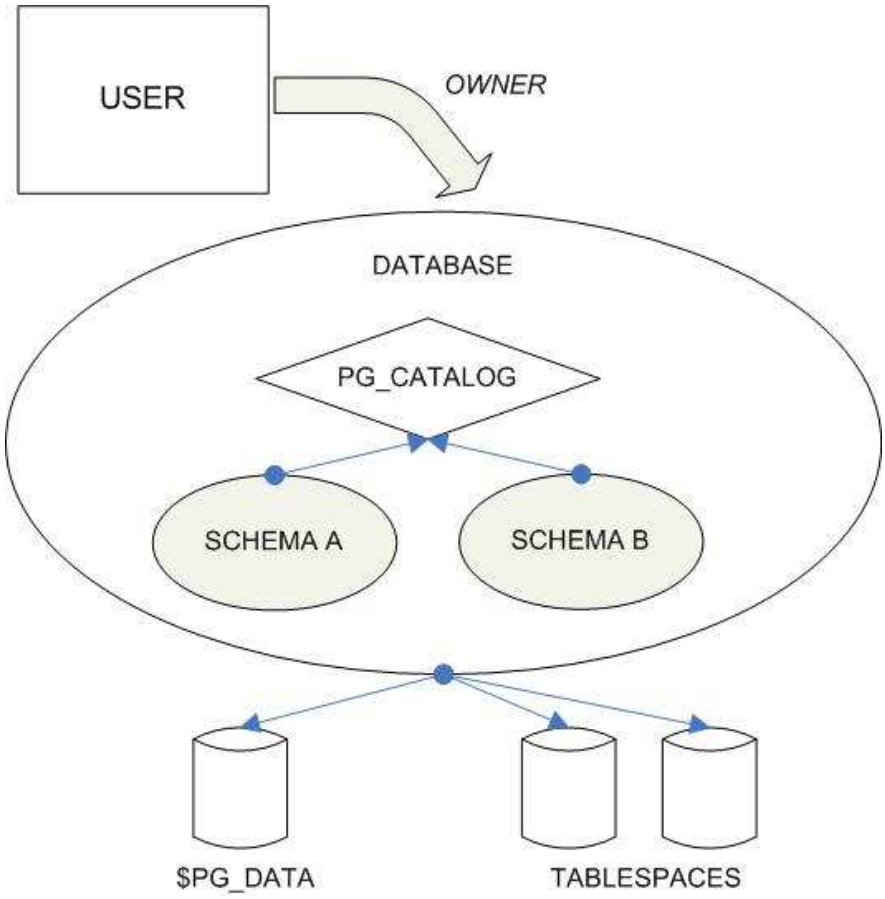

Figura 5 Struttura di un database PostgreSQL

Al momento della creazione di un database viene creato automaticamente lo schema **PG_CATALOG**, all'interno del quale sono presenti le tabelle che si occupano di memorizzare ogni singola informazione relativa al database, agli schemi che verranno eventualmente creati e adoperati e alle tabelle dei singoli schemi. In Figura 6 è riportata il menu ad albero che definisce la struttura di un database all'interno di PostgreSQL.

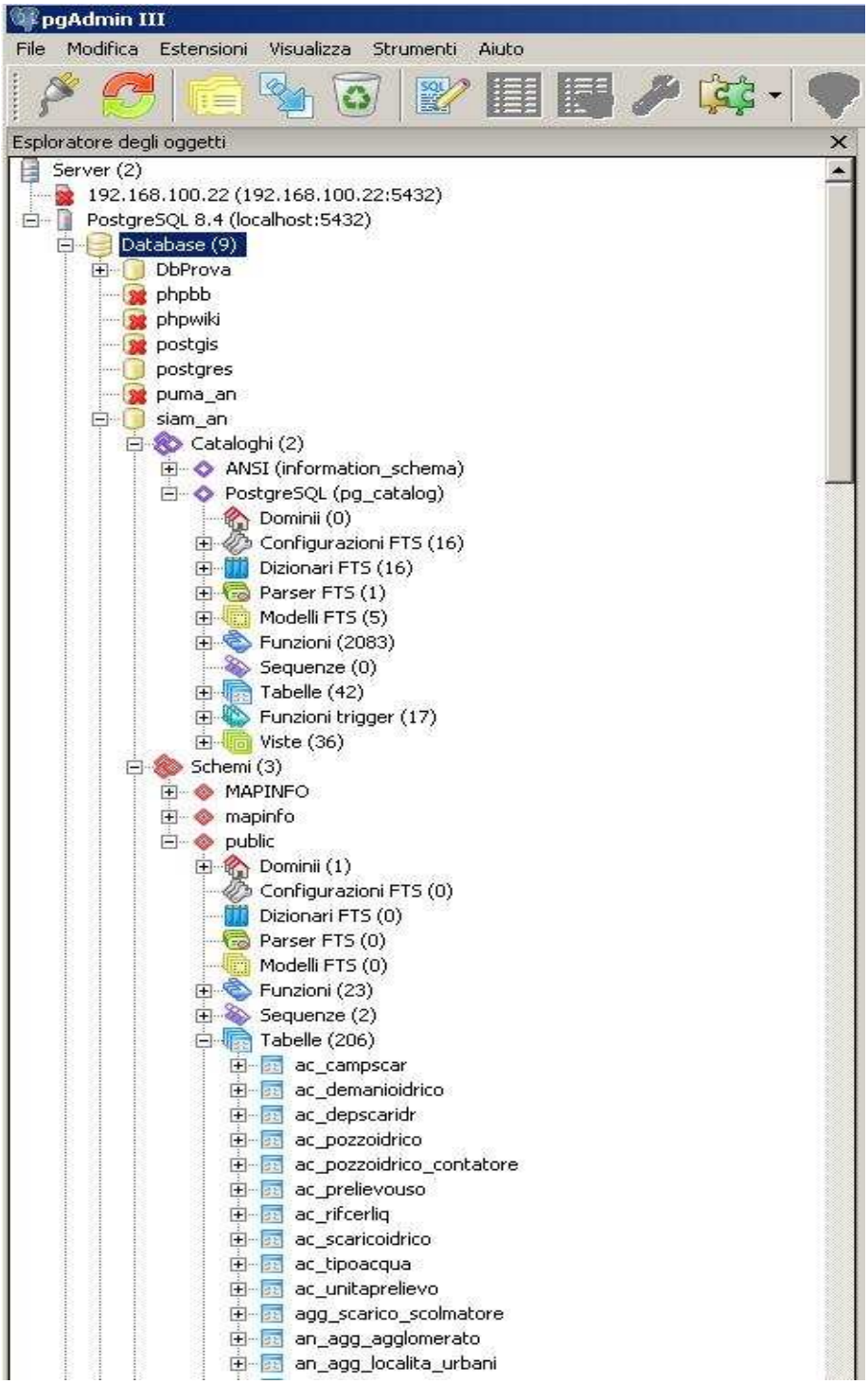

Figura 6 Interfaccia Client pgAdmin III e struttura di un database PostgreSQL

2.4 I driver JDBC/PostgreSQL

La tecnologia **JDBC**(Java Database Connectivity) consiste in un insieme di **API** che permettono l'accesso a diverse sorgenti dati tabulari dal linguaggio di programmazione **JAVA**. Consente quindi l'accesso a vari database SQL e a sorgenti di tipo tabulare come i fogli elettronici. Il vantaggio di questa tecnologia è quello di creare un accesso virtuale grazie alla Java Virtual Machine. Ogni programma che usi questa tecnologia sarà quindi in grado comunicare con diverse sorgenti dati differenti previa implementazione del relativo driver JDBC e non sarà necessaria quindi la creazione di software specifici per il tipo di sorgente.

Essenzialmente le funzionalità di base di un driver JDBC sono 3:

- Creare una connessione con una sorgente dati: dopo aver caricato il driver mediante il metodo **`Class.forName(String className)`** si provvedere ad istanziare un oggetto Connection mediante il metodo **`DriverManager.getConnection(String url).`** Tale oggetto rappresenta la connessione al database nella quale vengono continuamente inviati gli statement SQL.

- Interrogare la sorgente dati: si provvede a utilizzare degli oggetti **`Statement`** per eseguire degli query SQL senza parametri, con parametri o per invocare una Stored Procedure. In seguito alla creazione degli oggetti **`Statement`** mediante il metodo **`createStatement()`** della classe **`Connection`**, questi vengono inviati al database per interrogare la sorgente dati attraverso i metodi **`executeQuery()`** e **`executeUpdate()`** degli stessi Statement, a seconda che si stia eseguendo una query di "selezione" o di ""aggiornamento".

- Elaborare i risultati di un'interrogazione: in seguito all'invio degli Statement il database risponde con un oggetto **`ResultSet`** che rappresenta i risultati dell'interrogazione.

L'utilizzo di questi driver consente al Sistema Informativo MOMIS di realizzare il collegamento tra i diversi Wrapper presenti e le sorgenti dati su cui lavorare (in questo Tesi si tratteranno esclusivamente sorgenti di dati su DBMS PostgreSQL).

Attraverso il sito http://jdbc.postgresql.org è possibile scaricare i file *.jar* dei driver PostgreSQL e

la relativa documentazione. Sono driver JDBC di tipo **IV**[5], per cui sviluppati totalmente in linguaggio Java, indipendenti dalla piattaforma di utilizzo e utilizzabili semplicemente includendoli nel *classpath* del proprio progetto, in questo caso il Sistema Informativo MOMIS.

Nel corso degli anni sono stati rilasciate diverse versioni dei driver in concomitanza con lo sviluppo della tecnologia Java e di conseguenza della Java Virtual Machine. Per queste ragioni all'interno del progetto MOMIS sono stati inclusi 4 driver jdbc/PostgreSQL.

- **`Postgresql-8.3-603.jdbc2.jar`**: driver compatibile con JDK 1.2 e 1.3;
- **`Postgresql-8.3-603.jdbc2ee.jar`**: driver compatibile con JDK 1.3 + J2EE. Contiene un supporto addizionale per le classi del package `javax.sql`.
- **`Postgresql-8.3-603.jdbc3.jar`**: driver compatibile con JDK 1.4 e 1.5. Contiene un supporto per SSL e le classi del package `javax.sql`, ma non necessita della J2EE.
- **`Postgresql-8.3-603.jdbc4.jar`**: driver compatibile con JDK 1.6

Come precedentemente detto, per caricare i driver nel progetto basta semplicemente aggiungerli al classpath come librerie, mentre per caricare i driver all'interno di una classe si usa il metodo:

```
Class.forName("org.postgresql.Driver") throw ClassNotFoundException;
```

Per creare invece la connessione con il database con il metodo **`DriverManager.getConnection()`** la stringa da passare come parametro è:

```
jdbc:postgresql://host:port/database?user=aUser&password=aPassword
```

dove:

- **`Host`**: è l'host name del server. Di default è *localhost*. E' possibile specificare un indirizzo di tipologia IPv6 mediante l'utilizzo della seguente sintassi:

```
jdbc:postgresql://[::1]:5740/accounting
```

- **`Port`**: è la porta sulla quale il server si mette in ascolto. Di default, per PostgreSQL, è impostata a **5432**;

[5] http://java.sun.com/products/jdbc/driverdesc.html

- **`Database`**: è il nome del database;
- **`User`** è l'utente con cui si accede al server PostgreSQL;
- **`Password`**: è la password relativa all'utente, utilizzata per l'accesso al server PostgreSQL;

.

2.5 Interfacce Wrapper e WrapperCore

La classe **`Wrapper`** in MOMIS è un interfaccia CORBA che tutti i Wrapper presenti implementano. Nel caso in esame, il Wrapper che dovrà essere sviluppato è un Wrapper di tipo JDBC che presenta la seguente gerarchia:

`it.unimo.dbgroup.momis.communication.Wrapper`

- `it.unimo.dbgroup.momis.communication.WrapperCore`
- `it.unimo.dbgroup.momis.communication.core.jdbc.WrapperJdbcCore`
- `it.unimo.dbgroup.momis.communication.core.jdbc.WrapperJdbcCore_PostgresSQL`

L'interfaccia `Wrapper` dichiara i seguenti metodi:

- **`public String getDescription():`** ritorna una descrizione della sorgente in formato ODL_I[3];
- **`public String getDescriptionXml():`** ritorna una descrizione della sorgente in formato XML;
- **`public String getSourceName():`** ritorna un oggetto String con il nome della sorgente dati alla quale ci si connette;
- **`public String getType():`** ritorna il tipo di sorgente(JDBC, XML,..);
- **`public WrapperRS runQuery(String oql):`** esegue una query in formato **OQL**(Object Query Language);
- **`public void close():`** chude la connessione del Wrapper.

La classe `Wrapper` viene estesa dall' interfaccia **`WrapperCore`** dichiarando ulteriori metodi:

- **`public void initialize(Map parameters)`**: inizializza i parametri di cui il Wrapper ha bisogno per poter funzionare;
- **`public String getParametersAsPropertiesTemplate()`** : ritorna tutti i parametri di cui il Wrapper ha bisogno per funzionare in un formato tale da poter essere editato in un file di testo;

2.6 Il Wrapper JDBC

Il Wrapper JDBC è stato sviluppato per sorgenti relazionali interfacciabili tramite una connessione JDBC. La classe **`WrapperJdbcCore`** implementa l'interfaccia `WrapperCore`. Definisce quindi ulteriori metodi, fra cui il metodo `getSchema` che recupera l'oggetto **`Schema`**, classe ODL_I^3, che descrive la fonte dati integrata mediante l'istanziazione, a sua volta, di diverse classi ODL_I^3:

- **`Source`**: descrive la fonte dati da estrarre e interrogare;
- **`Interface`**: descrive la singola tabella della fonte dati;
- **`IntBody`**: descrive un oggetto che rappresenta l'insieme degli attributi di una tabella;
- **`SimpleAttribute`**: descrive un attributo della tabella;

Non rappresentando ancora un tipo di sorgente tabellare, dovrà ancora essere in qualche modo specializzata e quindi, di conseguenza, estesa. Oltre ad implementare i metodi esposti dalle due classi "*padre*", la classe `WrapperJdbcCore` definisce 3 ulteriori metodi per i quali è doveroso dare una descrizione:

- **`public Schema getSchema():`** metodo che recupera i metadati della connessione e inserisce in uno schema ODL_I^3 i componenti dello schema relazionale estratti(nomi tabelle, colonne, tipi di dato). Non estrae ancora le *constraint* di un entità;
- **`public boolean theTableShouldBeDescribed(String tableName)`**: metodo che ritorna un valore booleano(*true* o *false*) in base all'integrazione o meno della tabella in esame.
- **`public static Type getAttribute(int t)`**: metodo che prende come parametro un intero, che rappresenta un tipo di dato JDBC, e ne ritorna il rispettivo oggetto ODL_I^3;

In conclusione con un Wrapper di questo tipo si è in grado di estrarre lo schema di una sorgente, inteso come insieme di tabelle, di attributi e di tipi di dato, ma non si è ancora in grado di poter definire Primary Key e Foreign Key delle relative tabelle e quindi di completare con coerenza l'effettiva struttura di uno schema. Inoltre con un Wrapper di questo tipo non si è ancora in grado di escludere le tabelle di sistema presenti all' interno di un DBMS.

2.7 Il Wrapper JDBC/PostgreSQL

Il Wrapper per sorgenti PostgreSQL è stato sviluppato attraverso la classe **`WrapperCore_PostgreSQL`** all'interno del package `it.unimo.dbgroup.momis.communication.core.jdbc`.

Come visto nel paragrafo precedente, con il Wrapper JDBC non si è ancora in grado di poter escludere dall'interrogazione del Wrapper le tabelle di Sistema, ne di poter definire le *Primary Key*(PK) e *Foreign Key*(FK) delle diverse entità. Nella classe che implementa il Wrapper PostgreSQL sono stati definiti alcuni metodi in grado di fornire una soluzione a questa lacuna.

Per definire con esattezza le *constraints* si è utilizzato l'oggetto **`DataBaseMetadata`** con il quale si sono possono recuperare diverse peculiarità della sorgente alla quale ci si collega, comprese le PK e le FK dello schema. Per risolvere invece il problema riguardante le tabelle di sistema, attraverso opportune interrogazioni allo schema **PG_CATALOG** è stato possibile recuperare ogni informazioni relative agli schemi che effettivamente devono essere oggetto di successive integrazioni.

Operativamente l'implementazione centrale del Wrapper PostgreSQL ha comportato quindi la riscrittura di alcuni metodi descritti nelle Classi "*padre*".

- **`public Schema getSchema()`:** Attraverso l'instanziazione di un oggetto `DataBaseMetadata` è possibile recuperare un elenco delle tabelle dello schema al quale ci si collega e si evita quindi di recuperare entità non necessarie. Ogni entità risulta poi identificata da un attributo denominato **TABLE_TYPE**. Effettuando la scelta di considerare valori validi per questo attributo "*TABLE*" e "*VIEW*" si ottiene un insieme ancora più preciso delle tabelle. Infine, attraverso l'utilizzo del metodo `tableShouldBeDescribed(String tableName)`, descritto successivamente, si ottiene l'elenco definitivo delle tabelle che compongono effettivamente lo schema. Ottenuto questo insieme di Tabelle il metodo si occupa anche di recuperare le *constraints* dello schema, Interrogando ricorsivamente lo schema e analizzando le sue entità, tabelle e attributi, viene recuperata e definita la Primary Key di ogni tabella, e si verifica se siano presenti o meno delle Foreign Key. In seguito si provvede alla creazione e al popolamento delle classi ODL_I^3 descritte nel paragrafo 2.5. Il valore di ritorno del metodo sarà un oggetto `Schema`, il quale rappresenta coerentemente ora la sorgente alla quale ci si collega.
- **`public boolan the tableShouldBeDescribed(String tableName)`**: all'interno del metodo viene implementata una procedura che confronta il nome di una tabella, passata come parametro, con i nomi delle tabelle di sistema. Qualora vi sia una

corrispondenza fra stringhe il metodo torna un valore booleano settato a *false* in quanto la tabella non dovrà prendere parte al processo di Integrazione.

In figura 7 si riporta il Class Diagram della gerarchia delle classi relative al Wrapper realizzato per sorgenti PostgreSQL.

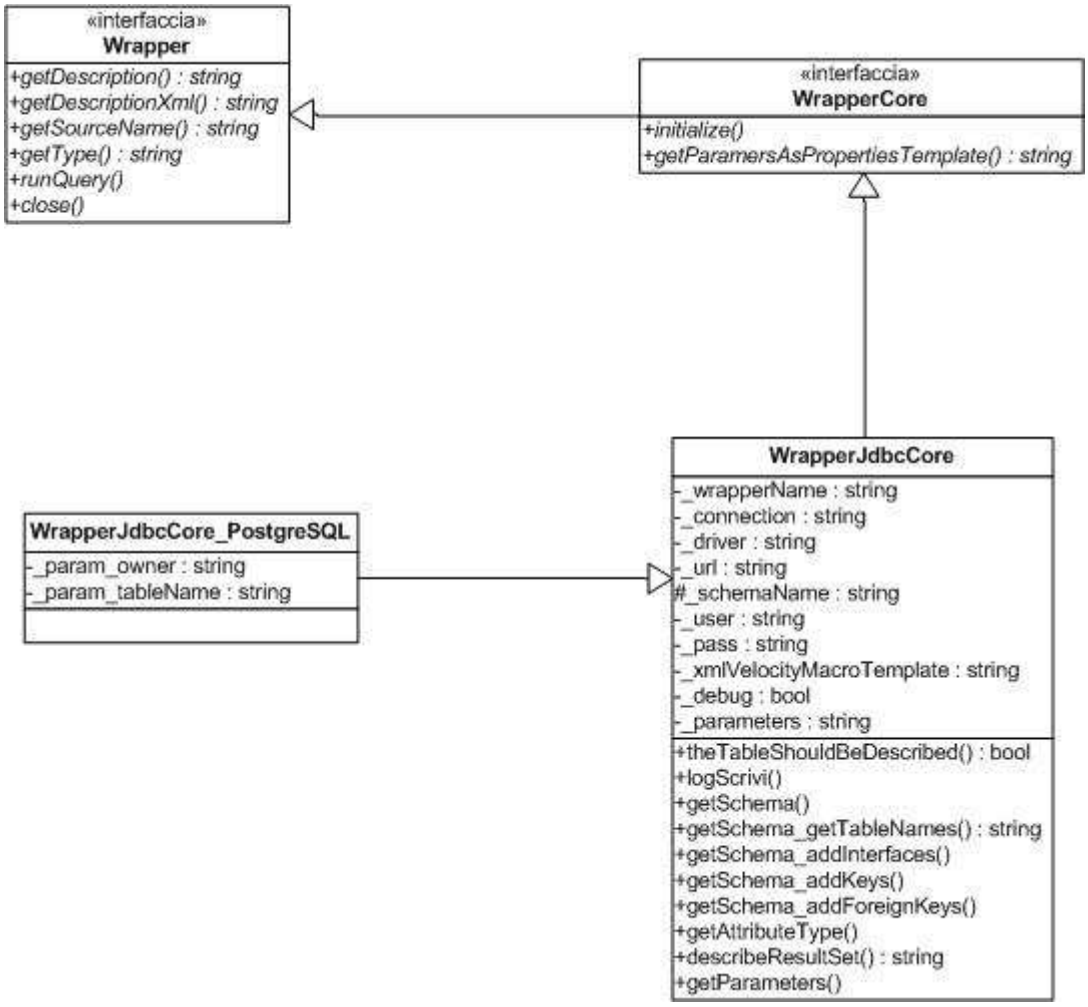

Figura 7 Gerarchia delle classi del Wrapper_PostgreSQL

2.8 Interfaccia grafica del Wrapper PostgreSQL

Similarmente ad altri Wrapper già implementati in MOMIS e utilizzabili attraverso il tool **SI-Designer** [16], in questa Tesi è stata creata un' interfaccia grafica come elemento conclusivo dell'implementazione del Wrapper per sorgenti PostgreSQL.

Tale interfaccia è stata creata mediante la creazione della classe **`PostgreSqlWrapperGUI`** all'interno del package `it.unimo.dbgroup.momis.SIDesigner.SAM.SAMGUI`.

L' interfaccia grafica permette all'utente di poter utilizzare il Wrapper in maniera più semplice ed intuitiva. Innanzitutto non è necessario specificare il driver JDBC che si andrà ad utilizzare, in

quanto basta scegliere il tipo di Wrapper che si vuole utilizzare, ovviamente in questo caso il Wrapper PostgreSQL, per caricare il driver JDBC corretto. Mediante l'utilizzo di campi di testo è possibile inserire i diversi parametri che solitamente vanno a formare la stringa JDBC necessaria per la connessione ad un database:

- l'indirizzo o il nome dell'host sul quale è presente la sorgente alla quale ci si vuole interfacciare ;
- il nome e la password dell'utente del servizio del DBMS;
- il nome stesso dello schema oggetto del collegamento e della successiva integrazione. Come per altri Wrapper è inoltre presente la possibilità di scegliere lo schema al quale potersi connettere mediante la visualizzazione di una lista di schemi presenti sull'host specificato nel parametro precedente;
- la stringa di connessione JDBC, composta mediante l'inserimento dei parametri precedenti;

Oltre a queste funzionalità è stata anche inserita la possibilità di scegliere la lingua con la quale il database è definito e che verrà utilizzata successivamente in fase di Annotazione Manuale e Automatica. Una più esaustiva analisi di questa implementazione è descritta nel capitolo 3.3.

In Figura 8 viene riportata per completezza una schermata dell'interfaccia grafica implementata per il Wrapper PostgreSQL, nella quale si possono notare tutti i parametri specificati poco sopra.

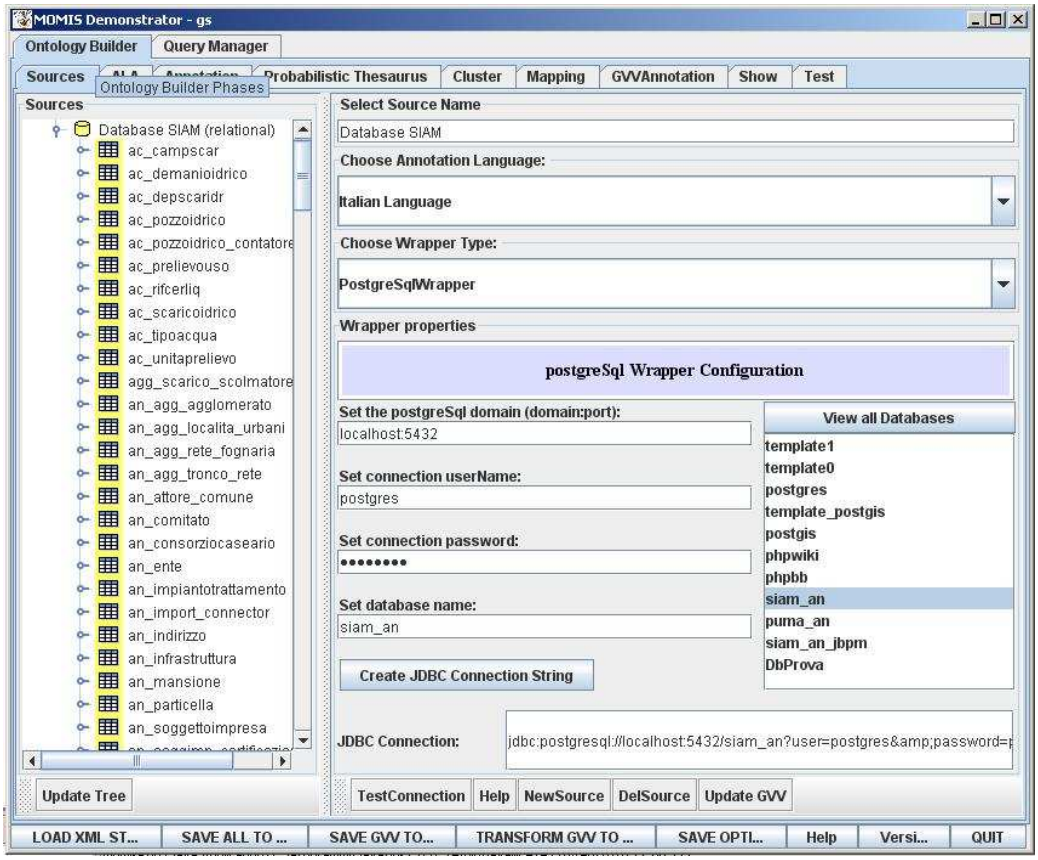

Figura 8 Interfaccia grafica del Wrapper PostgreSQL all'interno di MOMIS

3 Sviluppo di un database multilingua per l'utilizzo in MOMIS

Uno degli obiettivi che si propone questa Tesi è la realizzazione dell'integrazione di sorgenti multilingua all'interno del Sistema Informativo **MOMIS** e del loro utilizzo in fase di interrogazione dello schema globale. Come descritto nel capitolo 1.4, una delle funzionalità principali di MOMIS è l'annotazione semi-automatica o manuale delle entità di una sorgente dati rispetto ad un database . A partire dalle annotazioni sulle diverse entità MOMIS genera delle relazioni lessicali che vengono inserite nel *Common Thesaurus* e forniscono informazioni dettagliate sulle relazioni inter-schema / intra-schema contesto tra le sorgenti.

Il database lessicale attualmente presente in MOMIS è una la versione 2.0 di **Princeton WordNet** [17], successivamente estesa con WordNet Domains. Data questa impostazione il processo di annotazione è stato sinora condotto sulla base della sola lingua inglese. Poiché è presumibile immaginare una possibile integrazione fra sorgenti multilingua è necessario provvedere alla realizzazione di una metodologia che consenta un' annotazione multilingua.

La costruzione di un database lessicale multilingua utilizzabile all'interno di un progetto come MOMIS può presentare due diversi tipi di soluzione. E' possibile infatti utilizzare una sola sorgente *monolingua* per l'annotazione semiautomatica e procedere manualmente con l'aggiunta di nuovi termini in lingue differenti e andando in seguito a reperire il corrispondente termine inglese. Questa risulta anche essere la soluzione adottata fino a questo momento.

La seconda soluzione invece consiste nel procedere alla creazione di un database lessicale *multilingua* che permetta l'annotazione semiautomatica anche secondo lingue diverse da quella originale e che permetta di annotare sia le relazioni semantiche comuni alle diverse lingue oggetto di integrazione piuttosto che specifiche di una data lingua.

In questa tesi si propone una implementazione mediante l'analisi di due database lessicali multilingua, **MultiWordNet** e **EuroWordNet** e la conseguente integrazione di uno di essi. La scelta di considerare queste due risorse per l'eventuale integrazione di un dizionario multilingua è stata dettata dalla facilità con cui è possibile reperire entrambe e dalla profonda differenza strutturale consistente in due approcci in netta contrapposizione.

- *Expand Model*: modello che si basa sul presupposto che, tra gli stessi concetti, in linguaggi differenti, intercorrono le stesse relazioni, ovvero, parlando di ontologie, se due *Synset* in WordNet sono legati da una relazione, gli stessi due *Synset* saranno legati dalla stessa relazione anche in un WordNet di una lingua differente. Tale modello presenta

quindi due indissolubili vantaggi: offre un maggior grado compatibilità ed è il più semplice da implementare. Vantaggi dati dal fatto che il modello si basa su una comune base derivata dai *Synset* di Princeton WordNet, i quali sono legati a loro volta attraverso relazioni di equivalenza ai S*ynset* di WordNet. Questo tipo di modello può far scaturire alcuni problemi riguardo le strutture semantiche delle diverse lingue rispetto a quella della lingua inglese. Inoltre non è consentito l'uso di risorse esistente ma comporta una riscrittura completa sempre sulla base di WordNet. Tale modello viene implementato nel database lessicale multilingua **MultiWordNet**, analizzato successivamente e utilizzato in questo elaborato.

- *Merge Model*: modello che si prefigge di mantenere separate le diverse strutture semantiche dei linguaggi. I database specifici di ciascuna lingua sono sviluppati separatamente e autonomamente, mediante l'utilizzo di database e risorse preesistenti, e solo successivamente viene sviluppata la parte che si occupa del collegamento interlinguistico mettendo in relazione i *Synset* dei diversi WordNet fra questi database. Si mantiene in questo modo l'indipendenza strutturale delle diverse lingue. Anche questo modello comporta alcuni svantaggi. Riguardano in questo caso la necessità di poter assicurare una sufficiente sovrapposizione fra i vari WordNet e stabilire come debbano essere interpretate le incongruenze che possono esserci fra questi. Il modello viene implementato nel progetto **EuroWordNet** [18], anch'esso successivamente analizzato ma che invece non verrà utilizzato all'interno dell'elaborato.

3.1 Il database lessicale WordNet ®

Il database lessicale **WordNet** ® nasce presso il *Cognitive Science Laboratory* dell'**Università di Princeton** sotto la direzione del professor **George Miller** [17] nell'anno 1985. E' disponibile gratuitamente tramite una licenza d'uso BSD sul sito http://WordNet.princeton.edu/WordNet/ assieme alla documentazione relativa. La licenza d'uso consente l'utilizzo gratuito a patto che vengano citati gli autori ed il sito ufficiale del progetto.

WordNet è un sistema di gestione di un dizionario lessicale in lingua inglese basato sulle teorie psicolinguistiche della memoria lessicale umana. I termini sono infatti organizzati tenendo conto delle loro affinità di significato. Le categorie sintattiche (nomi, verbi, aggettivi e avverbi) sono organizzati in *insiemi di sinonimi* che rappresentano un inerente concetto lessicale, condiviso da tutti i termini ad esso associati. Gli insiemi di sinonimi sono collegati fra di loro attraverso diversi tipi di relazione.

L'organizzazione interna di WordNet è differente da quella di un normale dizionario. Mentre quest'ultimo addotta un ordinamento alfabetico e tutti i significati associati ad una parola sono presentati insieme, indipendentemente dalla presenza o meno di legami fra di loro, in WordNet i termini sono organizzati in concetti e non in ordine alfabetico, in base ai risultati dati da ricerche psicolinguistiche. Inoltre esiste una divisione delle parole in categorie sintattiche: si introduce cosi una ridondanza in quanto una parola può avere significati in categorie differenti; d'altra parte si ha però il vantaggio di poter sfruttare le differenze strutturali che esistono nell'organizzazione semantica di tali categorie.

WordNet non può quindi essere considerato un dizionario tradizionale in quanto manca delle principali caratteristiche di questi ultimi, ma di base è un modello innovativo perché sottolinea la differenza fra lemma e significato e offre una rappresentazione delle relazioni tra significati, le quali permettono la navigabilità delle informazioni durante l'utilizzo.

La caratteristica principale di WordNet è l'associazione fra il *lemma* (forma scritta) e il *significato* (il concetto), rappresentato in Figura 9. Esiste un associazione diretta tra una parola e il concetto che essa esprime; è un associazione *molti-a-molti* è dà luogo alle proprietà di:

- *Sinonimia*: proprietà di un significato di avere due o più lemmi in gradi di esprimerlo. Un gruppo di lemmi tra loro sinonimi è chiamato *Synset*;
- *Polisemia*: proprietà di un lemma di poter esprimere uno o più significati.

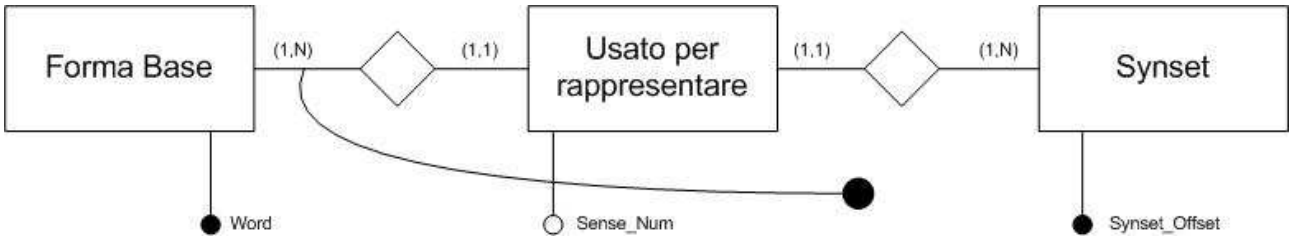

Figura 9 Relazione Lemma-Synset

Tali relazioni possono essere espresse con la *Matrice Lessicale* riportata in Figura 10, nella quale le righe riportano il significato delle parole, rappresentando quindi un *Synset* e le colonne riportano la forma delle parole, rappresentando di conseguenza un *Lemma*. Con questa rappresentazione viene posto subito in evidenza come ad un *lemma* possano essere associati più *synset* cosi come un *synset* possano essere associati diversi *lemmi*. Non solo, viene evidenziato anche come ogni synset possa essere identificato non solo dal suo *identifier* ma anche dalla coppia *lemma* - *sense_number*, dove il *sense_number* è una numerazione effettuata sulla relazione. Ad un *synset* è associata la *glossa*, ovvero una breve frase nella quale è riportata la spiegazione in lingua inglese del significato che esso rappresenta o un semplice esempio di utilizzo qualora la spiegazione non sia soddisfacente o lacunosa per l'identificazione del *synset* stesso.

Word Meanings	Word Forms F_1	F_2	F_3	. . . . .	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
.				. . . .	
.					
.					
M_m					$E_{m,n}$

Figura 10 Matrice Lessicale WordNet

La seconda caratteristica fondamentale di WordNet riguarda il tipo di relazione che può essere rappresentata, qui sotto solamente elencate e per le quali viene data una spiegazione dettagliata nel paragrafo successivo:

- *Relazione Semantica:* coinvolge due *synset* ed è valida per tutti i *lemma* ad essi associati;

- *Relazione Lessicale:* coinvolge due *lemma*, ma non è detto che sia valida per tutti i *lemma* di un determinato *synset*.

3.1.1 Relazioni Semantiche

Sono quelle relazioni che coinvolgono due *synset*:

- **Iponimia – ipernimia**: la relazione di *iponimia* e la sua inversa *ipernimia* sono ciò che la *specializzazione/generalizzazione* sono per un database o l'*ereditarietà* è per un modello ad oggetti. E' valida solo per le categorie sintattiche di nomi e verbi, per i quali si parla però di *toponimia*. Una relazione di *iponimia* lega un *synset* ad uno più generico. Per cui **X** è un *iponimo* di **Y** se risulta valida la relazione "*X is a kind of Y*". Per la relazione opposta, ovvero quella di *ipernimia*, si ha un legame fra un *synset* e uno più specifico. **X** è un *ipernimo* di **Y** se il *synset* X presenta tutte le caratteristiche di Y e in più ha delle caratteristiche "*proprietarie*".
- **Meronimia – olonimia:** anche la relazione di *meronimia* lega fra loro due *synset*. Un *synset* **X** è detto *meronimo* di **Y**, se è lecito dire "*X is a part of Y*". Anche per questa relazione esiste la relazione inversa, l'*olonimia*. Sono valide solo per la categoria sintattica dei nomi.
- **Implicazione:** la relazione di *implicazione* vale per due verbi. Può essere ritenuta simile alla relazione di meronimia sui nomi. Un verbo **X** implica un verbo **Y** se è possibile affermare che X non può verificarsi a meno che non si sia già verificato Y. Non è univoca, per cui se un verbo X implica un verbo Y non può essere vero il contrario.
- **Relazione Causale:** simile alla relazione di implicazione ma senza l'inclusione temporale.
- **Raggruppamento di Verbi:** relazione usata per identificare dei raggruppamenti nei verbi. All'interno di un tale gruppo i *synset* hanno tutti un significato semantico simile.
- **Similarità:** tale relazione viene utilizzata solamente nell'ambito della categoria degli aggettivi. Molti *synset* di questa categoria risultano essere legati tra di loro a coppie attraverso una relazione di *antinomia* (ad esempio **LEGGERO** vs. **PESANTE**); tali *synset* vengono chiamati *head - synset*. A questi *synset* principali sono collegati per similarità dei *synset satelliti*, che condividono indirettamente le relazioni di antinomia insieme al significato principale a cui sono legati.
- **Attributo:** la relazione di *attributo* rappresenta il legame che intercorre fra un aggettivo ed un nome di cui esprime il valore. Gli aggettivi in grado di descrivere il valore di un attributo sono gli aggettivi descrittivi. L'aggettivo *alto* può voler indicare il valore dell'attributo *altezza* riferito ad una persona o ad una cosa. Esiste quindi un legame fra aggettivo e nome e

WordNet contiene puntatori fra gli aggettivi descrittivi e i *synset* (nomi) che rappresentano gli attributi ai quali è riferito il valore.

- **Coordinazione:** non è un tipo di relazione base, ma derivata. Due *synset* sono *coordinati* se possiedono lo stesso *ipernimo*, cioè se risultano essere la specializzazione dello stesso concetto.

3.1.2 Relazioni Lessicali

Diversamente dalle relazioni semantiche questi tipi di relazione coinvolgono sempre due *lemma* e non due *synset*.

- **Sinonimia:** pur rappresentando una relazione lessicale, la sinonimia non è espressa formalmente in WordNet: non esiste alcun puntatore che leghi un *lemma* al suo sinonimo. La relazione è espressa tramite l'appartenenza da parte dei due *lemma* allo stesso *synset*.
    - *Due termini sono sinonimi se la sostituzione di uno per l'altro non cambia mai il valore della frase in cui è fatta la sostituzione* (Leibniz);
    - *Due termini sono sinonimi all'interno di un contesto linguistico **C** se la sostituzione di un termine con l'altro, all'interno di C, non varia il valore della frase* (definizione relativa ad un contesto).

  La seconda definizione risulta essere più permissiva rispetto alla prima: esistono pochi termini considerati sinonimi nel senso descritto dalla prima definizione. Non a caso WordNet adotta come definizione della relazione *sinonimia* la seconda relazione: due *lemma* sono sinonimi solo all'interno dello stesso contesto e di un certo *synset*. Di conseguenza pare ovvio che due termini appartenenti a *synset* differenti non potranno mai essere sinonimi. Questo rappresenta anche il motivo per cui WordNet è stato diviso nelle categorie sintattiche di nomi, avverbi, verbi e aggettivi.
- **Antinomia:** l'*antinomia* è una relazione lessicale tra due *lemma* tale che l'uno sia contrario dell'altro. E' doveroso sottolineare che non sempre è vera la seguente affermazione: *non X è antonimo di X*. Infatti, considerati i termini RICCO e POVERO non è detto che chi non è ricco sia per forza povero
- **Relazione di pertinenza:** la relazione di pertinenza coinvolge gli aggettivi relazionali. Tale aggettivo svolge un ruolo che può essere riassunto secondo l'espressione *associato con*, oppure *pertinente a*. L'aspetto di questo tipo di aggettivi risulta essere molto simile alla forma del nome a cui sono legati. Per esempio l'aggettivo *mentale* è pertinente al nome *mente*.

- **Vedi anche:** la relazione *vedi anche* è una relazione lessicale che lega singoli *lemma* di *synset* differenti.
- **Relazione partecipiale:** lega fra di loro avverbi o aggettivi, detti fra loro partecipiali, rispettivamente ai nomi o ai verbi da cui derivano. Per esempio esiste una relazione partecipiale fra l'aggettivo *bruciato* e il verbo *bruciare*.
- **Derivato da:** riguarda aggettivi che derivano da antichi nomi Greci o Latini. Ad esempio l'aggettivo relazionale *verbale* deriva dal nome latino *verbum*.

3.1.3 Struttura WordNet

La struttura originale di WordNet risulta essere molto diversa dal database relazionale implementato in MOMIS. Viene rilasciato infatti tramite due file:

- **File indice** (*idx*) che contiene su ciascuna delle righe un *lemma* di una determinata categoria e una lista di puntatori, *byte offset*, a tutti i *synset* di cui tale *lemma* fa parte. Inoltre sono riportate altre informazioni come l'ordine che ha il *lemma* in ciascuno dei *synset* associati;
- **File data** (*data*) nel quale ciascuna riga è associata ad un *synset*, contiene l'offset in byte della stessa riga e altre informazioni che riguardano le relazioni lessicali e semantiche che coinvolgono *synset* o *lemma*.

Da questa struttura è possibile estrarre le informazioni riguardanti *lemma*, *synset*, relazioni lessicali e relazioni sintattiche.

3.2 WordNet Domains

In ambito lessicale un *dominio* può essere definito come un'area comune della discussione umana, come politica, sport, chimica. Tali "aree" contribuiscono in maniera rilevante a evidenziare l'esistenza di una coerenza lessicale dei termini di cui si fa uso. Un' importante parte del linguaggio è costituita da termini i cui significati si riferiscono a domini specifici e che spesso compaiono in testi o discussioni che discutono del dominio corrispondente.

Due sono i ruoli che i domini hanno all'interno della descrizione linguistica:

- un ruolo caratterizza il senso dei termini, cosi come il campo semantico di un senso di un termine all'interno di un dizionario. Il termine "*crane*" (gru) ha senso se usato in domini come la zoologia e le costruzioni. Un' ulteriore descrizione è data in [21];
- un secondo ruolo è quello di caratterizzare i testi attraverso un generico livello di categorizzazione del testo;

Relativamente alla disambiguazione dei termini, utilizzata per permettere l'annotazione automatica, un dominio può essere considerato sotto due distinti punti di vista:

- l'informazione di un dominio rappresenta un'informazione sostanziale e fondamentale per la disambiguazione di un termine. Molte delle caratteristiche che contribuiscono al processo di disambiguazione identificano un dominio che caratterizza un particolare senso o un particolare sottoinsieme di sensi. Per esempio i termini economici forniscono aspetti caratterizzanti sensi di termini finanziari (*bank* e *interest*) mentre i termini legali caratterizzano invece i sensi dei termini giuridici(*sentence* e *court*);
- i domini forniscono un ulteriore utile livello di granularità riguardo la distinzione dei *synset*. Molte applicazioni spesso non sfruttano l'alto livello di granularità fornito da strumenti come WordNet e tale lacuna spesso rende oneroso il processo di disambiguazione. Si pensi infatti ad esempio come alcuni verbi in WordNet contengano oltre 40 *synset* differenti.

WordNet Domains è un progetto sviluppato presso il **Centro di Ricerca FBK -irst**[6]. Si tratta di un database che associa ad ogni *synset* di WordNet il dominio relativo. In seguito allo studio del ruolo che i domini hanno nell'ambito della disambiguazione del testo [22] è nata l'ipotesi che i domini forniscano un potente mezzo per stabilire le relazioni semantiche fra i *synset*. Oltre ad associare ad ogni *synset* l'etichetta del dominio WordNet Domains consente anche di stabilire il dominio prevalente di un testo o di una porzione di esso, in modo poi da determinare come questo

[6] http://wndomains.itc.it/wordnetdomains.html

influisca sulla disambiguazione dei testi.

3.3 WordNet in MOMIS

Come descritto nel capitolo 1.3 il sistema MOMIS utilizza come risorsa lessicale il database lessicale **WordNet 2.0**. La disponibilità gratuita e la sua completezza rendono WordNet uno dei database lessicali maggiormente usati e son stati questi i motivi principali che hanno portato alla sua integrazione all'interno del progetto MOMIS. La risorsa lessicale WordNet, fornita sottoforma di file, è stata integrata in MOMIS all'interno di un database relazionale avente la struttura mostrata in figura 11. Si specifica che d'ora in avanti, nella rappresentazione di uno schema relazionale di un database, le frecce indicano le Foreign Key.

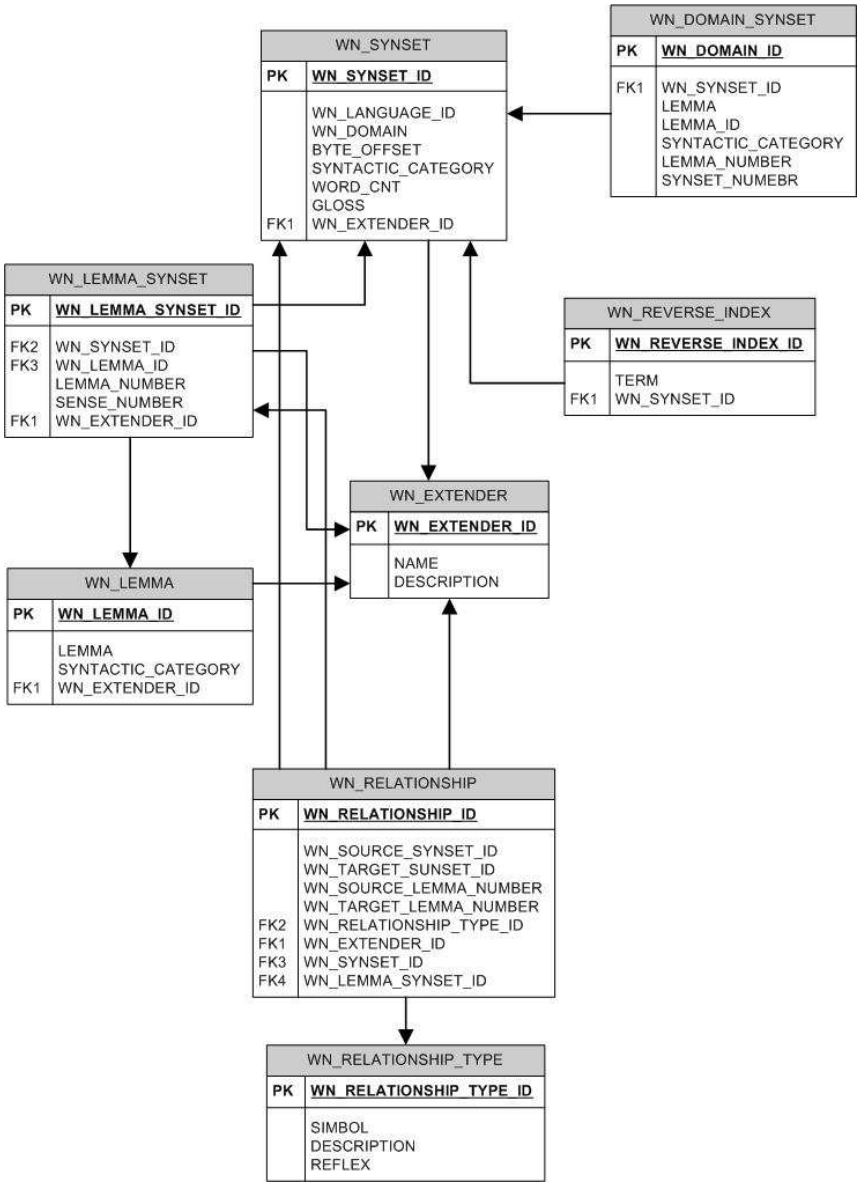

Figura 11 Schema Relazionale MOMISWND

- **WN_SYNSET:** tabella che contiene le *glosse* dei *synset*. L'associazione tra *synset* e *glossa* è data dai campi BYTE_OFFSET e SYNTACTIC_CATEGORY i quali consentono di individuare un synset all'interno della struttura di WordNet;
- **WN_RELATIONSHIP_TYPE:** tabella che contiene i tipi di relazione previsti nella versione 2.0 di WordNet;
- **WN_RELATIONSHIP:** tabella che contiene le relazioni semantiche tra *synset* riportate in WordNet;
- **WN_LEMMA_SYNSET:** tabella nella quale sono riportate le associazioni fra ciascun *lemma* ai *synset*;
- **WN_LEMMA:** tabella che contiene i *lemma* presenti in WordNet indicando la categoria sintattica di appartenenza;
- **WN_EXTENDER:** ogni tabella di WordNet contiene una *Foreign Key* verso l' attributo WN_EXTENDER_ID di questa tabella, mediante il quale è possibile risalire alla versione dei dati presenti su WordNet. Questa tabella infatti contiene un elenco di record che offrono un'identificazione dei dati presenti su WordNet in base alla loro origine;

L'utilizzo di WordNet all'interno di MOMIS consente di determinare le relazioni inter-schema al fine di scoprire le affinità tra classi appartenenti a diverse sorgenti dati e risulta quindi di importanza cruciale nella fase di Annotazione del processi di Integrazione.

Diversi sono i modi in cui l'utente può interfacciarsi con WordNet durante la fase di Annotazione:

- ***Annotazione manuale***: per ogni elemento dello schema l'utente sceglie il significato corretto, considerando il particolare contesto delle fonti che si stanno integrando. L'operazione viene eseguita in due passi:
  - *Scelta del termine*: si cerca il termine corretto all'interno di WordNet. In questa fase si è coadiuvati dal *WordNet Morphologic Processor* che, ove possibile, esegue delle operazioni di *stemming* per riportare un termine alla sua *base form*;
  - *Scelta del significato*: a ciascun termine individuato possono essere associati zero o più significati;
- ***Annotazione automatica***: vengono eseguiti degli algoritmi che cercano autonomamente la *base form* di un termine e vi associano il significato più probabile; verranno maggiormente descritti nei capitoli successivi.
- ***WordNet Editor***: modulo sviluppato da Veronica Guidetti nel 2002 [20]. Fino ad allora l'utente non poteva modificare o integrare a mano il database di WordNet durante la

fase di Annotazione. Tale modulo è presentato all'utente sotto forma di GUI che permette di poter arricchire WordNet di nuovi termini, in particolare, di inserire *synset*, *lemma* e relazioni.

L'implementazione delle funzionalità lessicali all'interno del sistema informativo **MOMIS** è gestita mediante un motore di persistenza, **Torque**, il quale si occupa in maniera semi-automatica di effettuare le operazioni di inserimento o modifica dei record all'interno del database WordNet. Per identificare univocamente ogni record di WordNet è stato creato un apposito *identifier* in ogni tabella, in maniera tale da non dover utilizzare delle chiavi composte date da altri attributi e facilitare cosi il reperimento di un singolo record. In realtà WordNet prevedrebbe un *identifier* solo per i singoli *synset,* il quale racchiude sia le informazioni sulla categoria sintattica, sia sulla posizione all'interno del file originale. Un esempio si ha con il *synset* il quale è identificato in WordNet con la stringa "*n#00002790*", nella quale il primo carattere rappresenta la categoria sintattica, il carattere "#" un generico separatore e il numero "00002790" il *byte_offset,* ovvero la posizione all'interno del file originale. In Tabella 1 sono riportate le diverse rappresentazioni della *syntactic_category* all'interno di WordNet e all'interno di MOMIS.

SYNTACTIC_CATEGORY	WORDNET	MOMIS
Nouns	n	1
Verbs	v	2
Adjectives	a	3
Adverbs	r	4

Tabella 1 Categorie sintattiche in WordNet e MOMIS

L'attributo identifier generato automaticamente da Torque viene rappresentato con un suffisso "*_ID*" nel nome stesso e viene usato sia come **Primary Key** all'interno della singola tabella, sia come **Foreign Key** nelle tabelle che specificano relazioni tra *lemma* o *synset* (ad esempio gli attributi WN_SOURCE_SYNSET_ID e WN_TARGET_SYNSET_ID in **WN_RELATIONSHIP**) .

3.4 Il database lessicale MultiWordNet

Il progetto **MultiWordNet** [23], sviluppato presso l'istituto **FBK-irst di Trento** si propone di creare un database lessicale multilingua, comprendendo fra le varie lingue supportate anche l' Italiano. E' stato quindi definito un **Italian WordNet** strettamente allineato alla versione di **Princeton WordNet 1.6**. Dalla rete, all'indirizzo http://multiWordNet.itc.it, è possibile reperire la documentazione e qualsiasi altra informazione relativa al progetto. Sul sito è presente inoltre un'interfaccia web attraverso la quale è possibile testare il suo funzionamento. Come riporta il sito sino a maggio 2008 il progetto era ancora attivo e le lingue sino ad allora implementate erano **Italiano**, **Rumeno**, **Latino**, **Ebraico**, **Spagnolo**, **Portoghese**. Prendendo in considerazione solo Italian WordNet questo risulta essere allineato alla versione 1.6 di Princeton WordNet e contiene 38654 *synset* e 45089 *lemma*. Il progetto viene distribuito sottoforma di dump di tabelle per un DBMS MySql ed è corredato da un file di ausilio che spiega brevemente quali sono le caratteristiche delle tabelle e che tipologia di dati contengono.

Italian WordNet rappresenta un'implementazione di un modello *expand-model*, in quanto ha come obiettivo quello di creare un database strettamente allineato a Princeton WordNet pur mantenendo un'architettura che sia il più possibile flessibile all'inserimento di diverse lingue. Inoltre offre una soluzione a quei problemi che tale modello può far scaturire, consentendo di specificare laddove necessario le particolarità semantiche tipiche di ogni lingua e dando una rappresentazione dei *lexical gap* qualora ce ne fossero.

3.4.1 Descrizione del modello

Nel capitolo 3.1 è stato descritto il modello WordNet come una *matrice lessicale* nella quale sono riportati significati e termini. Italian WordNet estende questa matrice in una *matrice lessicale multilingua*, rappresentata in Figura 12, aggiungendo una terza dimensione data appunto dalla lingua.

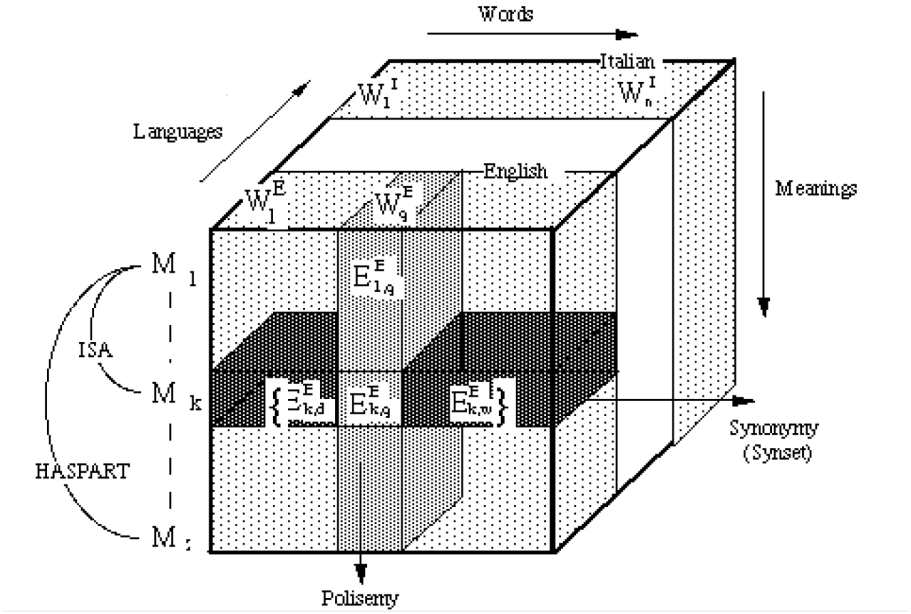

Figura 12 Matrice Lessicale in MultiWordNet

Partendo dall'implementazione del modello di WordNet, la costruzione di Italian WordNet consiste in un *re-mapping* dei lemmi italiani secondo i *synset* della lingua inglese ridefinendo completamente le relazioni lessicali che collegano *lemma* e *synset*. Le relazioni semantiche che invece non sono specifiche per ciascuna lingua ma rappresentano legami fra concetti, per cui il più delle volte comuni, rimangono dove possibile quelle già definite dalla lingua inglese, fatta eccezione per quei significati che nella versione di Princeton WordNet mancano.

Il concetto di *synset* si evolve quindi in *multisynset*, ovvero il significato comune alle due lingue viene rappresentato e individuato univocamente nell'ontologia. Il multisynset quindi non avrà più il semplice scopo di individuare relazioni di sinonimia tra *lemma* di una stessa lingua ma individuerà una relazione di *sinonimia* tra *synset* equivalenti e i relativi *lemma* in lingue diverse. Una relazione semantica viene ad essere estesa in validità a tutte le lingue.

3.4.2 Architettura del modello

La struttura di MultiWordNet si compone di una parte comune a tutti i database lessicali implementati e di una parte specifica per ogni lingua. In Figura 13 viene mostrato uno schema semplificato della struttura di MultiWordNet:

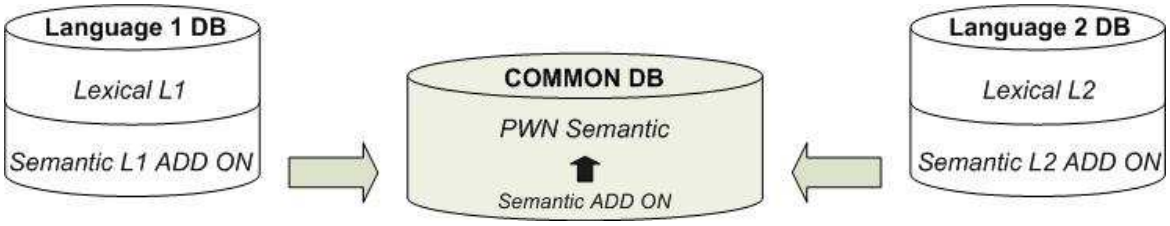

Figura 13 Struttura generalizzata di MultiWordNet

.

Con riferimento al caso in questione, ovvero Italian WordNet, saranno presenti i seguenti moduli:

- *Common-DB*: modulo che contiene tutte le relazioni semantiche presenti in Princeton WordNet. Di conseguenza è possibile affermare che il Common-DB contiene anche tutti i *multisynset*, e non i *lemma*. In realtà all'interno del database i *multisynset* non sono implementati, ma vengono definiti indirettamente, utilizzando lo stesso identificatore del *synset* generico per i diversi modelli linguistici;
- *English-DB* e *Italian-DB*: sono moduli contenenti le relazioni lessicali di una specifica lingua, in questo caso l'Inglese e l'Italiano, i cui *lemma* sono gli stessi riportati in Princeton WordNet;

Tale struttura risulta essere adatta a rappresentare con esattezza le informazioni condivise dalle diverse lingue, tuttavia non permette di specificare le principali differenze strutturali che invece si vorrebbero rappresentare. Per avere la possibilità di poter modificare le relazioni semantiche già esistenti e di aggiungerne di nuove, sono stati inseriti i moduli *add-on*, i quali sovrascrivono sezioni consistenti del database WordNet senza andare ad alterarne la struttura e che hanno come principale obiettivo quello di rappresentare le relazioni semantiche incompatibili con la lingua d' origine:

- *Semantic add-on*: modulo incapsulato nel Common-DB. Permette di modificare l'insieme delle relazioni semantiche comuni;
- *L1 Semantic add-on, L2 Semantic add-on*: moduli incapsulati all'interno della sezione specifica di ciascuna lingua, il cui ruolo consiste nel rappresentare relazioni semantiche incompatibili con una determinata lingua e offrire quindi un maggior dettaglio su eventuali lacune. Il contenuto viene a sua volta diviso in due tipologie:
  - *Relazioni semantiche specifiche di una lingua;*
  - *Lexical Gap;*

Un **Lexical Gap** si presenta quando un determinato concetto, che in una lingua viene associato ad un *synset* non nullo, esprimibile quindi attraverso singoli *lemma*, in una lingua differente non trova una corrispondenza diretta con un *synset*, ma può solo essere tradotto con una frase o con un termine più generico.

Nel primo caso viene riportata la frase espressa nella *glossa* mentre il termine vero e proprio viene individuato dal *lemma* generico "**GAP!**". Nel secondo caso si ha una *denotation difference* che viene risolta mediante l'inserimento di un nuovo tipo di relazione fra *synset* che mira a collegare il gap con il *synset* generico più vicino: ***nearest***.

3.4.3 Schema del database MultiWordNet

MultiWordNet è disponibile come file dump di un DBMS MySql sul sito ufficiale del progetto. Dal sito sono scaricabili gratuitamente anche una serie di documenti informativi sulla struttura del database stesso.

Una volta scaricato il dump di MultiWordNet è possibile ripristinarlo attraverso un qualsiasi client MySql a disposizione o attraverso linea di comando, ottenendo quindi lo schema mostrato in Figura 14.

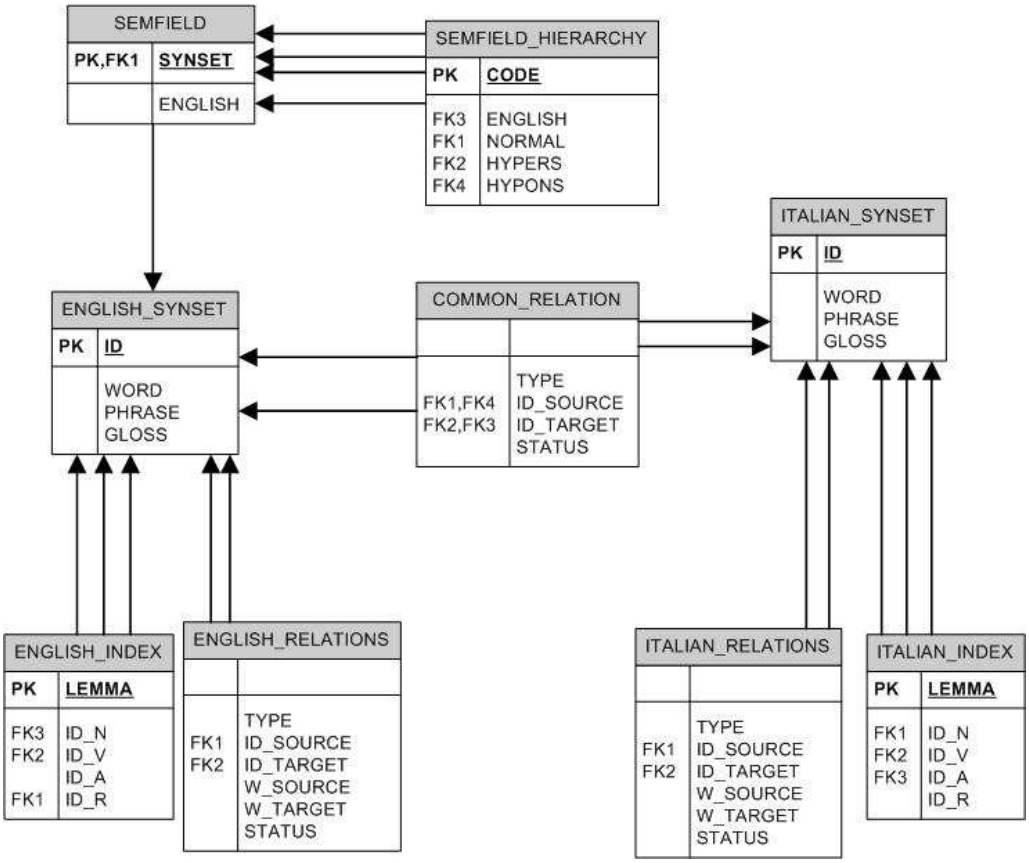

Figura 14 Schema Relazionale di MultiWordNet

- **COMMON_RELATION:** tabella contenente tutte le informazioni relative a relazioni semantiche fra *synset*,e per questo, comuni a tutte le lingue. Vengono riportati il tipo di relazione e i due *synset* coinvolti,
- **ENGLISH_RELATION / ITALIAN_RELATION:** tabelle contenenti le relazioni lessicali specifiche di ogni lingua;
- **ENGLISH_SYNSET / ITALIAN_SYNSET:** tabelle contenenti le informazioni riguardanti i *synset* inglesi o italiani. L'identificativo della tabella, **ID**, è una stringa formattata come

segue: "*n#00002790*". La tabella memorizza in un attributo, **WORD,** la lista dei *lemma* associati al *synset*, la *glossa* del *synset*, **GLOSS**, e talvolta una frase di esempio, **PHRASE**, utilizzata al posto della *glossa* o del *synset* stesso in caso di *Lexical Gap*.

- **ENGLISH_INDEX / ITALIAN_INDEX**: tabelle contenenti i *lemma* delle diverse lingue. Oltre alla Primary Key, data dall' attributo **LEMMA**, riportano memorizzati in attributi differenti ma simili per proprietà, ciascuno relativo ad una propria categoria sintattica, i *synset* con cui i *lemma* sono in relazione.

Ai fini della presente tesi non interessa l'analisi delle tabelle **SEMFIELD** e **SEMFIELD_HIERARCHY**, in quanto sono tabelle di contorno della lingua Inglese, utilizzate per la definizione dei verbi composti.

3.5 Il database lessicale EuroWordNet

EuroWordNet [18] è un database multilingua costituito da WordNet di diverse lingue (Tedesco, Italiano, Spagnolo, Olandese, Francese, Ceco ed Estone), strutturati tutti nella stessa maniera di Princeton WordNet, in termini di *synset* con inter – relazioni semantiche di base. Il progetto è iniziato con il finanziamento della Comunità Europea nel 1996, si è concluso nel 1999 ed ha visto la partecipazione di molti degli stati membri. In Italia è stato l'**Istituto di Linguistica Computazionale "Antonio Zampolli"** di Pisa ad occuparsene. Attraverso il sito ufficiale di riferimento è ancora possibile scaricare una versione dimostrativa del database e del browser Periscope. In particolare in questo paragrafo verrà analizzato brevemente la versione Italiana di EuroWordNet, *ItalWordNet.*

Ciascun WordNet rappresenta un unico sistema di lessicalizzazione del linguaggio e sono collegati tra di loro mediante l' *Inter Lingual Index*, modulo che verrà descritto in seguito.

Il progetto si proponeva di creare un database lessicale multilingua che fosse consistente e affidabile, e che non perdesse le peculiarità delle diverse lingue. L'implementazione è stata fatta sulla base di un modello *merge-model*, discusso nei paragrafi precedenti.

3.5.1 Descrizione

Il progetto prevedeva come base di partenza lo sviluppo di database lessicali WordNet – compatibili, prendendo i dati da risorse preesistenti. La creazione dei diversi *wordnet* nelle diverse lingue era vincolata dalla versione di Princeton WordNet disponibile in quel momento, ovvero la 1.5. I *synset* dei *wordnet* associati mediante una relazione di equivalenza a quelli di Princeton Wordnet e in seguito si provvedeva a creare un modulo centrale che raccogliesse i *synset* e i *lemma* più importanti delle diverse lingue in modo da garantire un alto livello di consistenza e di poter mantenere invece nei moduli riguardanti le singole lingue le diverse caratteristiche. In figura 14, presente nel sito ufficiale del progetto[7], viene mostrato uno schema a blocchi che descrive il processo di creazione di un WordNet.

Tale processo prevede il reperimento da risorse già esistenti, quali *wordnet* obsoleti, dizionari elettronici e altro, di un sottoinsieme di significati, per i quali vengono estratte tutte le parole che le definiscono. Si riesce ad ottenere quindi un insieme di *synset* ed un insieme di *lemma*. In seguito

[7] http://www.illc.uva.nl/EuroWordNet/

vengono rilevate all'interno del sottoinsieme di significati le relazioni particolari della lingua, *language internal relations*. E' cosi possibile individuare una *equivalence relation* fra i *synset* di una lingua ed i *synset* di Princeton WordNet 1.5. I dati vengono resi maggiormente consistenti in seguito ad un ulteriore controllo interlinguistico, ristrutturando qualora vi fosse il bisogno le gerarchie dei *synset* e le loro relazioni. Il processo viene ripetuto per le singole lingue oggetto dell'integrazione ed infine si definisce una *top – ontology* comune, recuperando dai singoli *wordnet*, i *synset* e le relazioni maggiormente coinvolte.

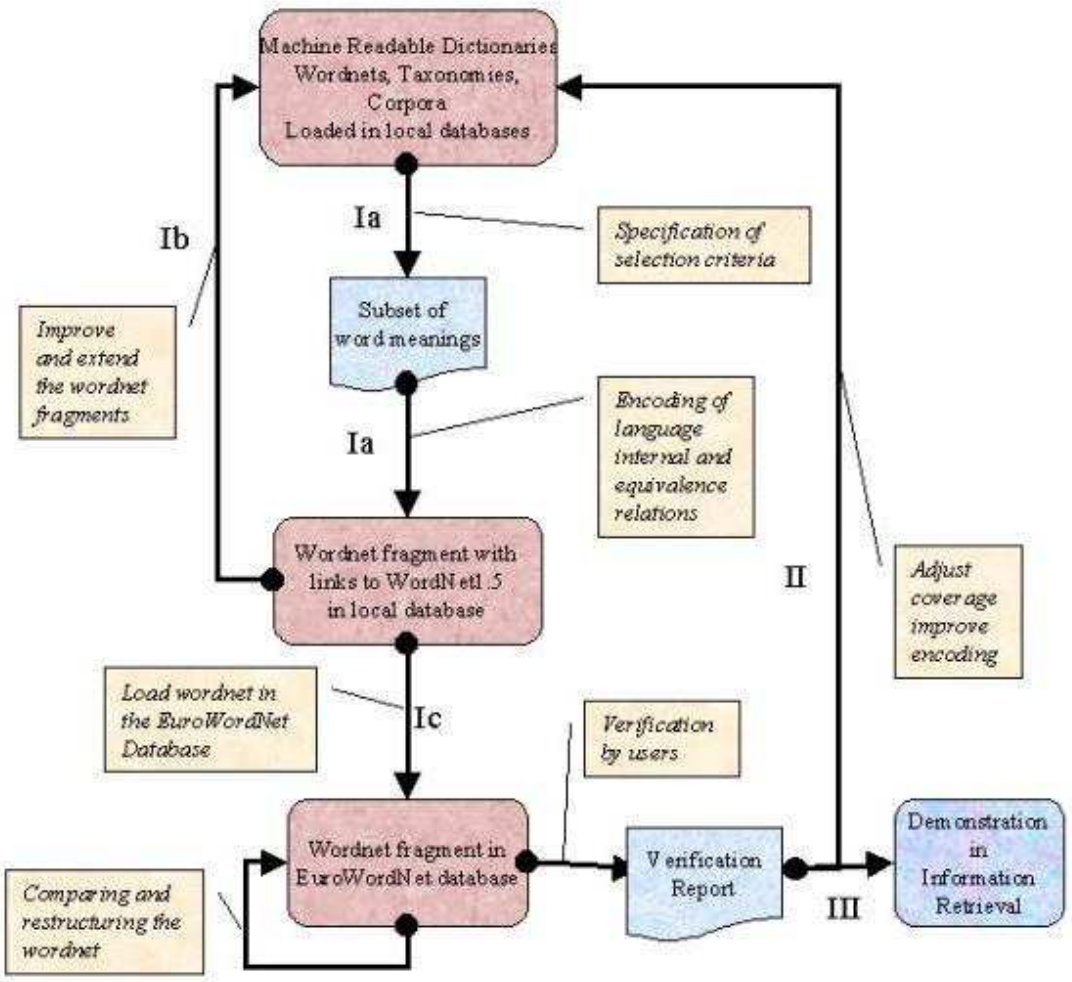

Figura 15 Schema a blocchi del processo di creazione di un singolo wordnet

3.5.2 Architettura

Come descritto in precedenza il database EuroWordNet presenta due caratteristiche fondamentali: un' alta consistenza dei record lessicali e una particolare attenzione alla ricchezza di

ciascuna lingua implementata. Queste due caratteristiche sono individuate internamente da due moduli: uno costituito da un insieme di diversi *Language Model*, ciascuno rappresentato da un singolo *wordnet*, e uno costituito dal *Language Indipendent Model*, nel quale è contenuta la *top – ontology*, elemento che funge da ponte fra le varie risorse linguistiche presenti. In figura 16, nella quale è riportato lo schema della struttura di EuroWordNet, è possibile individuare questi due moduli.

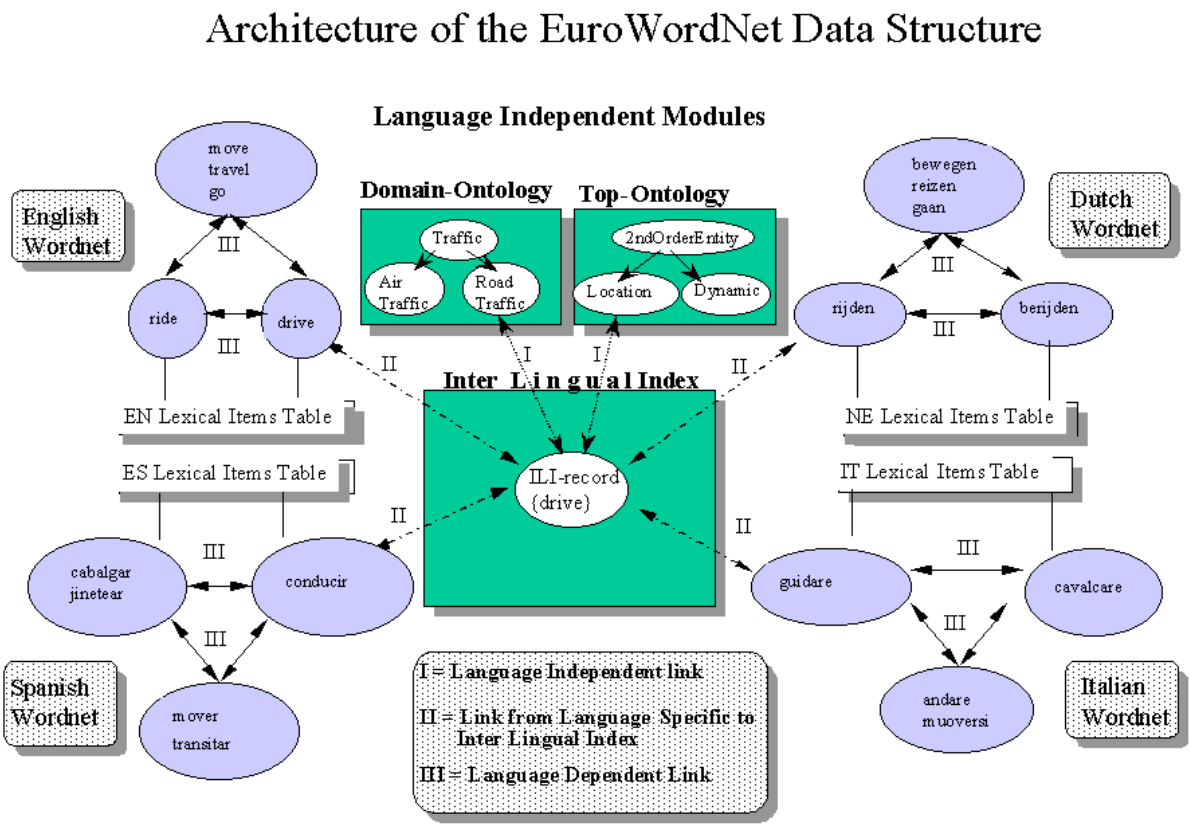

Figura 16 Schema della struttura di EuroWordNet

Ogni Language Model contiene due tipi di dato, i ***Word Meaning*** record (**WM**) e i ***Word Instance*** Record (**I**). Corrispondono ambedue ai *synset* di WordNet con la differenza che i WM ammettono qualsiasi tipi di relazione, mentre gli I si occupano di gestire le relazioni di una sola categoria sintattica, propria di EuroWordNet: *Proper Nouns*. Inoltre all'interno di questo modulo è contenuto l'insieme dei *lemma* della lingua a cui si fa riferimento. Questi record vengono coinvolti principalmente in due tipi di relazioni, *Language Internal Relation* che interessa due record appartenenti alla stessa lingua, e *Interlingual Relation*, la quale interessa a sua volta i record di un

Language Model con quelli del modulo ILI (*Inter Lingual Index*), descritto in seguito.

Il Language Indipendent Model si suddivide a sua volta in 3 sottomoduli:

- **Inter Lingual Index (ILI).** Contiene gli **ILI-Record**, i quali vengono coinvolti nelle *Interlingual Relation* e nelle *Language Indipendent Relation*, che interessano i *synset* della top – ontology ;
- **Domain Ontology.** Contiene i **Domain Record** (**DR**), i quali vengono coinvolti in due tipologie di relazioni: *Module Internal Relation* e *Module External Relation*, la cui differenza consiste nel coinvolgere due DR o un DR e un ILI – Record. Entrambe le relazioni vengono raggruppate nelle *Language Indipendent Relation*;
- **Top - Concept Ontology.** Contiene i **Top – Concept Record** (**TC**), anch'essi coinvolti in due tipi di relazioni uguali a quelle della Domain Ontology.

La versione italiana di EuroWordNet, **ItalWordNet**, utilizzata in questo elaborato viene fornita sottoforma di file .xml all'interno del quale sono definiti i record tramite il tag principale **WORD – MEANING**. Si riporta un breve estratto di tale file.

```
<WORD_MEANING ID="N#82" PART_OF_SPEECH="N">
        <GLOSS>prima lettera dell'alfabeto</GLOSS>
         <VARIANTS>
                <LITERAL LEMMA="a" SENSE="1" STATUS="CT"/>
        </VARIANTS>
        <INTERNAL_LINKS>
        <RELATION TYPE="has_hyperonym" ID="8" INV_ID="8">
                <TARGET_WM ID="338" PART_OF_SPEECH="N" LEMMA="carattere" SENSE="4" GLOSS="ciascuno
degli elementi di un alfabeto  dell'alfabeto."/>
        </RELATION>
        </INTERNAL_LINKS>
        <EQ_LINKS>
                <RELATION TYPE="eq_has_hyperonym" ID="3" INV_ID="3">
                <TARGET_WM ID="N#4451043#letter, letter_of_the_alphabet, alphabetic_character"/>
                </RELATION>
        </EQ_LINKS>
         <TOP_ONTO>
                <RELATION CONCEPT=" LanguageRepresentation"/>
                <RELATION CONCEPT=" Artifact"/>
        </TOP_ONTO>
</WORD_MEANING>
```

Il tag si occupa innanzitutto di definire l'*id* e la *gloss* del synset. In seguito vengono mappate le diverse relazioni e raggruppate mediante la loro tipologia. Si nota come per le relazioni di

equivalenza si ha come *id* del tag simile all'*identifier* descritto nel capitolo 3.1 con l'aggiunta . dei vari concetti.

In base agli studi condotti su questo progetto e a quelli riportati in [19] non si è proseguito oltre con lo studio di EuroWordNet per una possibile integrazione in MOMIS. Diverse le motivazioni che hanno portato a questa scelta. Innanzitutto l'approccio *expand* model, usato da MultiWordNet, consente una più rapida integrazione rispetto al *merge model* di EuroWordNet, grazie al fatto che la struttura di MultiWordNet è la stessa di WordNet. Non solo, lo stesso formato in cui sono state fornite le due risorse lessicali è stato un fattore discriminante nel processo di integrazione. Avendo a diposizione un database relazionale e un file .xml si è preferito privilegiare il primo in quanto è risultato più semplice e veloce analizzare e processare un database piuttosto che un file semi – strutturato. Ciò non esclude che futuri sviluppi potranno riguardare anche l'integrazione fra WordNet ed EuroWordNet..

3.6 Integrazione di MultiWordNet all'interno di MOMIS

Questo capitolo tratta le modifiche apportate al sistema MOMIS per dotarlo della funzionalità di annotazione multilingua. Nel caso specifico, essendo l'obiettivo l'integrazione di sorgenti in lingua italiana e inglese, si è integrato in MOMIS un database lessicale provvisto di queste due lingue. La soluzione ha visto l'implementazione di un dizionario lessicale multilingua composto da WordNet 2.0 esteso con l'ontologia lessicale MultiWordNet italiano,e la gestione delle diverse funzionalità durante il processo di Annotazione.

Com'è noto MOMIS è in grado di effettuare un'Annotazione sulla sorgente dati fornita, per costruire un proprio Common Thesaurus e proseguire con il Processo di Integrazione.

La versione di WordNet presente in MOMIS, chiamata MOMISWND, è stata creata sulla base di WordNet 2.0 e di WordNet Domains, la versione di MultiWordNet utilizzata in questa tesi risulta essere allineata a WordNet 1.6. E' stato quindi necessario procedere con un aggiornamento della versione di MultiWordNet per allinearla alla versione 2.0 di WordNet ed avere così dati confrontabili con quelli attualmente memorizzati in MOMISWND.

In seguito a tale aggiornamento è stata realizzata l'integrazione vera e propria di MultiWordNet in MOMISWND seguita da alcune importanti modifiche su parte del sistema informativo MOMIS per la gestione delle nuove proprietà del dizionario lessicale all'interno delle diverse fasi del processo di integrazione.

3.6.1 Allineamento di MultiWordNet a WordNet 2.0

La versione di MultiWordNet a disposizione si basa su una versione di WordNet ormai desueta, la versione 1.6. Nel paragrafo 3.4.2 si è focalizzata l' attenzione sui cosiddetti *Lexical Gap* e sugli svantaggi che possono portare. Se da un lato le lacune sui *synset* possono essere considerate come un problema trascurabile, o comunque di relativa importanza, in una risorsa ampiamente estendibile come MultiWordNet, non si può trascurare il fatto che la versione 1.6 di WordNet differisce da quella utilizzata attualmente in MOMIS non solo per la quantità di *synset* presenti all'interno di esse, maggiore ovviamente in WordNet 2.0 in quanto più recente, ma anche alcuni *synset* già presenti, per i quali si ha una alterazione dei dati nel passaggio da una versione all'altra. Questa alterazione ha riguardato in particolar modo l'identificativo usato da WordNet per la definizione univoca del *synset* e comporta un diverso allineamento dei dati delle due versioni.

Questo particolare problema non ha permesso la semplice copia dei dati di MultiWordNet in

MOMISWND come risoluzione al problema dell'integrazione, in quanto *synset* identici avrebbero avuto identificativo diverso, e ciò avrebbe portato notevoli problemi nelle fasi di Annotazione, rendendo quest'ultima errata e di conseguenza inutile ai fini del Processo di Integrazione. E' stato obbligatorio prima un aggiornamento del dizionario MultiWordNet per renderlo coerente con la versione di MOMISWND.

Tale allineamento è stato eseguito sulla base degli studi[8] condotti dalla professoressa **Rada Mihalcea** [24], del **Dept. Of Computer Science and Engineering, University of North Texas.** All'interno della sua pagina personale dell'Università è possibile scaricare un file di testo, il quale evidenzia le differenze tra gli identificativi dei *synset* della versione WordNet 1.6 e quelli della versione WordNet 2.0, suddivisi in base alla categoria sintattica.

Con tale file a disposizione si è potuto procedere ad un allineamento che preservasse i dati originali di MultiWordNet e che consentisse allo stesso tempo di poter disporre degli stessi dati presenti in MOMISWND. Come si vedrà nel paragrafo successivo la procedura di allineamento ha comportato una piccola modifica alla struttura del database di MultiWordNet.

Step 1: Modifica dello schema MultiWordNet

Il primo step ha comportato quindi la modifica delle tabelle nelle quali fosse presente l'identificativo di uno o più *synset*. Si è semplicemente trattato dell'aggiunta di uno o più attributi atti a memorizzare i nuovi identificativi riportati nel file di allineamento. In Figura 17 tali attributi sono riconoscibili dal prefisso "*MWN20_ID_[nome]*".

In questa maniera si è cercato di porre rimedio a due possibili problemi che sarebbero sorti con la modifica diretta dell'identificativo stesso dei record:

- mancata conservazione delle relazioni fra *synset* e *synset* e fra *lemma* e *synset* durante la procedura di allineamento, per la quale una la soluzione alternativa sarebbe stata mettere l'intera operazione di allineamento dei record sotto transazione;
- perdita definitiva degli identificativi originali di MultiWordNet.

[8] http://www.cse.unt.edu/~rada/downloads.html

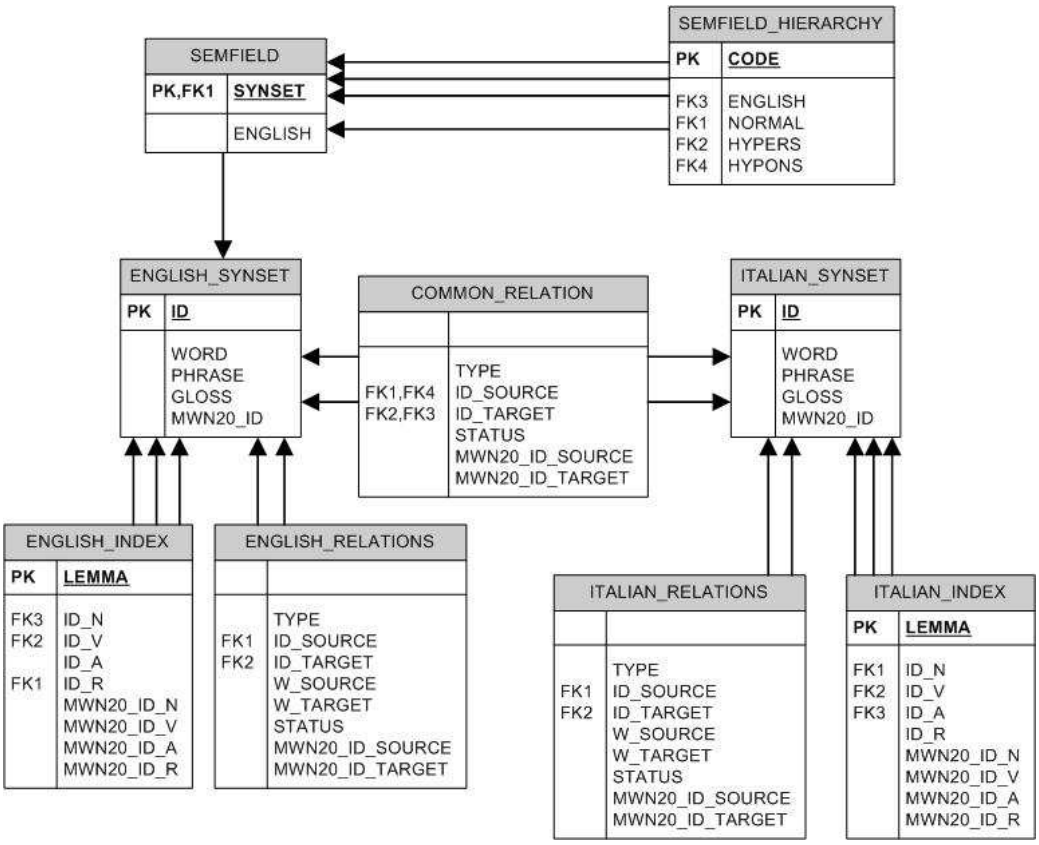

Figura 17 Schema Relazionale di MultiWordNet modificato

Step 2: Procedura di allineamento

In seguito alle modifiche sulla schema di MultiWordNet la procedura di allineamento ha comportato la creazione della classe Java **`UpgradeMultiWordNetIt`** all'interno del package **`it.unimo.dbgroup.momis.annotationOntology.wnManagerFiles.`** All'interno di questa classe sono stati implementati i differenti metodi per la gestione della logica di allineamento. L'esecuzione di questa classe, a mano o da un ambiente IDE, prevede l'uso di almeno un parametro che stabilisce se la procedura dovrà essere eseguita su una particolare tabella o sull'intero schema. Questo parametro è stato introdotto per eliminare i problemi derivanti dall'onerosità del calcolo di allineamento, data dall'elevato numero di record processati.

Senza voler entrare troppo nel dettaglio si rende necessario dare una semplice spiegazione degli algoritmi implementati per l'allineamento e di conseguenza dei metodi creati:

- **`upgradeEnglishSynset()`** e **`upgradeItalianSynset().`** Metodi che si occupano di aggiornare le tabelle ENGLISH_SYNSET e ITALIAN_SYNSET.

    - *Lettura del file "Wn16ToWn20.TXT";*
    - *Per ciascuna riga istanziazione dell'oggetto* LineRecord;

        - *Se ID_v_1.6 != ID_v_2.0*
            - *Salvataggio di ID_v_2.0 in* ENGLISH_SYNSET.MWN20_ID;
        - *Altrimenti*
            - *Salvataggio di ID_v_1.6 in* ENGLISH_SYNSET.MWN20_ID;

- **upgradeIndex(String indexTable, SyntacticCategory syntacticCategory)**. Metodo che esegue l'allineamento delle tabelle ENGLISH_INDEX e ITALIAN_INDEX. Nella firma del metodo vengono passati un indice indicante la tabella da aggiornare e la categoria sintattica da prendere in considerazione in fase di allineamento.

    - *Lettura del file "Wn16ToWn20.TXT";*
    - *Per ciascuna riga istanziazione dell'oggetto* LineRecord;
        - *Selezione da* tableIndex *dei lemmi con categoria sintattica* = syntacticCategory;
        - *Per ciascun lemma recuperato:*
            - *Se ID_v_1.6 != ID_v_2.0*
                - *Sostituzione di ID_v_1.6 con ID_v_2.0 e salvataggio in* [tableIndex].MWN20_id_[syntacticCategory];
            - *Altrimenti*
                - *Salvataggio di ID_v_1.6 in* [tableIndex].MWN20_id_[syntacticCategory];

- **upgradeEnglishRelation(), upgradeItalianRelation()** e **upgradeCommonRelation().** Metodi che eseguono l'allineamento delle tabelle COMMON_RELATION, ENGLISH_RELATION e ITALIAN_RELATION.

    - *Lettura del file "Wn16ToWn20.TXT";*
    - *Per ciascuna riga istanziazione dell'oggetto* LineRecord;
        - *Se ID_v_1.6 != ID_v_2.0*
            - *Salvataggio di ID_v_2.0 nella colonna del Synset Source o del Synset Target;*
        - *Altrimenti*

- *Salvataggio di ID_v_2.0 nella colonna del Synset Source o del Synset Target;*

E’ possibile notare come quasi tutti i metodi implementati in queste procedure presentino sempre l’istanziazione di un oggetto **`LineRecord;`** `LineRecord` è stato creato per la memorizzazione dei dati di una singola riga del file utilizzato per l’allineamento (e di eventuali dati ad essi correlati), e per la verifica degli identificatori delle due versioni (che stabilirà poi quale logica seguire per l’allineamento del singolo record).

3.7 Integrazione di MultiWordNet in WordNet

La procedura di allineamento effettuata ha consentito di avere una versione di MultiWordNet coerente con MOMISWND, e questa è stata la base da cui partire per la realizzazione dell'integrazione fra i due diversi database lessicali.

Nell'integrazione successiva è stata effettuata la scelta di non modificare in alcuna maniera la struttura di MOMISWND, in modo da non creare svantaggi nell'utilizzo del motore di persistenza Torque durante la fase di interrogazione del dizionario stesso, e di non agire in maniera invasiva sull codice sorgente di MOMIS, frutto di diversi anni di sviluppo.

Si è scelto di operare sulla falsa riga dell'allineamento effettuato in precedenza, ovvero la creazione di una classe Java, **`MultiWordNetLoader`**, all'interno del package **`it.unimo.dbgroup.momis.annotationOntology.wnManagerFiles`**. Anche per questa classe è stato ipotizzato l'utilizzo di alcuni parametri per provvedere all'integrazione delle singole entità del dizionario MultiWordNet piuttosto che dell'intero schema. L'operazione infatti risulta avere un costo computazionale maggiore della precedente in quanto si rende necessario realizzare alcune *query* di ricerca sullo schema MultiWordNet, delle opportune logiche di confronto e infine aggiornare dello schema di MOMISWND.

Step 1: Integrazione WN_RELATIONSHIP_TYPE e WN_EXTENDER.

Il primo passo ha comportato l'integrazione delle tabelle WN_RELATIONSHIP_TYPE e WN_EXTENDER, dato che queste tabelle non necessitavano di un'analisi dei record lessicali. Si è trattato di aggiornare la tabella WN_RELATIONSHIP_TYPE con i nuovi tipi relazione codificati all'interno di MultiWordNet. Rispetto alla versione di MOMISWND sono presenti alcuni tipi di relazione aggiuntivi, di cui si è data breve spiegazione nel capitolo 3.3.2. La tabella 2 riporta i tipi di relazione usati dallo schema integrato.

Tabella 2 Tipi di relazione MultiWordNet - MOMISWN

WN_RELATIONSHIP_TYPE_ID	SYMBOL	DESCRIPTION
1	!	Antonym (nouns,verbs,adjectives,adverbs)
2	@	Hypernym (nouns,verbs)

3	~	Hyponym (nouns,verbs)
4	#m	Member meronym (nouns)
5	#s	Substance meronym (nouns)
6	#p	Part meronym (nouns)
7	%m	Member holonym (nouns)
8	%s	Substance holonym (nouns)
9	%p	Part holonym (nouns)
10	=	Attribute (nouns,adjectives)
11	+	Derivationally related form (nouns,verbs)
12	;c	Domain of synset - TOPIC (nouns,adjectives,verbs,adverbs)
13	-c	Member of this domain - TOPIC (nouns,adjectives)
14	;r	Domain of synset - REGION (nouns,adjectives,verbs,adverbs)
15	-r	Member of this domain - REGION (nouns,adjectives)
16	;u	Domain of synset - USAGE (nouns,adjectives,verbs,adverbs)
17	-u	Member of this domain - USAGE (nouns,adjectives)
18	*	Entailment (verbs)
19	>	Cause (verbs)
20	^	Also see (verbs,adjectives)
21	$	Verb Group (verbs)
22	&	Similar to (adjectives)
23	<	Participle of verb (adjectives)
24	\\	Pertainym(pertains to noun,adjectives) Derived from adjective(adverbs)
25	;d	Domain of synset-WND (nouns,adjectives,verbs,adverbs)
26	-d	Member of this domain-WND (nouns,adjectives,verbs,adverbs)
27	d>	Hypernym of domain-WND (nouns)
28	d<	Hyponym of domain-WND (nouns)
29	~i	Instance Hyponym (nouns)
30	|	Synset nearest to *(nouns,verbs,adjectives,adverbs)
31	+;c	Composed-of(is composed of *) (nouns,verbs,adjectives,adverbs)
32	-;c	Composes(compose of *) (nouns,verbs,adjectives,adverbs)

Anche per la tabella WN_EXTENDER l'integrazione ha comportato l'aggiunta diretta di alcuni dati, con l'unica differenza che i record aggiuntivi non derivano da dati contenuti all'interno del database lessicale MultiWordNet ma piuttosto da scelte di progetto intraprese in fase di sviluppo. Dall'analisi condotta nel capitolo 3.1.3 è emerso che la tabella WN_EXTENDER contiene i dati relativi all'origine di un record all'interno di MOMISWND. Viene definita mediante l'uso di un

attributo identificativo che funge da chiave primaria, **WN_EXTENDER_ID**, un codice ragionato in formato String , **NAME** e una descrizione esaustiva della fonte di provenienza, **DESCRIPTION**.

Nel caso analizzato, dovendo integrare un database multilingua inglese-italiano è stato scelto di aggiungere tre nuovi record, uno relativo alla parte italiana di MultiWordNet,uno relativo ai nuovi record aggiunti in lingua italiana e l'ultimo relativo alla parte inglese di MultiWordNet. Si vedrà in seguito che quest'ultimo viene utilizzato in pochi casi in quanto definisce un tipo di origine che rimane coerente con la lingua di partenza.

In tabella 3 sono elencati gli *extender* utilizzati dallo schema integrato.

Tabella 3 WN_EXTENDER

WN_EXTENDER_ID	NAME	DESCRIPTION
1	wn	WordNet
2	new	Generic Extender
3	wnd	WordNet Domain
4	mwnit	MultiWordNet_IT
5	mwnen	MultiWordNet_EN
6	mwnitnew	MultiWordNet_IT_N

Step 2: Integrazione WN_SYNSET, WN_LEMMA, WN_RELATIONSHIP, WN_LEMMA_SYNSET

L'integrazione delle tabelle WN_RELATIONSHIP_TYPE e WN_EXTENDER è stata svolta come primo step non solo per l'immediatezza dell'operazione, ma anche perché costituiva uno sviluppo necessario per poter integrare le restanti tabelle di MultiWordNet, ovvero quelle riguardanti *synset*, *lemma* e relazioni. Come conseguenza diretta di questa prima integrazione si ha la possibilità di individuare all'interno di MOMISWND i nuovi tipi di relazione dati da MultiWordNet, di identificare correttamente la lingua di un particolare record e di poter infine utilizzare questi due dati nella logica di integrazione successiva.

Come fatto per la procedura di allineamento vengono brevemente descritti gli algoritmi utilizzati per l'integrazione:

- **`loadWnSynsetItalian()`** e **`loadWnSynsetEnglish()`**: metodi che si occupano si integrare i *synset* di MultiWordNet recuperandoli direttamente dalle tabelle ENGLISH_SYNSET e ITALIAN_SYNSET.

- *Scansione delle tabelle [ENGLISH_SYNSET,ITALIAN_SYNSET];*
- *Per ciascun record trovato:*
  - *Analisi dell'attributo MWN20_ID attreverso* byte_offset *e* syntactic_category*;*
  - *Verifica dell'esistenza del synset in WN_SYNSET;*
  - *Se non esistente:*
    - *Salvataggio del nuovo synset. L'attributo WN_EXTENDER_ID viene recuperato coerentemente con la tabella di partenza;*
    - *Se la glossa è presente:*
      - *Salvataggio del synset con la glossa;*
    - *Altrimenti:*
      - *Salvataggio del synset senza la glossa;*
    - *Uscita;*
  - *Altrimenti:*
    - *Synset già presente: uscita;*

- **loadWnRel(),loadWnRelItalian()** e **loadWnRelEnglish():** Metodi che si occupano dell'integrazione delle relazioni sintattiche presenti in MultiWordNet all'interno delle tabelle COMMON_RELATION, ITALIAN_RELATION e ENGLISH_RELATION.

  - *Scansione delle tabelle [COMMON_RELATION, ITALIAN_RELATION, ENGLISH_RELATION];*
  - *Per ciascun record trovato:*
    - *Analisi degli attributi MWN20_ID_SOURCE e MWN20_ID_TARGET*
    - *Analisi dell'attributo TYPE;*
    - *Verifica dell'esistenza dei synset_Source e synset_Target;*
    - *Se entrambi esistenti e se esiste il tipo di Relazione (TYPE):*
      - *Verifica dell'esistenza della relazione all'interno della tabella WN_RELATIONSHIP;*
      - *Se la relazione non è esistente:*
        - *Salvataggio della relazione all'interno della tabella WN_RELATIONSHIP;*
      - *Altrimenti:*
        - *Relazione già esistente: uscita;*
    - *Altrimenti:*
      - *Corrispondenze non trovate: uscita;*

- **`loadWnLemmaEnglish()`** e **`loadWnLemmaItalian()`:** Metodi che si occupano dell'integrazione dei lemmi presenti in MultiWordNet all'interno delle tabelle ENGLISH_INDEX e ITALIAN_INDEX.

    - *Scansione delle tabelle [ENGLISH_INDEX, ITALIAN_INDEX];*
    - *Per ciascun record trovato:*
        - *Analisi dell'attributo LEMMA;*
        - *Analisi degli attributi MWN20_ID_[SYNTACTIC_CATEGORY];*
        - *Se lemma non presente:*
            - *Salvataggio del lemma all'interno della tabella WN_LEMMA;*
        - *Altrimenti:*
            - *Lemma già presente: uscita;*

- **`loadWnLemmaSynsetItalian()`** e **`loadWnLemmaSynsetEnglish()`:** Metodi che si occupano dell'integrazione delle relazioni lessicali presenti in MultiWordNet.
    - *Scansione delle tabelle [ENGLISH_SYNSET,ITALIAN_SYNSET];*
    - *Analisi dell'attributo MWN20_ID;*
    - *Analisi dell'attributo WORD;*
    - *Analisi dell'attributo PHRASE;*
        - *Verifica esistenza e parsing dell'attributo WORD;*
        - *Costruzione* `HashMap<String idSynset, List<String> lemmasList>;`
        - *Se* `HashMap` *popolata:*
            - *Per ciascuna* `key`:
                - *Analisi della lista dei lemmi;*
                - *Per ciascun lemma:*
                    - *Parsing Lemma;*
                    - *Scansione delle tabelle [ENGLISH_INDEX,ITALIAN_INDEX];*
                    - *Analisi e recupero del synset e del Lemma relativi all'interno di MOMISWND;*
                    - *Se entrambi esistenti:*
                        - *Se la relazione lessicale non esistente:*

- *Salvataggio della relazione all'interno della tabella WN_LEMMA_SYNSET;*

- *Altrimenti:*
  - *Relazione già presente: uscita dal ciclo;*

- *Altrimenti:*
  - *Corrispondenze non trovate in MOMISWND: uscita dal ciclo;*

- *Altrimenti:*
  - *Nessun lemma associato a* idSynset: *uscita dal ciclo;*

3.8 Modifiche al Data Integration System MOMIS

Per concludere con la realizzazione dell'Integrazione del database lessicale MultiWordNet all'interno del sistema MOMIS si sono rese necessarie alcune modifiche al codice stesso del Sistema Informativo per una corretta gestione delle procedure di Annotazione per un corretto funzionamento del processo di integrazione e per offrire all'utente una serie di funzionalità aggiuntive basate sul nuovo database lessicale. Questo capitolo conclude quindi l'analisi di questa implementazione e propone una descrizione delle principali modifiche apportate senza entrare nel dettaglio.

3.8.1 Modifiche al file siDesigner.conf

Nel progetto MOMIS è presente il file di configurazione *siDesigner.conf*, nel quale sono memorizzate alcune proprietà di configurazione del Sistema Informativo, quali il mapping dei Wrapper presenti, il mapping degli algoritmi di Annotazione Automatica o ancora, i parametri di connessione al database lessicale mediante il motore di persistenza **Torque**.

Torque è un *motore di persistenza*, o *persistence layer*, dell' **Apache Software Fundation Database Project**[9], che permette all'utente di accedere e manipolare dati in database relazionali attraverso oggetti Java. In poche parole crea una collezione di classi Java, denominate **Peer,** attraverso un file XML di configurazione, che risulta essere la mappatura esatta delle entità del database relazionale gestito dal motore. Le relative operazioni sul database (SELECT, CREATE, INSERT, UPDATE, DELETE) vengono quindi gestite automaticamente da questa tecnologia mediante classi e metodi auto – generati ed estendibili.

Il software MOMIS utilizza Torque per la gestione del database lessicale MOMISWND e la stessa scelta è stata fatta per la gestione del database lessicale multilingua. Come requisito fondamentale per la gestione di un database Torque ha bisogno di essere inizializzato attraverso alcuni parametri, i quali solitamente identificano i parametri di connessione ad un database. All'interno di MOMIS questi parametri sono forniti a Torque attraverso il file di configurazione siDesigner.conf con la seguente sintassi:

```
#
# MySql (default per english WordNet)
#
#torque.database.MOMISWN.adapter = mysql
#torque.dsfactory.MOMISWN.factory = org.apache.torque.dsfactory.SharedPoolDataSourceFactory
#torque.dsfactory.MOMISWN.factory = org.apache.torque.dsfactory.SharedPoolDataSourceFactory
#torque.dsfactory.MOMISWN.connection.driver = org.gjt.mm.mysql.Driver
#torque.dsfactory.MOMISWN.connection.url = jdbc:mysql://localhost:3306/momiswnd20
```

[9] Official Site http://db.apache.org/torque/

```
#torque.dsfactory.MOMISWN.connection.user = root
#torque.dsfactory.MOMISWN.connection.password = root
```

Avendo implementato un database lessicale differente da quello impostato automaticamente dal programma si è provveduto ad aggiungere una nuova specifica di configurazione che memorizzasse i parametri di connessione corretti mediante l'aggiunta delle seguenti righe di codice e si è provveduto a commentare le precedenti per evitare un'inizializzazione errata di Torque:

```
#
# MySql (Momis for MultiWordNetIt)
#
torque.database.MULTIWORDNETIT.adapter = mysql
torque.dsfactory.MULTIWORDNETIT.factory = org.apache.torque.dsfactory.SharedPoolDataSourceFactory
torque.dsfactory.MULTIWORDNETIT.connection.driver = org.gjt.mm.mysql.Driver
torque.dsfactory.MULTIWORDNETIT.connection.url = jdbc:mysql://localhost:3306/multiWordNetit
torque.dsfactory.MULTIWORDNETIT.connection.user = root
torque.dsfactory.MULTIWORDNETIT.connection.password = root
```

Sempre nello stesso file è presente anche un altro parametro di configurazione oggetto di una modifica:

```
torque.database.default=MOMISWN
```

Questo parametro si occupa di impostare il database di default gestito da Torque all'interno del progetto. Anche in questo caso si è provveduto a creare una nuova riga di codice, mostrata poco sotto e a commentare la precedente:

```
torque.database.default=MULTIWORDNETIT
```

3.8.2 Enum AnnotationLanguage

Sinora le modifiche apportate al progetto MOMIS hanno riguardato la struttura stessa del dizionario multilingua integrato o parti di progetto "periferiche". Con questo paragrafo e con i seguenti si vuole dare una breve spiegazione di quali siano state invece le modifiche apportate al codice sorgente del progetto. Il primo quesito che è sorto in fase di sviluppo è stato come poter definire e utilizzare lingue differenti all'interno del progetto.

La scelta effettuata ha comportato la creazione di una classe **Enumerated Type,** o **Enumerazione,** la quale consente di definire un insieme di valori predefiniti, senza ricorrere alla mediazione di costanti intere, con la possibilità però di avere in tempo reale un Oggetto Java vero e proprio.

Tale Enumerazione è stata chiamata **`AnnotationLanguage`,** ed è stata implementata secondo lo schema riportato in Figura 18:

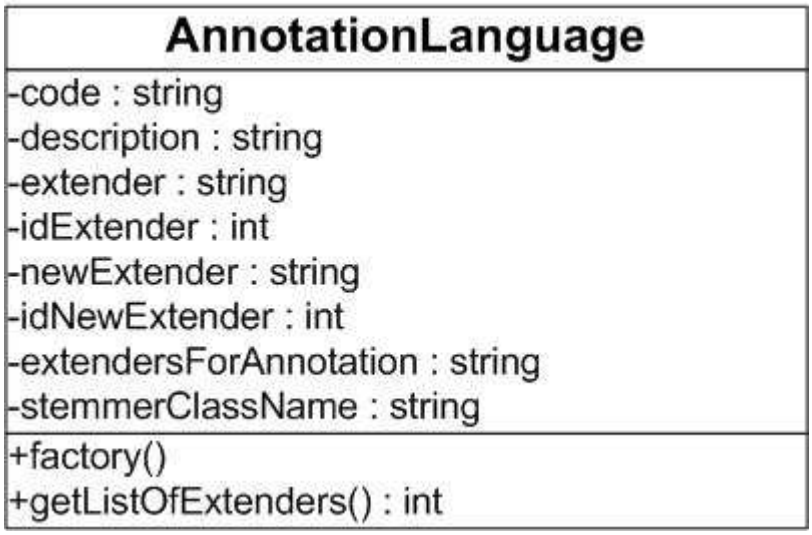

Figura 18 Enumerazione AnnotationLanguage

L'elenco dei valori predefiniti all'interno di questa classe è dato, per ora, dalle due lingue del dizionario MultiWordNet, ormai assunto come nuova risorsa lessicale di MOMIS, ed assume i seguenti valori:

```
ENGLISH(
      "e",
      "English Language",
      "mwnen",
      5,
      "mwnen",
      2,
      "1;2;5"
      "it.unimo.dbgroup.momis.projects.autoAnnotation.algorithms.stemmers.Porter
      StemmerForAnnontation.java"
);

ITALIANO(
      "i",
      "Italian Language",
      "mwnit",
      4,
      "mwnitnew",
      6,
      "4;6"
      "it.unimo.dbgroup.momis.projects.autoAnnotation.algorithms.stemmers.Porter
      ItalianStemmerForAnnotation.java"
);
```

Dato il loro frequente utilizzo nelle modifiche successive viene data una breve descrizione degli attributi dell' Enumerazione in maniera tale da consentire una più rapida comprensione dei termini ad essa correlati.

- **code**: codice in formato String della Lingua. Usato la maggior parte delle volte come identificativo della stessa e come parametro passato al metodo `factory();`

- **`description`**: descrizione in formato String. Usato all'interno del programma come attributo da mostrare all'Utente;
- **`extender`**: codice in formato String usato all'interno del database MultiWordNet nella tabella WN_EXTENDER;
- **`idExtender`**: codice in formato Intero usato all'interno del database MultiWordNet in diverse tabelle per identificare l'origine di un *lemma*, di un *synset* o di una relazione e all'interno del software come parametro passato ai metodi del motore di persistenza Torque qualora si debbano effettuare operazioni sul dizionario che comportano l'uso specifico di una determinata lingua;
- **`newExtender`**: codice in formato String usato all'interno del database MultiWordNet nella tabella WN_EXTENDER per la memorizzazione di nuovi record;
- **`newExtender:`** codice in formato Intero usato all'interno del database MultiWordNet in diverse tabelle per memorizzare nuovi record;
- **`extendersForAnnotation:`** serie di extender utilizzati per la ricerca di record in una determinata lingua;
- **`stemmerClassName`**: nome della classe Stemmer in formato String. Il suo utilizzo verrà specificato in un paragrafo successivo. Identifica quale classe caricare all'interno di MOMIS per effettuare operazioni di *stemming* nella fase di Annotazione Manuale.

3.8.3 Impostazione Lingua "lato Backend"

La naturale conseguenza delle modifiche descritte nel paragrafo 3.1.3, riguardo l'inizializzazione del motore di persistenza Torque è stata l'analisi e la successiva implementazione di logiche che potessero tener conto esclusivamente dei parametri impostati nel file di configurazione per la gestione di un dizionario multi-lingua ed è consistita nella riscrittura di diversi metodi adibiti alla gestione dei dati sul dizionario nonché all'implementazione di nuovi metodi qualora si sia reso necessario.

Una breve descrizione delle modifiche apportate al codice di MOMIS è data nell'elenco seguente.

- Creazione di una nuova mappatura delle entità del database lessicale MultiWordNet, all' interno del package **`it.unimo.dbgroup.momis.om.map`** mediante la creazione delle classi **`MapBuilder`**, una per ogni entità dello schema. Tali classi si rappresentano *l'immagine* Java dell'entità relazionale alla quale sono collegate. Al loro interno sono

definite tante variabili quanti sono gli attributi di un entità, coerenti con il rispettivo *data type* e viene implementato il metodo **`doBuild(),`** il quale a partire dal nome del dizionario, e quindi dello schema stesso, crea un oggetto **`TableMap`** corrispondente all'entità. Per consentire un corretto utilizzo sono state quindi create le classi:

- **`MwnExtenderMapBuilder`**: mappatura della tabella WN_EXTENDER;
- **`MwnLemmaMapBuilder`**: mappatura della tabella WN_LEMMA;
- **`MwnSynsetMapBuilder`**: mappatura della tabella WN_SYNSET;
- **`MwnLemmaSynsetMapBuilder`**: mappatura della tabella WN_LEMMA_SYNSET;
- **`MwnRelationshipMapBuilder`**: mappatura della tabella WN_RELATIONSHIP;
- **`MwnRelationshipTypeMapBuilder`**: mappatura della tabella WN_RELATIONSHIP_TYPE;
- **`MwnReverseIndexMapBuilder`**: mappatura della tabella WN_REVERSE_INDEX.

- La mappatura delle nuove entità all'interno del codice di MOMIS non basta a consentire un uso corretto del dizionario. E' Torque, come detto precedentemente, che si occupa della generazione automatica della struttura di classi *`Peer`* che permettono il dialogo fra codice sorgente e database relazionale. In queste classi sono però presenti alcune logiche, o nel caso peggiore delle variabili statiche, che dipendono fortemente dal nome dello schema del database attraverso il quale sono state create e sinora utilizzato normalmente in MOMIS. Questo vanifica quanto fatto nel file *siDesigner.conf* riguardo l'inserimento di un diverso database lessicale di default e la creazione di nuovi parametri di connessione. Durante l'esecuzione del progetto ciò che viene impostato mediante il file di configurazione viene sostituito da ciò che invece è impostato direttamente nelle sorgenti del programma, diventando fonte di errori durante il caricamento o la modifica dei dati del dizionario. Per eliminare questi errori sono state corrette tutte le classi attraverso l'implementazione di logiche che si basano sulla possibilità di stabilire quale database lessicale utilizzare solamente attraverso il file di configurazione, partendo dall'assunto che in base a questi parametri devono essere caricate coerentemente le classi di Mapping e le classi autogenerate da Torque. Torque genera infatti una struttura che rispecchia lo schema riportato in Figura 19. Le classi con prefisso Base contengono la logica generata da Torque in termini di metodi di accesso al singolo record e alla tabella per interrogazioni e modifiche. Le restanti classi aggiungono invece la logica di business, ovvero quella che serve all'applicazione per operare sui dati. All' interno della classe `BaseWnSynsetPeer`, al momento della

instanziazione è stata inserita la seguente logica, valida per tutte le classi "*Base*" relative alle diverse entità:

```
try
{
      if(DATABASE_NAME_MWN.equals(Torque.getDefaultDB())) {
            Torque.getMapBuilder(MwnSynsetMapBuilder.CLASS_NAME);
      } else {
            Torque.getMapBuilder(WnSynsetMapBuilder.CLASS_NAME);
      }
}
```

E' possibile notare come venga caricata una differente classe di mapping in base a un parametro dato dal nome del database lessicale al quale si vuole accedere.

Un'altra delle modifiche effettuate in questo contesto ha coinvolto tutti quei metodi che recuperavano il database di default mediante la variabile statica poc'anzi descritta e non mediante il parametro di configurazione. Torque permette infatti di recuperare questo parametro mediante l'utilizzo di un metodo statico di cui si dà il seguente esempio:

```
public static TableMap getTableMap() throws TorqueException {
      return Torque.getDatabaseMap(Torque.getDefaultDB()).getTable(TABLE_NAME);
}
```

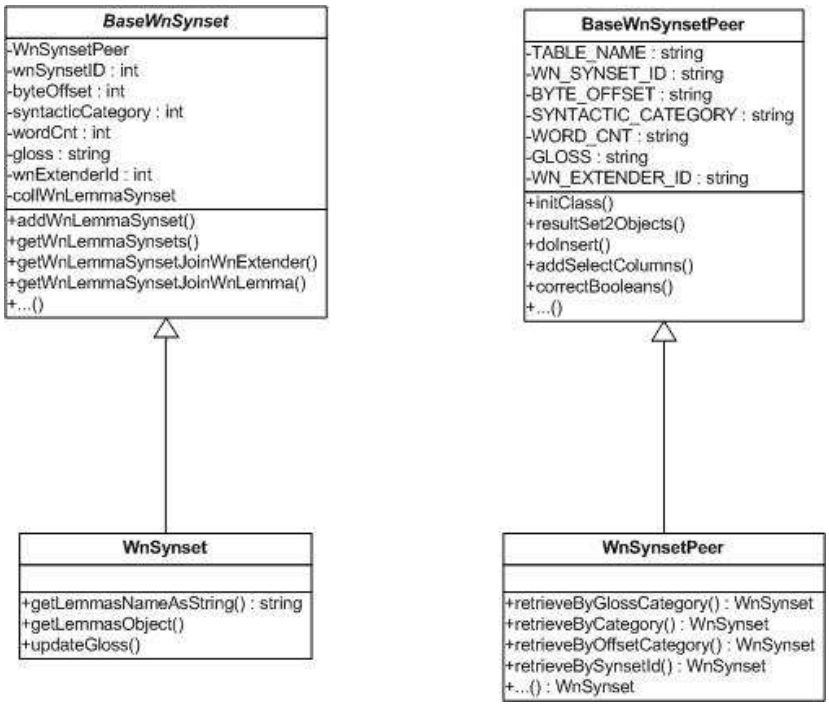

Figura 19 Schema classi generate da Torque

3.8.4 Impostazione Lingua “lato Frontend”

Le modifiche sulla logica interna del progetto hanno avuto come scopo la creazione delle condizioni necessarie per il caricamento corretto di un database lessicale multilingua e per il suo utilizzo. Durante la fase di sviluppo di queste modifiche si è tenuto anche conto di una delle caratteristiche fondamentali di MOMIS, ovvero l’interazione con l’utente durante ogni singola fase del Processo di Integrazione. Si è costituita perciò la *base* di alcune variazioni delle interfacce grafiche del tool **SI-Designer**. Principalmente hanno riguardato la possibilità di poter scegliere in quale lingua poter annotare una sorgente,e di conseguenza associarla alla sorgente stessa, e successivamente con quale lingua gestire l’inserimento o la ricerca di record all’interno di MultiWordNet. Di seguito vengono elencate brevemente tutte le modifiche o aggiunte riguardanti le GUI di MOMIS:

- **Scelta *AnnotationLanguage* nel tab Sources.** Si è scelto di poter dare all’utente di MOMIS la possibilità di scegliere quale AnnotationLanguage impostare in fase di caricamento della sorgente. Ovviamente una conoscenza a priori delle fonti che si vogliono integrare è fondamentale per questa scelta. E’ bene precisare che in caso contrario nulla viene compromesso durante il Processo di Integrazione. Nel caso non venga scelta alcuna AnnotationLanguage le procedure di Annotazione verranno semplicemente eseguite indipendentemente da questo parametro, dando la possibilità di annotare una sorgente in più lingue. In Figura 20 è riportata l’interfaccia che permette di scegliere l’AnnotationLanguage.

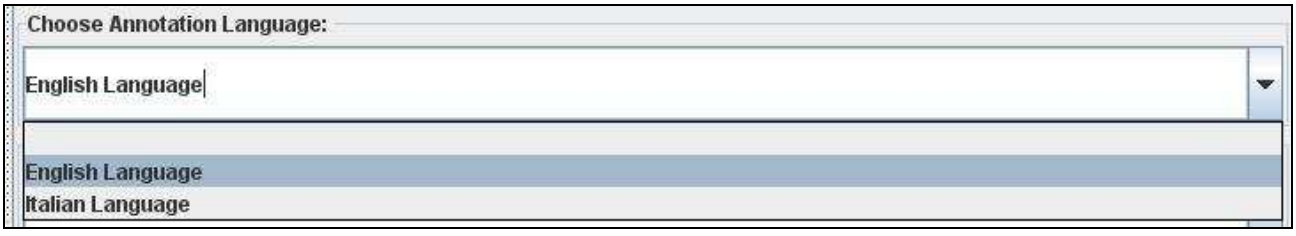

Figura 20 Menu per la scelta dell' Annotation Language

- **Scelta AnnotationLanguage nel tab Annotation.** Quanto sviluppato nella modifica precedente è stato preso come spunto per una modifica che riguarda la fase di Annotazione Manuale del Processo di Integrazione. In questa fase, visualizzata

dall'utente nel tab Annotation, è possibile scegliere e visualizzare il *synset* per ciascuna entità della fonte selezionata in precedenza. Avendo ora a disposizione un dizionario multilingua è stata considerata l'ipotesi di poter effettuare le ricerche sulla base di una delle lingue a disposizione, diminuendo il carico computazionale e allargando o restringendo l'insieme dei risultati ottenuti dal database lessicale. Tale modifica è stata implementata tramite un menù a tendina simile a quello presente nell'interfaccia grafica dei Wrapper, il quale è stato opportunamente inserito in alcune delle interfacce e delle finestre Popup mostrate in questa fase. Tale modifica ha riguardato anche il tool **WordNet Editor** [20], strumento di modifica dei record di WordNet, adattato in questo contesto per permettere la modifica di record multilingua. In figura 21 viene mostrata una delle sezioni di programma oggetto di tali modifiche.

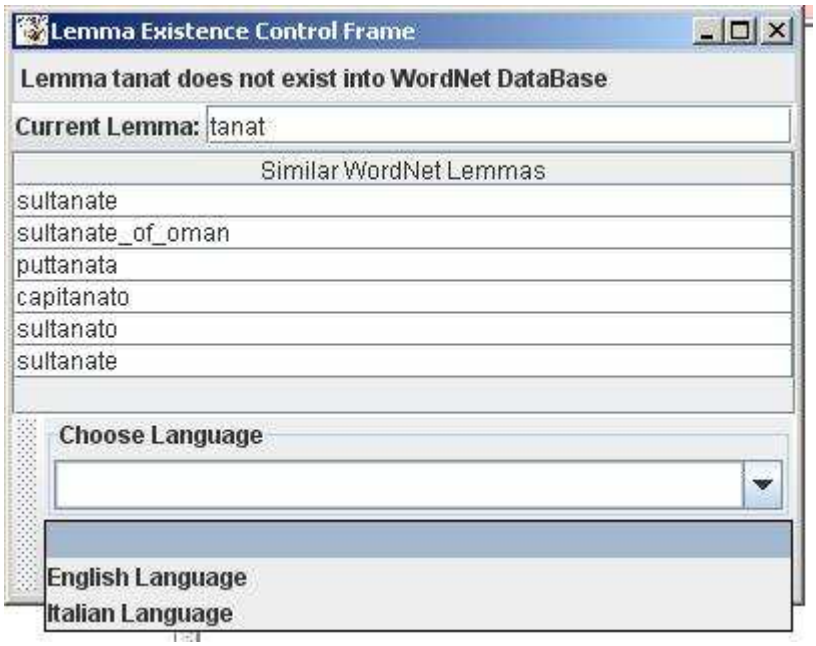

Figura 21 Popup di ricerca dei Lemmi simili

- **Presentazione Glossa.** Come detto nel Capitolo 3.4 MultiWordNet è un progetto che si è prefissato l'obiettivo di implementare un database lessicale multilingua strettamente basato sulla struttura di Princeton WordNet. In poche parole la maggior parte dei *synset* presenti nelle diverse implementazioni di MultiWordNet sono una immagine speculare dei *synset* di WordNet, *identifier* compresi. Proprio per questo motivo nel database italiano di MultiWordNet non sono quasi mai riportate le glosse dei *synset* in italiano, in quanto espressione dello stesso concetto. Solo una minima parte presenta una glossa o ancora nel peggiore dei casi una *phrase*. Ciò ha comportato diverse tipologie di errore durante l'utilizzo del pannello **Annotation** del **SI-Designer** tool, il quale mostra in

diverse sezioni il risultato del processo di annotazione tramite un elenco di lemmi e dei rispettivi synset. A seconda della sezione in cui viene visualizzata la glossa si sono avute principalmente due tipologie di errori:

- `NullPointerException` lanciata dal programma;
- Campo "*empty*" per l'attributo glossa;

Per risolvere queste casistiche è stata effettuata la scelta di visualizzare la glossa del *synset* inglese corrispondente, riuscendo comunque a fornire all'utente un'informazione soddisfacente sull'annotazione in corso.

3.8.5 Modifiche agli Algoritmi di Annotazione Automatica

Come ampiamente descritto da Serena Sorrentino [25], all'interno di MOMIS sono implementati alcuni algoritmi di Annotazione Automatica i quali consentono di svolgere delle procedure di disambiguazione del testo in maniera semi-automatica. L'annotazione manuale infatti, per quanto controllata direttamente dall'utente, potrebbe risultare un processo particolarmente oneroso se si è in presenza di fonti con un elevato numero di entità. Il caso in esame studiato nel Capitolo 5 presenterà proprio questo tipo di problema. Gli algoritmi di Annotazione Automatica intervengono esattamente in questi casi, permettendo di svolgere autonomamente le operazioni di annotazione mediante metodi di disambiguazione del testo ben documentati [26].

Tali metodi di disambiguazione del testo si dividono in due macro-categorie:

- *Algoritmi supervisionati.* Sono metodi che necessitano dell'uso di collezioni di testi etichettati manualmente; richiedono l'intervento di una persona che sia in grado di effettuare una identificazione delle parole all'interno di una frase come pertinenti ad un senso piuttosto che a un altro ;
- *Algoritmi non supervisionati.* Metodi che non richiedono una particolare supervisione da parte dell'utente. Si basa sull'utilizzo di ontologie o dizionari come risorsa lessicale di riferimento;

Gli algoritmi di Annotazione Automatica sono raggruppati in MOMIS all'interno della sezione **ALA** (**A**utomatic **L**exical **A**nnotator) e possono essere usati indipendentemente l'uno dall'altro o in combinazione, stabilendo ordine e modalità di esecuzione. Inoltre si ha la possibilità di intervenire sui alcuni parametri dell'Algoritmo per migliorare e raffinare il processo di annotazione. In generale la loro funzione è quella di estrarre le relazioni presenti tra i concetti dell'ontologia in

modo da contestualizzare ciascun termine e in seguito confrontare il termine stesso con i legami presenti nel database lessicale. Gli algoritmi di annotazione Automatica presenti in MOMIS sono:

- **Structural Disambiguation algorithm (SD).** Algoritmo di disambiguazione strutturale. Ricava dallo schema ODL_I^3 le relazioni strutturali tra gli elementi dello schema, per dedurne le relazioni lessicali in riferimento a WordNet.
- **WordNet First Sense algorithm (WNFS).** Semplice algoritmo il quale per ciascun termine ritorna il primo senso del thesaurus di WordNet, che risulta quindi essere il senso più comune e usato per definirlo. Tale algoritmo viene sempre utilizzato nel Processo di Integrazione.
- **WordNet Domains algorithm(WND).** Algoritmo che fa riferimento a WordNet Domains. A partire dai *synset* estratti da ciascun termine vengono ricavati i domini associati e con queste informazioni calcola una lista di domini prevalenti. Successivamente vengono estratte le informazioni necessarie per determinare gli algoritmi prevalenti del contesto e i termini vengono annotati con i *synset* che appartengono a quei domini.
- **Gloss Similarity algorithm (GS).** Questo algoritmo utilizza un metodo di disambiguazione del significato basato sull'estrazione delle glosse collegate ai termini all'interno di WordNet. Ciascuno di esso è infatti legato ad una o più glosse che descrivono tutti i possibili sensi della parola, e ad una serie di frasi-esempio che descrivono il possibile uso di ciascun significato della parola in un certo contesto.
- **Gloss Similarity Iterator algorithm (IGS).** Propone lo stesso metodo dell' algoritmo GS, ma invece che far riferimento alle glosse di un solo termine, considera tutte le glosse presenti nello stesso contesto.

Come risulta evidente tutti gli algoritmi sinora presenti in MOMIS sono strettamente legati al database lessicale originariamente importato, ovvero Princeton WordNet. E' quindi palese il fatto che questi algoritmi non riescano ancora a offrire un valido supporto nell'annotazione semi-automatica di una sorgente multilingua. A livello di codice, ciò è dato dal mancato riferimento alla lingua nelle classi Java che implementano tali algoritmi. In base al caso di studio esaminato nel capitolo 5 verranno presi in considerazione gli algoritmi WNFS e WND per lo sviluppo di alcune modifiche. Si è scelto di scartare l'algoritmo SD poiché la fonte dati integrata non presenta caratteristiche particolarmente rilevanti da un punto di vista strutturale, mentre non sono stati presi in considerazione gli algoritmi GS e IGS poiché, basandosi su un' analisi dettagliata delle glosse dei

synset non avrebbero dato risultati positivi a causa delle lacune riscontrate nei capitoli 3.4.1 e 3.4.2.

Gli algoritmi WNFS e WND, basandosi esclusivamente sui *synset* e sui *lemma*, possono costituire un importante supporto all'Annotazione Automatica, previa la realizzazione di eventuali modifiche alle classi che li implementano. Tali modifiche hanno comportato una modifica nei metodi **`getMeanings(String lemma, OntologyManager ontologyManager )`** nel quale si è provveduto a verificare l'esistenza di un lemma e successivamente a recuperare i *synset* in base alla `AnnotationLanguage` di cui è stata data possibilità di scelta mediante la funzionalità aggiunta nell'interfaccia del Wrapper e descritta nel capitolo 3.8.2.

3.8.6 Stemming in fase di Annotazione Manuale e analisi dei termini composti di una sorgente.

Lo **stemming** è il processo di riduzione della forma flessa di una parola alla sua forma radice, detta **tema**. Il tema non corrisponde necessariamente alla radice morfologica (*lemma*) della parola: normalmente è sufficiente che le parole correlate siano mappate allo stesso tema (ad esempio, che *andare*, *andai*, *andò* mappino al tema *and*), anche se quest'ultimo non è una valida radice per la parola.

Nell'ambito della disambiguazione del termini all'interno di MOMIS risulta quindi essere una valida metodologia per il riconoscimento di una forma base di un *lemma* e la presenza dello stesso all'interno di WordNet. Per esempio il *lemma* "**cats**" non ha un rifermento all'interno di WordNet, in quanto forma plurale del *lemma* "**cat**", il quale invece risulta essere presente. Essendo impensabile di inserire nuovi *lemma* e nuovi *synset* per la gestione delle forme flesse in MOMIS è stato da tempo sviluppato un algoritmo di stemming per la risoluzione di queste casistiche. L'algoritmo si basa sugli studi di **Martin F. Porter** e del suo **Porter Stemming Algorithm** [27], il quale è un algoritmo che si occupa di rimuovere le influenze morfologiche nei termini in lingua inglese ed è largamente usato nell'ambito dell'Information Retrieval. Questo algoritmo è stato ampiamente tradotto nei più comuni linguaggi di programmazione, di scripting e nei differenti dialetti SQL[10].

Il progetto MOMIS, utilizzando questo algoritmo nella sua implementazione Java con la classe **`PorterStemmerForAnnotation`** e richiamandola in fase di Annotazione di una sorgente, è in grado eseguire una prima verifica sulla forma flessa dei *lemma* in lingua Inglese presenti nella sorgente integrata e ricavarne la conseguente forma base, la quale viene presa in esame dall'algoritmo WNFS descritto in precedenza.

[10] Official Site http://tartarus.org/~martin/PorterStemmer/

Anche in questo caso però la funzionalità già presente è strettamente legata al database WordNet. Non solo, sebbene l'algoritmo di Porter sia ormai largamente utilizzato e costituisca quindi una certezza nell'ambito dell'Information Retrieval rimane comunque limitato a causa della sua applicazione su contesti in lingua inglese. Esistono però delle diverse implementazioni di questo algoritmo che riguardano altre lingue, fra cui il progetto **Snowball**, un linguaggio di *string processing* , che annovera anche lo stesso Martin Porter fra i creatori[11]. Fra le lingue per le quali l'algoritmo di Porter è stato "tradotto" è presente anche l'italiano[12].

L'implementazione di un nuovo algoritmo di stemming che si basi su sorgenti in lingua italiana costituirebbe quindi un importante valore aggiunto per l'integrazione di fonti multilingua, obiettivo principale di questa Tesi. Tale implementazione ha comportato il *refactoring* di alcuni package e la conseguente creazione del package **`it.unimo.dbgroup.momis.projects.autoAnnotation.algorithms.stemmers.`**

All'interno di questo nuovo package sono state importate le classi precedentemente create per la risoluzione dell'algoritmo di Porter e sono state successivamente create alcune classi per la gestione dell'algoritmo in "versione italiana". La scelta per cui si è optato è stata quella di creare una classe astratta, **`AbstractStemmer,`** nella quale sono definiti dei metodi astratti che devono essere implementati nelle diverse versioni dell'algoritmo. La scelta di utilizzare una classe astratta permette l'istanziazione dela classe estesa da utilizzare *runtime* in base all'AnnotationLanguage scelta nelle precedenti fasi del Processo di Integrazione. Infatti, come accennato nel capitolo 3.8.2, uno degli attributi di questa Enumeration è proprio la classe di stemmer relativa alla lingua, ed è mediante questo attributo, passato come parametro al metodo **`Class.forName(String class)`** che si ha la possibilità di utilizzare la giusta versione dell'algoritmo.

[11] http://snowball.tartarus.org/texts/introduction.html
[12] http://snowball.tartarus.org/algorithms/italian/stemmer.html

4 MOMIS “Datariver”: creazione della vista materializzata dello Schema Globale

Riassumendo quando analizzato, MOMIS è in grado di integrare diverse sorgenti dati locali, procedere alla disambiguazione delle diverse componenti di queste sorgenti e successivamente di generare una Vista Globale (GVV, Global Virtual View) attraverso la quale possibile interrogare globalmente i dati delle diverse fonti. Tale procedimento è stato ampiamente descritto nel Capitolo 1.3 ed è chiamato Processo di Integrazione. Per quanto riguarda le problematiche di disambiguazione dei termini si faccia invece riferimento al Capitolo 3. In altre parole MOMIS è in grado di “*presentare*” i dati delle fonti locali come se fossero un'unica sorgente, indipendentemente dai tipi di fonte a cui si fa riferimento. Ciò che invece MOMIS ancora non è in grado di fare è provvedere a materializzare fisicamente la GVV e i dati contenuti in essa in un DBMS o sotto diverse forme. Ritenendo che possa essere un aspetto interessante, sia nel contesto della Tesi, sia per eventuali sviluppi futuri, viene proposta una prima soluzione per la materializzazione della GVV, mediante l’inserimento di una funzione che permetta di creare la vista materializzata dello Schema Globale di MOMIS su un database relazionale.

4.1 MOMIS “Datariver”

Per lo sviluppo di queste ultime funzionalità si è momentaneamente abbandonata la versione originale del progetto MOMIS utilizzata sinora e ci si è concentrati sull’utilizzo di una versione differente dello stesso progetto, la versione **Datariver**[13]**.**

La versione Datariver di MOMIS parte nel 2008 sempre ad opera del **DbGroup** dell’Università di Modena e Reggio Emilia. Tale versione è la conseguenza diretta della “prodottizzazione” nel progetto MOMIS [28]. Sostanzialmente questa versione presenta gli stessi moduli di MOMIS, eccetto il modulo ALA di Annotazione Automatica ed il tool Query Manager. Si è scelto infatti di optare per un totale *refactoring* del Query Manager e di dotare il sistema di nuove funzioni di gestione del progetto. Nell’elenco seguente sono indicate le principali differenze e innovazioni della versione Datariver rispetto la versione originale:

- Riorganizzazione del progetto e dei package i sotto-progetti tramite l’utilizzo di

[13] http://www.dbgroup.unimo.it/datariver/

Maven[14], progetto di Apache per la gestione dei progetti;

- *Refactoring* delle interfacce grafiche del progetto MOMIS mediante l'utilizzo dell'ambiente **Eclipse – RCP**, una versione dell' IDE **Eclipse** che permette lo sviluppo di una grafica basata sull'ambiente di sviluppo stesso;
- *Refactoring* del tool Query Manager e utilizzo come database di supporto **HSQLDB[15] (HyperSQL Database)**, un RDBMS scritto completamente in Java **;**
- Gestione di WordNet tramite il formato originale di distribuzione o tramite **JPA[16] (Java Persistence API),** motore di persistenza dei dati scritto totalmente in Java e che sfrutta il linguaggio **JPQL (Java Persistence Query Language)**;
- Sviluppo di un' architettura Server per il Query Manager nell'ottica di poter fornire dei Web Service per l'estrapolazione dei dati integrati;

Nonostante queste differenti caratteristiche il Processo di Integrazione implementato in Datariver è esattamente lo stesso della versione originale di MOMIS, per cui non si ritiene necessario proseguire oltre con la descrizione.

L'uso di Datariver all'interno di questa Tesi si colloca dopo la generazione della GVV tramite il progetto MOMIS di una sorgente dati in lingua italiana. Attraverso Datariver verranno effettuate alcune interrogazioni di test e, in seguito ad alcune fasi di sviluppo, verrà effettuata un' esportazione dei dati verso una sorgente PostgreSQL, opportunamente predisposta.

Prima di poter proseguire con la descrizione di quanto svolto, è necessario precisare che alcune delle modifiche effettuate al codice di MOMIS, mostrate nei Capitoli 2 e 3.8.2 sono state riportate anche nella versione Datariver. Si è dovuto infatti provvedere alla creazione di un Wrapper per sorgenti PostgreSQL, che consentisse la connessione con un DBMS locale, e l'importazione del dizionario MultiWordNet, attraverso la modifica di alcuni parametri di configurazione e l'istanziazione di un manager di JPA per la gestione diretta del database relazionale, piuttosto che la gestione tramite i file di origine di WordNet, impostazione di default della versione Datariver. Queste modifiche, essendo in parte già effettuate per il progetto MOMIS, non hanno comportato grossi problemi in fase di sviluppo, in quanto per il Wrapper si è scelto di importare direttamente la classe `WrapperCore_PostgreSQL,` creata precedentemente, mentre per quanto riguarda il database lessicale si è semplicemente provveduto alla modifica dei parametri di configurazione, come descritto nel capitolo 3.8.2. In Figura 22 è riportata una delle interfacce di Datariver:

[14] Offical Site http://maven.apache.org/
[15] http://hsqldb.org/
[16] http://java.sun.com/developer/technicalArticles/J2EE/jpa/

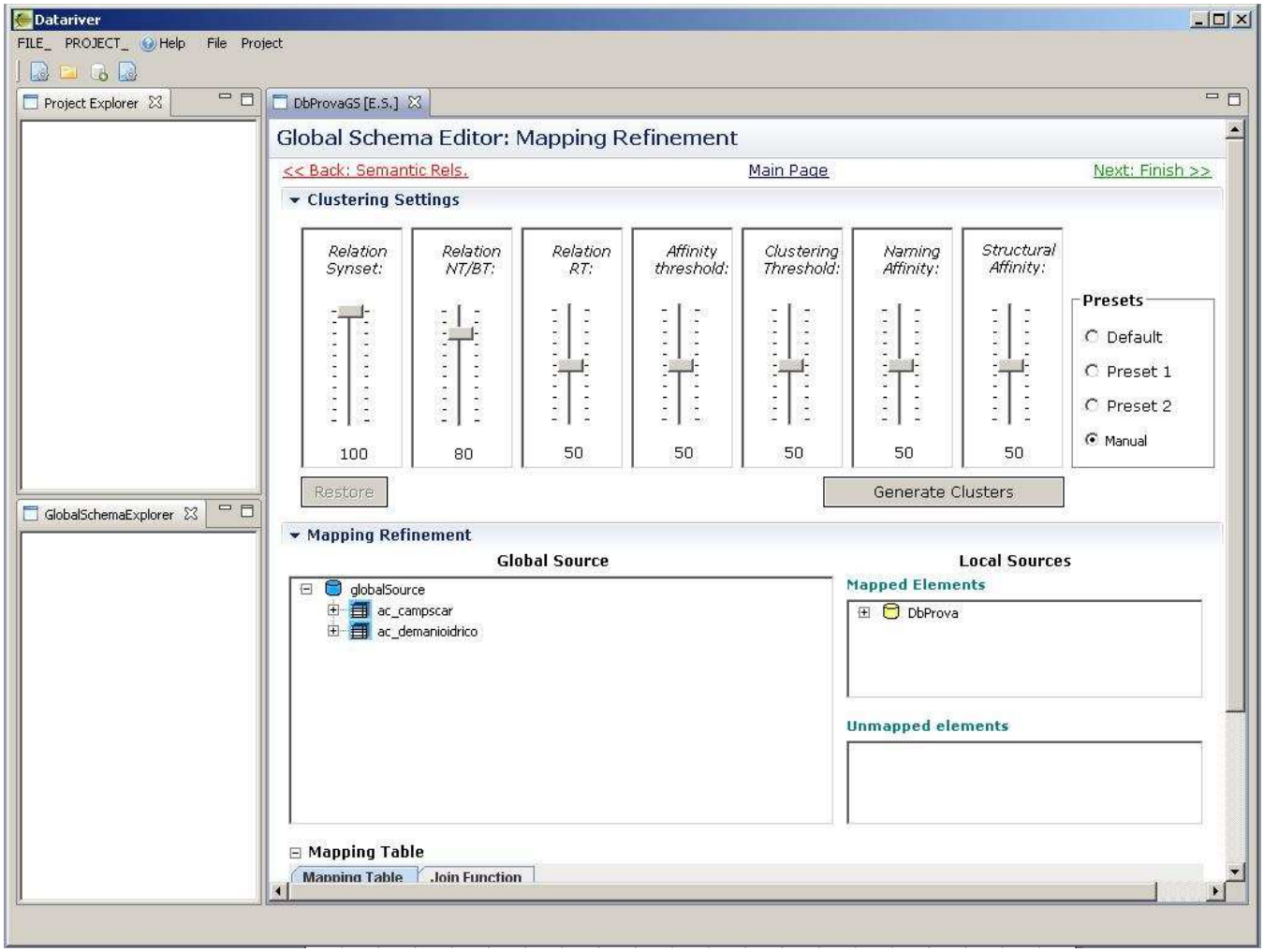

Figura 22 Interfaccia Datariver

Prima di procedere con quanto sviluppato precisare gli obiettivi che ci si è posti per la materializzazione della GVV. Innanzitutto cosa si vuole realmente materializzare? Indipendentemente dal caso di studio analizzato in seguito, stabilire cosa si voglia materializzare non è un problema di facile e immediata risoluzione in quanto apre una serie di quesiti affrontabili solamente con uno studio che meriterebbe sicuramente più tempo. Alla fine del processo di integrazione, con una GVV a disposizione e un collegamento verso un server in uscita tramite uno dei Wrapper implementati si potrebbero presentare essenzialmente 3 ***use case***:

- Creazione del database;
- Creazione dello schema relazionale;
- Esportazione di tutti i dati;

La creazione automatica di un database è un'operazione che risulta essere per ora ancora inadatta

a un *data integration system* in quanto strettamente legata al DBMS sul quale si vuole materializzare la GVV e particolarmente determinante per le successive implementazioni. Inoltre non è compito di un *data integration system* svolgere un'operazione normalmente eseguibile con qualsiasi client SQL. All'interno di questo elaborato si assume quindi di poter materializzare la GVV su un database già esistente.

Supponendo di avere a disposizione un database relazionale già esistente viene affrontato il problema di potervi materializzare la GVV. A livello di implementazione si tratterà di creare le tabelle basate sulle Classi Globali presenti nella GVV tenendo conto del fatto che il linguaggio con il quale è definita la GVV, ODL_I^3, è un linguaggio descrittivo e non presenta informazioni dettagliate sulla struttura delle sorgenti locali integrate, come ad esempio *size* e *precision* per i tipi di dato di un Attribute, o una definizione specifica per i vincoli di integrità. Non solo, lo script di creazione dovrà essere coerente anche con il DBMS di uscita al quale ci si connette. Le soluzioni proposte provvederanno ad ampliare le informazioni presenti nella GVV, in modo tale da poter comporre degli script tramite SQL Standard, indipendenti quindi dal DBMS, e ad eseguire questi script. Inoltre si propone la possibilità di selezionare un Set di GlobalClass da materializzare, rispecchiando quanto già presente durante la fase di caricamento di una nuova sorgente in Datariver.

Stabilito, ed eventualmente creato, lo schema sul quale materializzare i dati, si tratta di capire quali dati esportare. Se da un lato potrebbe essere naturale materializzare tutti i dati ricavabili da una GlobalClass, dall'altro sorgono alcune problematiche su un utilizzo successivo della stessa funzione sullo stesso schema di uscita. Si deve tener conto infatti che questa funzionalità potrebbe essere eseguita più volte sullo stesso schema ma con GVV che potrebbero essere differenti. Cosa accadrebbe nel caso in cui la GVV cambiasse? O ancora, supponendo che i dati presenti sulle sorgenti locali di partenza subiscano delle variazioni nel tempo, come gestire questo aggiornamento? Anche questi sono casi non banali, ma non improbabili in un'applicazione reale e sarebbe necessario uno studio più approfondito. Le soluzioni proposte nell'elaborato si occuperanno di materializzare tutti i dati ricavabili da un Set di GlobalClass. Si propone inoltre la possibilità di poter provvedere alla "pulizia" delle tabelle dello schema fisico mediante una funzione che a partire da un Set di GlobalClass componga uno script basato sull'istruzione SQL Standard `DELETE`.

Per la realizzazione di queste funzionalità che comportano un uso costante della GVV si lavorerà all'interno del Modulo QueryManager, richiamato attraverso nuove interfacce grafiche appositamente create. Si è provveduto quindi alla creazione di una classe specifica all'interno del package blabla: la classe **`QueryManagerExportData`**

4.2 La classe `QueryManagerExportData`

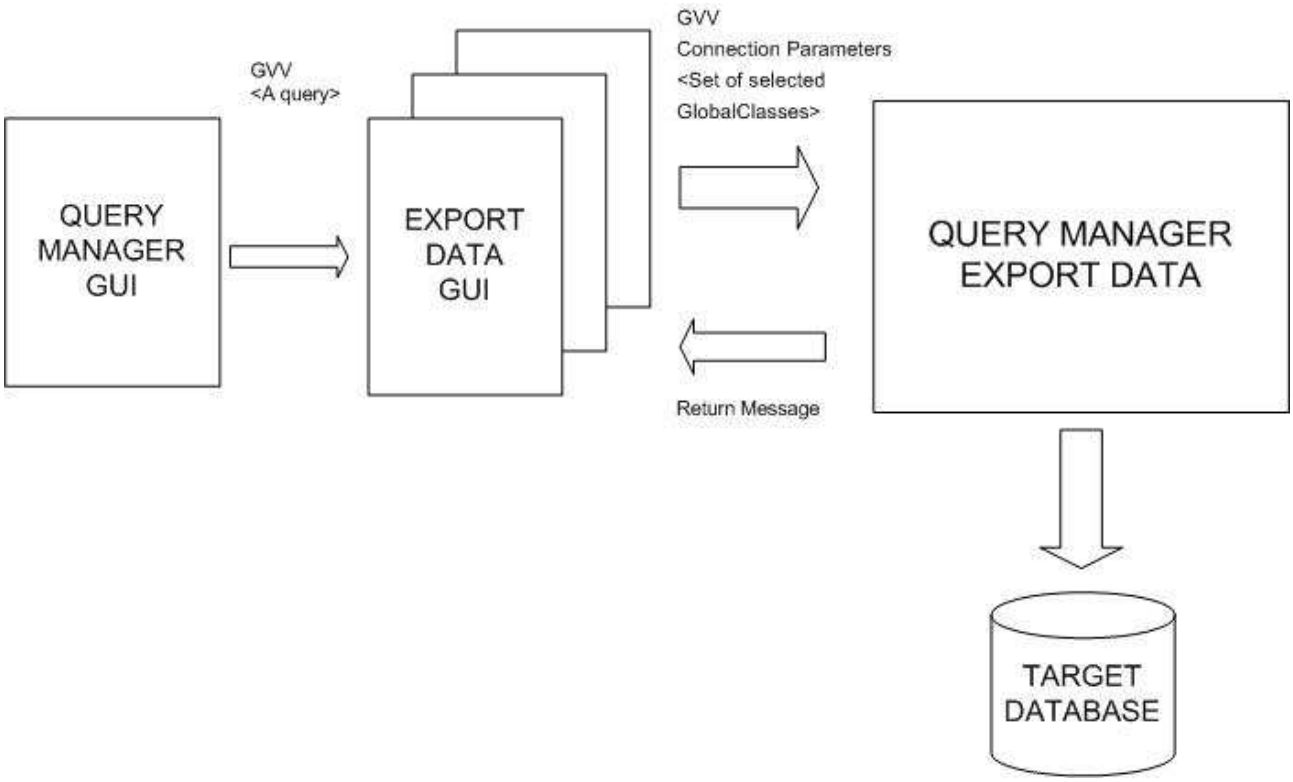

Figura 23 Schema a blocchi per la materializzazione della GVV

In figura 23 è riportato un semplice ma indicativo schema del processo di materializzazione della GVV. A partire dall'interfaccia grafica del modulo QueryManager vengono richiamate la interfacce grafiche tramite le quali è possibile stabilire una connessione con un *target database* sul quale memorizzare la GVV. Il fulcro è il blocco **QUERY MANAGER EXPORT DATA**, rappresentazione della classe `QueryManagerExportData.` Tale classe si occupa di implementare e di invocare i metodi per l'analisi della GVV e per la seguente materializzazione.

La classe riceve in ingresso la GVV stessa, tutti i parametri necessari per la connessione verso il *target database* e, se specificato, una lista di GlobalClasses. Come valore di ritorno viene passato un messaggio di ritorno da visualizzare tramite GUI.

In base alle funzionalità descritte nel paragrafo successivo all'interno di questa classe sono stati realizzati due metodi principali:

- **`String createSchema()`**: metodo che si occupa di recuperare una lista di GlobalClass, di verificare per ciascuna di queste classi se esiste sul *target database* la tabella corrispondente e di generare ed eseguire lo script di creazione della tabella, in

base al dialetto SQL dipendente dal DBMS;

- **`String populateSchema()`**: metodo che si occupa di recuperare una lista di GlobalClass, di recuperare relativi a ciascuna di esse e di materializzarli all'interno della tabella corrispondente nel *target database.* Sostanzialmente per ogni GlobalClass si esegue l'istruzione SQL Standard "**`SELECT * FROM [GlobalClassName]`**", ricavando in questa maniera un `ResultSet`;
- **`String createPartialSchema()`**: metodo che si occupa di recuperare un sottoinsieme di GlobalClass, di verificare per ciascuna di queste classi se esiste sul *target database* la tabella corrispondente e di generare ed eseguire lo script di creazione della tabella, in base al dialetto SQL dipendente dal DBMS;
- **`String populatePartialSchema()`**: metodo che si occupa di recuperare un sottoinsieme di GlobalClass, di recuperare i dati relativi a ciascuna di esse e di materializzarli all'interno della tabella corrispondente nel *target database.* Sostanzialmente per ogni GlobalClass si esegue l'istruzione SQL Standard "**`SELECT * FROM [GlobalClassName]`**", ricavando in questa maniera un `ResultSet`;

La logica che stabilisce quale metodo invocare viene gestita tramite un parametro viene gestita mediante l'utilizzo di un parametro di tipo `Integer` e il metodo **`String exportData(int mode)`**in modo tale da semplificare successivi sviluppi.

4.3 Interfacce grafiche

L'interfaccia grafica di cui si è parlato nel paragrafo precedente è stata sviluppata sulla base delle interfacce grafiche già presenti per l'importazione di una nuova sorgente. I Wrapper vengono in questo caso utilizzati per connettersi a un server sul quale materializzare i dati, completando in questa maniera l'insieme delle funzionalità gestibili da tipologia di servizi.

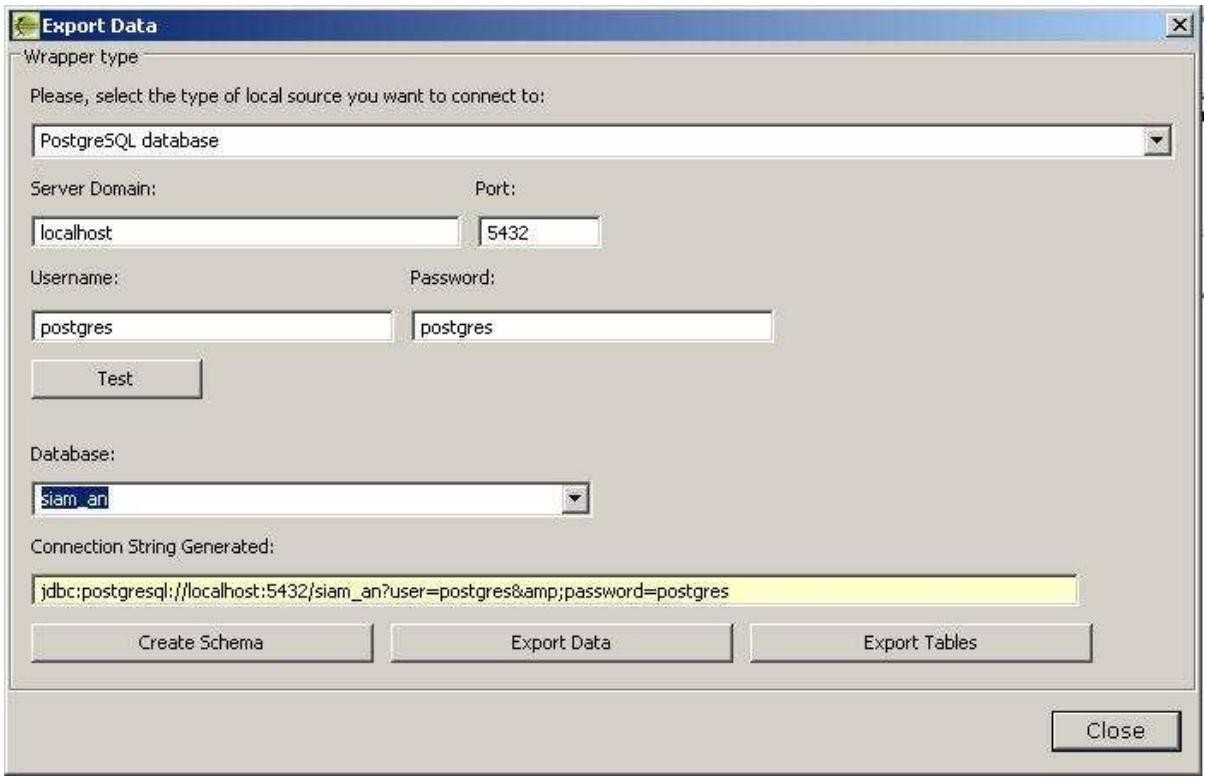

Figura 24 Interfaccia Grafica per la connessione a un target database

Come si nota dalla Figura 24 l'interfaccia presenta una serie di parametri necessari per la connessione a un server e per la scelta di uno dei Wrapper presenti. E' possibile eseguire un test della connessione per recuperare la lista degli database presenti in seguito al quale viene generata in automatico la stringa di connessione JDBC. Infine sono riportati i Button per l'esecuzione delle diverse funzioni:

- Creazione dello schema;
- Esportazione dei dati;
- Esportazione di un sottoinsieme di GlobalClass;

L' ultima funzionalità è quella che consente all'utente di Datariver di costruire manualmente un Set di GlobalClass sul quale eseguire le funzioni di materializzazione e ha comportato la creazione di un' ulteriore interfaccia. Questa interfaccia, mostrata in Figura 25, presenta all'utente di Datariver una lista di GlobalClass presenti nella GVV e consente tramite dei *checkbox* di selezionarle tutte o solo un sotto-insieme e in seguito di procedere con le funzioni di materializzazione.

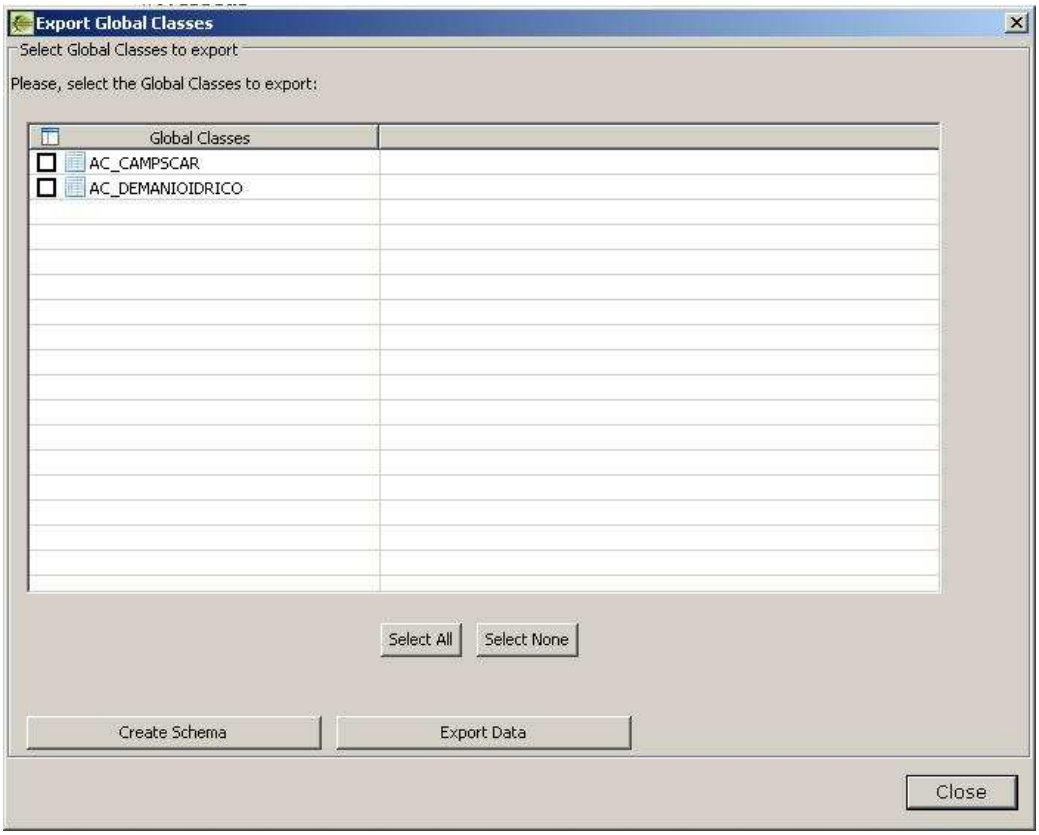

Figura 25 Interfaccia grafica per la selezione delle Global Classes da esportare

4.4 Creazione dello schema

La funzione che si occupa della creazione dello schema viene gestita dai metodi **`createSchema()`** e **`createPartialSchema()`**, in modo totalmente indipendente rispetto alla funzione di materializzazione dei dati. Il metodo si occupa, dopo aver instanziato una connessione con il *target database*, di verificare per ciascuna delle GlobalClass della lista la presenza o meno della tabella corrispondente sul database. Nel caso la tabella non esista si procede con la costruzione dello script di creazione mediante un'istruzione **`CREATE`** SQL STANDARD. Le informazioni necessarie per la definizione dei singoli attributi vengono reperite mediante il metodo **`getAttributes()`**, il quale recupera un oggetto `Map` nel quale la chiave è il nome dell'attributo e il valore un vettore di oggetti `Attribute`. Questa tipologia di oggetti definisce in linguaggio ODL_I^3 tutto quanto concerne un attributo, compreso il tipo definito durante la creazione GVV e mappato sui rispettivi tipi JDBC. Per poter generare correttamente lo script di creazione si è provveduto a modificare alcuni Tipi di Dato ODL_I^3 definiti all'interno delle differenti versioni di MOMIS per poter tenere traccia di alcune informazioni essenziali, quali la *precision* e lo *scale* di un tipo di dato numerico.

La funzione di creazione finisce con l'identificazione dei vincoli di integrità delle tabelle per la

creazione delle Primary Key e delle eventuali Foreign Key e con l'esecuzione dello script. Per dare informazione all'utente un feedback sull'esecuzione della funzione il metodo ritorna un oggetto `String` che tiene conto di messaggi o di eventuali eccezioni.

4.5 L'esportazione dei dati

La funzione di esportazione dei dati viene implementata attraverso i metodi **`populateSchema()`** e **`populatePartialSchema()`** e ricalca come logica la funzione precedente, pur rimanendo totalmente indipendente. Si occupa infatti di creare una connessione al *target database* e di recuperare le GlobalClass che saranno oggetto di esportazione. Per recuperare tutti i dati delle tabelle viene eseguita una istruzione SQL STANDARD "`SELECT * FROM [GlobalClassName]`" utilizzando per l'esecuzione l'oggetto `QueryManagerImpl`. Tale metodo ritorna un oggetto `ResultSet` che può essere opportunamente interrogato per recuperare i dati delle sorgenti locali. Per ogni iterazione dell' oggetto `ResultSet` si provvede alla creazione di uno script di `INSERT`, anche questo attraverso l'SQL STANDARD, e alla sua esecuzione. Anche in questi caso per tenere traccia dello stato di esecuzione della procedura il metodo ritorna un oggetto `String` che presenta all'utente un messaggio visibile in interfaccia.

5 Un caso di studio: SIAM (Sistema Informativo AMbientale)

SIAM (Sistema **I**nformativo **AM**bientale **)** è un programma "*web-based*" che ha lo scopo di gestire il flusso completo di tutte le pratiche elaborate dall'ufficio ambiente delle Province: la gestione delle richieste delle imprese, la realizzazione delle previste istruttorie, il rilascio o meno delle autorizzazioni necessarie e la vigilanza sul rispetto delle normative ambientali vigenti. Elevata automazione dei processi,condivisione e immediata disponibilità dei dati necessari permettono di dedicare la massima concentrazione alle attività a più alto valore aggiunto con una drastica riduzione del carico di lavoro amministrativo e del numero di errori dovuti alla gestione "*manuale*". In Figura 26 è mostrato uno schema delle funzionalità principale che il SIAM propone all'utenza.

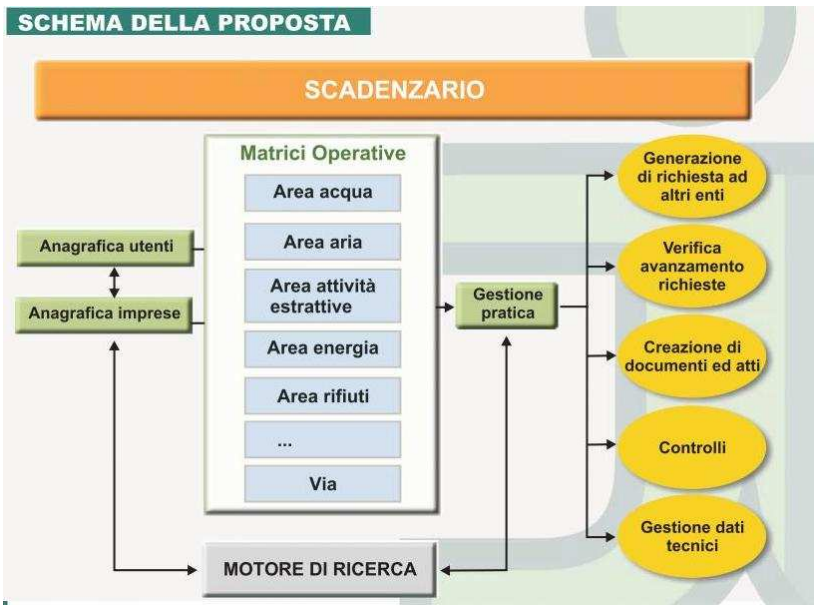

Figura 26 Funzionalità base del sistema SIAM

Essendo una Web Application qualsiasi funzione è direttamente accessibile per mezzo di un normale collegamento alla rete internet, tramite un comune PC e ovunque si trovi l'operatore. E' aperto ad un'interazione diretta fra le imprese e il sistema: le aziende infatti potranno produrre i vari moduli necessari alla presentazione delle richieste , direttamente on-line e per mezzo di un sistema di autenticazione digitale, in un'ottica di semplificazione amministrativa e apertura degli enti pubblici al settore privato.

SIAM è un valido strumento di integrazione e di collaborazione fra tutti i soggetti coinvolti nei processi amministrativi che regolano la gestione dell'ambiente:

- Operatori dell'ufficio ambiente;
- Singole imprese;
- Operatori degli sportelli SUAP dei vari comuni;
- Dirigenti;
- Assessore all'ufficio ambiente;

L' intera struttura del sistema è stata progettata per permettere la piena compatibilità e la possibilità di scambio di informazioni con altri sistemi informatici già implementati dall'Ente. Si ottiene cosi una facile installazione della soluzione, senza interventi sulle applicazioni esistenti e con un recupero dei dati storici svolto nel minor tempo possibile. SIAM è in grado di interagire con:

- Database delle Camere di Commercio per ottenere tutte le informazioni relative alle imprese richiedenti;
- Programma di gestione delle pratiche adottato dallo sportello unico (SUAP) dei comuni, per lo scambio di documenti e informazioni relative lo stato di avanzamento della pratica;
- Programma di gestione del protocollo dell'ente, per l'automatizzazione del processo di entrata/uscita dei documenti, e la verifica dei dati inseriti dagli operatori;
- Programmi di gestione delle pratiche ambientali a livello sovra comunale e provinciale per l'inserimento e la manutenzione automatica delle informazioni necessarie;
- Software di integrazione cartografica quali **GoogleMap, GIS** territoriali in uso alle diverse province, estrazione di dati cartografici sottoforma di ShapeFile;

Attraverso queste funzioni l'operatore non deve ripetere l'inserimento dei dati ai fini di

monitoraggio e rendicontazione limitando le perdite di tempo ed eventuali errori dovuti all'imputazione manuale. I dati inseriti sono disponibili inoltre per essere riportati automaticamente nei rapporti istruttori, nelle determine e negli atti gestiti con SIAM. In Figura 27 è riportato un breve resoconto delle interazioni tra il SIAM e gli Enti o i Software esterni

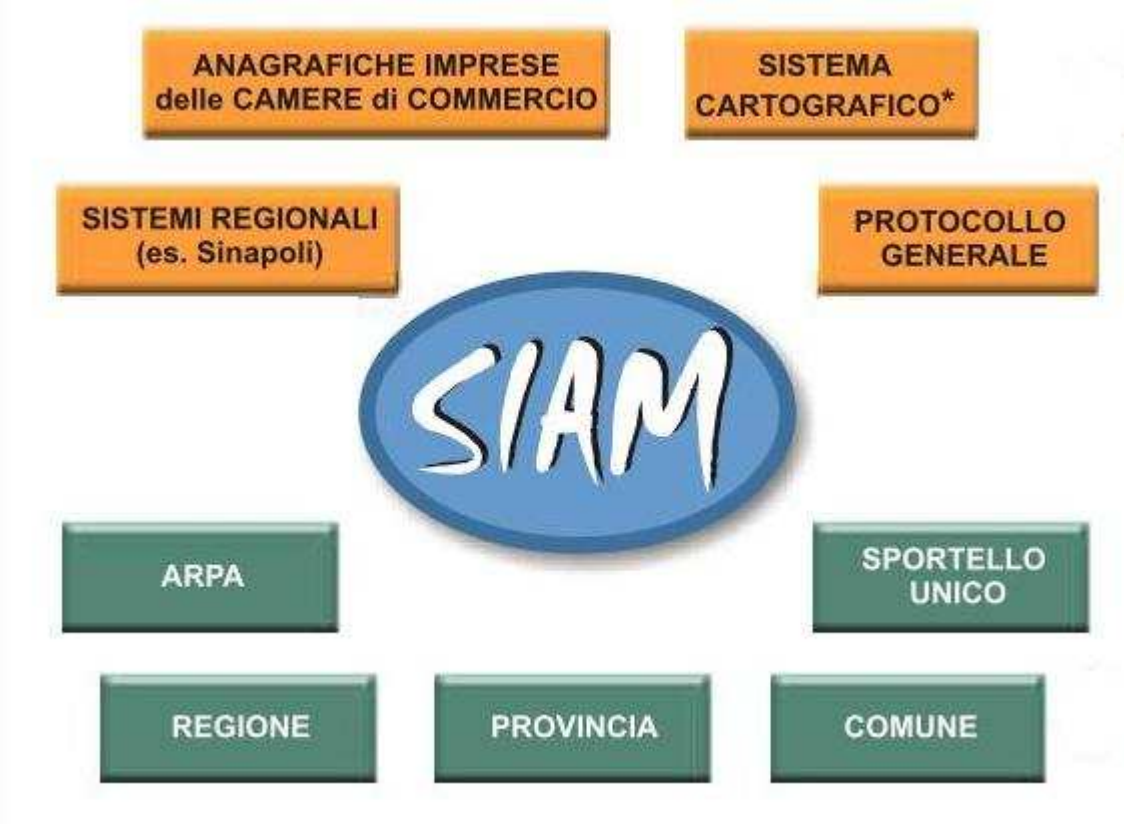

Figura 27 Interazioni SIAM - Enti e Software esterni

Il SIAM si colloca all'interno della "*Suite Ambiente*" sviluppata da **Quix S.r.l** e da **Injenia S.r.l.**. La "Suite Ambiente" è un insieme coordinato di soluzioni informatiche destinato alle Pubbliche Amministrazioni ma pensate, realizzate e testate per tutti i soggetti, pubblici e privati, che impattano con le problematiche ambientali.

L'obiettivo della Suite è quindi quello di "mettere in rete" tutti i protagonisti dei processi ambientali:

- **Le imprese** coinvolte, direttamente o tramite consulenti specializzati, nel caricamento diretto delle domande di autorizzazione e/o dei dati relativi agli autonomi controlli. Imprese che possono ora verificare lo stato delle proprie pratiche, con funzioni di avviso automatico delle scadenze che le riguardano;
- **I cittadini e tutti i portatori di interesse** che la legge impone siano informati sullo stato

dell'ambiente e sui processi autorizzativi fin dal primo momento di apertura di un procedimento.

- **Le Pubbliche Amministrazioni Locali -** Regione, Provincia, Comune, ARPA, ASL - per uno scambio dati finalizzato al consolidamento dei catasti autorizzativi ambientali e per una rapida circolazione delle informazioni richieste (pareri, esame delle autorizzazioni concesse).

5.1 Integrazione di un' istanza SIAM

Per testare quanto realizzato in questa Tesi è stato preso come caso d'uso il database del SIAM in uso presso l'Ufficio Ambiente della Provincia di Ancona. Si tratta di un database localizzato su un server Windows sul quale è installato l' RDMBS PostgreSQL. Il database conta 90 tabelle per un totale di 1711 attributi. E' una versione ridotta dello schema normalmente utilizzato dal SIAM in quanto viene utilizzato dall'Ufficio Acqua del Settore Ambiente della Provincia e presenta al suo interno tutto quanto concerne la gestione dei dati tecnici relativi a questa Matrice e le tabelle necessarie al normale funzionamento della Web Application. Il costante uso del SIAM presso l'Ufficio Ambiente di Ancona consente comunque di avere un ambiente di prova di discrete dimensioni e quindi pienamente soddisfacente per l'esecuzione di diverse tipologie di test.

E' importante premettere che i test effettuati hanno avuto come obiettivo la verifica sulle nuove funzionalità inserite in MOMIS e sulle modifiche apportate a funzioni già presenti.

I test effettuati sull'ambiente di prova hanno comportato i seguenti step:

- Interfacciamento ad un server PostgreSQL mediante il Wrapper PostgreSQL;
- Annotazione automatica del database mediante gli algoritmi WNFS e WDN;
- Completamento dell'annotazione e generazione della GVV;
- Creazione e popolamento dello schema,intero o parziale, mediante DataRiver;

5.1.1 Interfacciamento ad un server PostgreSQL

In seguito ad alcune direttive da parte del cliente il server della provincia di Ancona è raggiungibile solamente tramite collegamento in Desktop Remoto e inoltre non è consentito l'accesso diretto al Database. Non avendo quindi la possibilità di collegarsi direttamente alla

sorgente tramite MOMIS si è provveduto ad eseguire un *dump* del database del SIAM, allineato con l'ultimo backup di febbraio 2010, e di effettuare i test su un server PostgreSQL locale previo ripristino del database in oggetto.

Durante l'importazione delle sorgenti in MOMIS ci si è preoccupati di eseguire i test sulle diverse funzionalità implementate nel Wrapper, compresa la visualizzazione dei database di un server e la scelta della lingua di Annotazione in fase di importazione.

Essendo l'implementazione di un Wrapper una parte ormai "standard" all'interno degli sviluppi riguardanti MOMIS il test ha dato da subito risultati totalmente positivi che consentiranno quindi un'immediata integrazione nelle diverse versioni del progetto.

5.1.2 Annotazione del database del SIAM

Un database con 1711 attributi ha rappresentato un valido banco di prova per le modifiche apportate ai metodi di Annotazione, sia Automatica che Manuale. La procedura di Annotazione Automatica assume inoltre maggior rilievo proprio a causa dell'elevato numero di attributi e sottolinea il ruolo fondamentale svolto da questa procedura all'interno di un *data integration system*. Sarebbe oneroso, se non impensabile, provvedere ad un' annotazione esclusivamente manuale dell'intero schema.

Con la procedura di annotazione automatica è stato possibile poter testare le modifiche apportate a MOMIS riguardo gli algoritmi **WordNet First Sense Algorithm** e **WordNet Domains Algorithm**.

Con la procedura di annotazione manuale si è invece avuta la possibilità di testare tutte le modifiche fatte per una gestione multilingua di questa fase di annotazione.

Inoltre, in entrambe le procedure, si è presentata l'occasione di poter testare la nuova funzione di *stemming*, descritta e implementata nel capitolo 3.8.6.

Test 1: Annotazione del database SIAM mediante MultiWordNet: ITALIANO

Caricata una sorgente "**SIAM_TEST_1**" con AnnotationLanguage = "**ITALIANO**", si è provveduto all'Annotazione mediante l'esecuzione degli algoritmi WNFS e WND, con modalità di esecuzione **PIPE**. Il risultato del test presenta **1036** Lemmi Annotati su un totale di **1711**, con una

percentuale di annotazione pari quindi al **60,5 %** e una probabilità di annotazione del **57 %.**

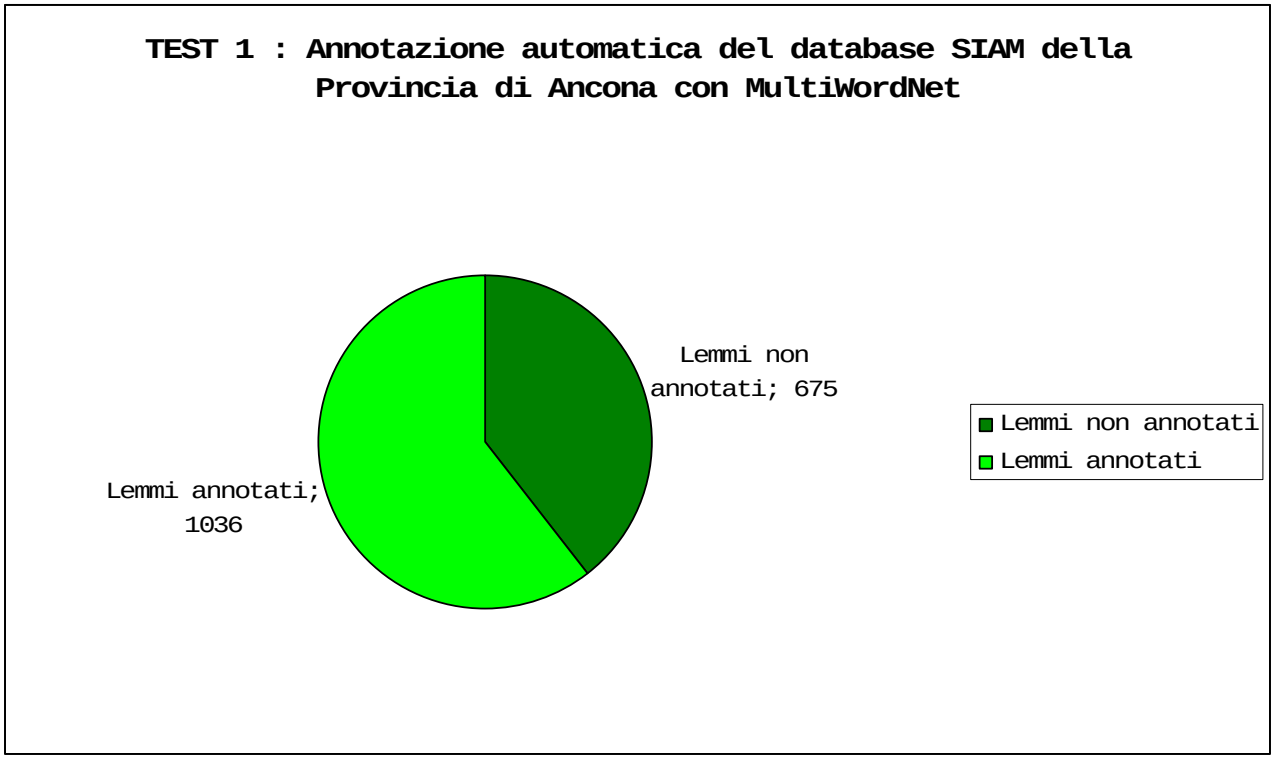

Figura 28 Annotazione del SIAM con AnnotationLanguage = ITALIANO

Test 2: Annotazione del database SIAM mediante MultiWordNet: ENGLISH

Caricata una sorgente "**SIAM_TEST_2**" con AnnotationLanguage = "**ENGLISH**", si è provveduto all'Annotazione mediante l'esecuzione degli algoritmi WNFS e WND, con modalità di esecuzione **PIPE**. Il risultato del test presenta **788 Lemmi Annotati** su un totale di **1711**, con un una percentuale di annotazione pari quindi al **46,5 %** e una probabilità di annotazione del **64 %**.

Il test è stato effettuato in primo luogo per fare un raffronto con il test precedente e in secondo luogo per verificare la coerenza del funzionamento della procedura di Annotazione rispetto alla versione originale presente in MOMIS. Da una prima analisi potrebbe sorgere un quesito: come mai la somma dei Lemmi annotati nei due test non dà come risultato il numero dei Lemmi presenti effettivamente?

Le cause di questa "anomalia" sono diverse. Innanzitutto lo schema del SIAM, nonostante sia il risultato di diversi anni di sviluppo non è mai stato dotato di uno standard, per cui è possibile trovare sia termini in lingua inglese (ad esempio: *id*, *counter*, *value*..), relativi nella maggior parte dei casi a un linguaggio informatico, sia termini in lingua italiana (ad esempio: *atto*,

pratica,documento, pozzo, impianto..), relativi invece a termini comunemente usati nella Pubblica Amministrazione e nel linguaggio comune.

Inoltre vi sono termini in lingua Inglese che vengono utilizzati nel linguaggio comune e che ormai sono entrati di diritto nella Lingua Italiana (ad esempio: *mail, fax, ok, ...*). Questi termini sono presenti quindi in entrambe le lingue presenti in MultiWordNet e le procedure di Annotazione Automatica li rilevano correttamente nei due diversi casi.

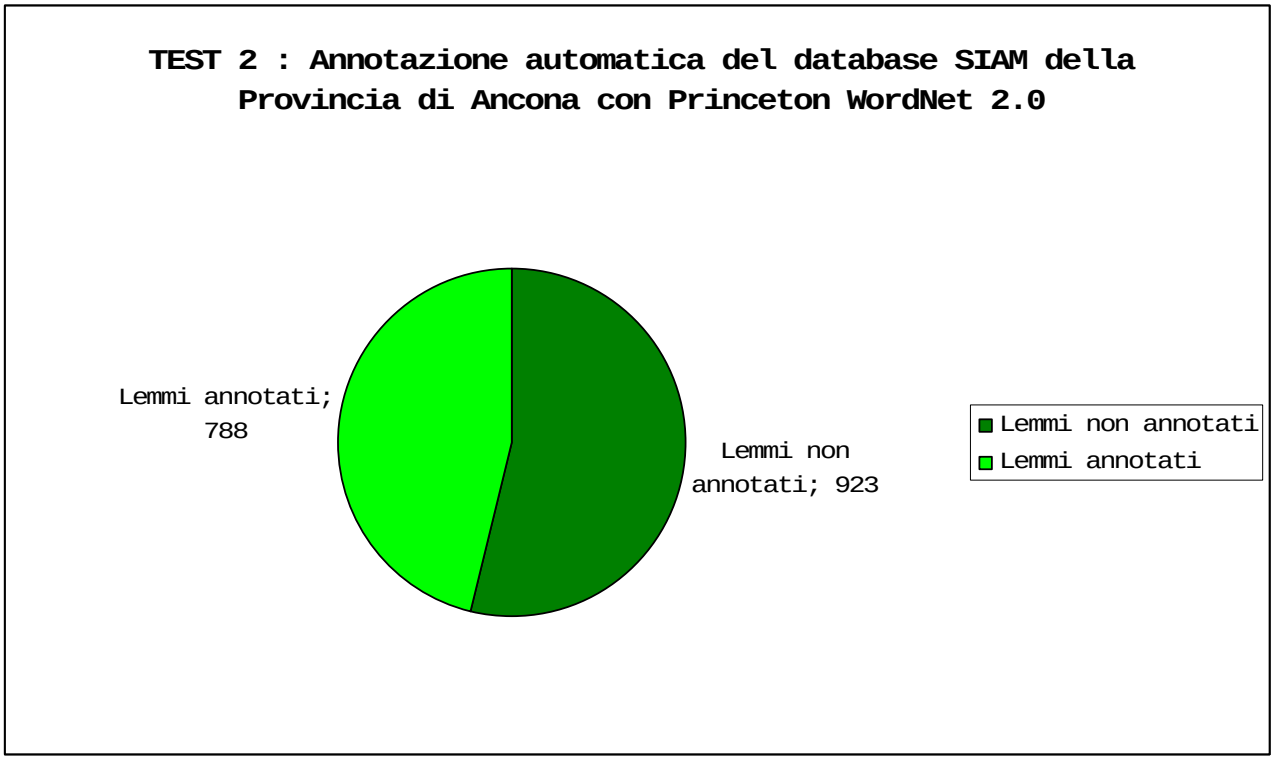

Figura 29 Annotazione del SIAM con AnnotationLanguage = ENGLISH

Test 3: Annotazione del database SIAM mediante MultiWordNet: ITALIANO e ENGLISH

Caricata una sorgente "**SIAM_TEST_3**",senza specificarne l'AnnotationLanguage, si è provveduto all'Annotazione mediante l'esecuzione degli algoritmi WNFS e WND, con modalità di esecuzione **PIPE**. Il risultato del test presenta **1069 Lemmi Annotati** su un totale di **1711**, con un una percentuale di annotazione pari quindi al **62 %** e una probabilità di annotazione del **58 %**.

In questo test viene posto in evidenza quanto evidenziato nel test precedente, ovvero sia il fatto che all'interno dello schema SIAM sono presenti termini in entrambe le lingue, sia il fatto che all'interno dei due database lessicali sono presenti dei termini comuni. Il test migliora le prestazioni di quanto ottenuto nel Test 1 proprio sulla base delle precedenti considerazioni. Inoltre è servito per

verificare la coerenza delle funzionalità già presenti.

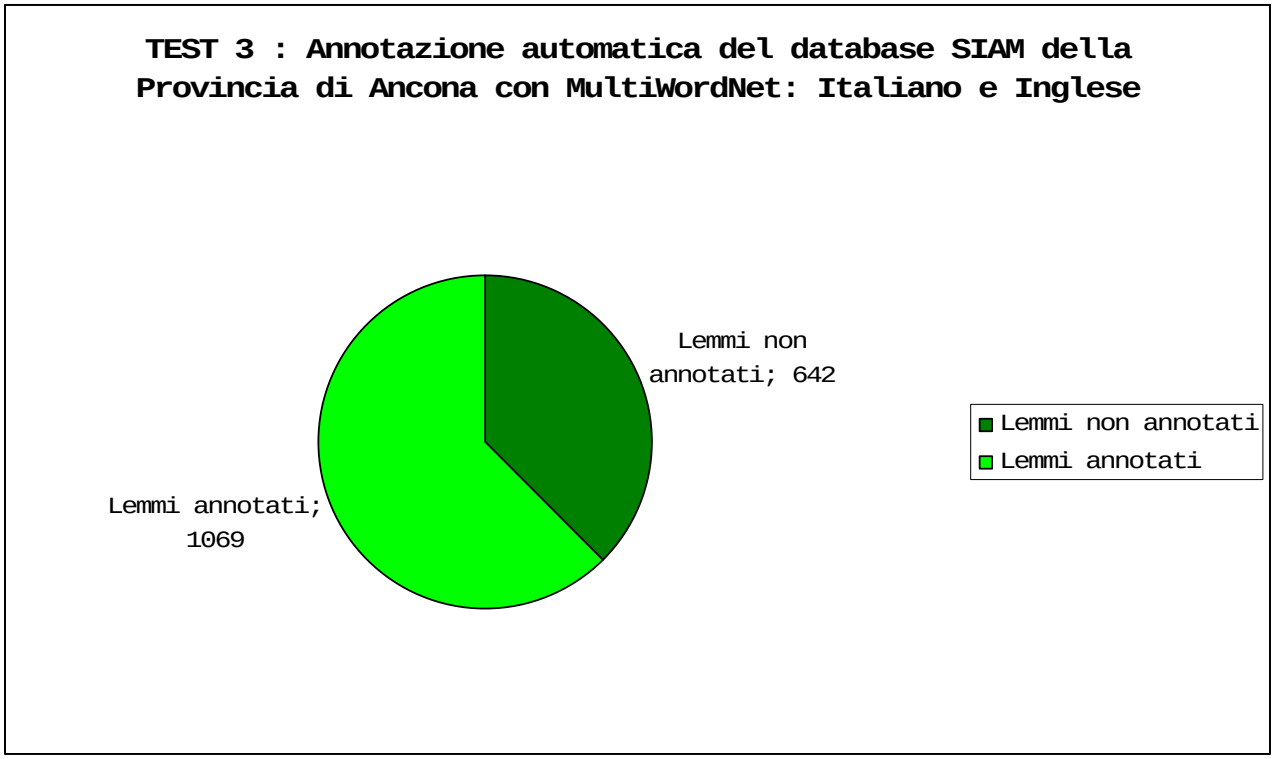

Figura 30 Annotazione del SIAM senza AnnotationLanguage

Considerazioni

	ITALIANO	ENGLISH	NO ANNOTATION LANGUAGE
Lemmi Totali	1711	1711	1711
Lemmi Annotati	1036	788	1069
Percentuale di Annotazione (%)	60,5	46	62
Probabilità di Annotazione (%)	57	64	58

Tabella 4 Riepilogo risultati dei test sulle annotazioni

Tenendo conto della presenza prevalente termini in lingua italiana all'interno del database usato come caso di studio e della numerosa presenza di termini composti (ad esempio: l'attributo "**FLAG_ART13_91_271_CEE**" della tabella **AC_SCARICOIDRICO**), sulla base dei risultati dei test effettuati, è possibile notare un funzionamento più che soddisfacente delle funzioni di Annotazione Automatica sia per la lingua Inglese e in particolar modo per la lingua Italiana. In

quest'ultimo caso viene infatti annotato, nel peggiore dei casi, oltre il 60 % dei termini presenti. Nonostante questa percentuale sia largamente influenzata dal numero di attributi dello schema, si è comunque ancora lontani da un risultato che consenta di ritenere conclusi gli sviluppi riguardanti l'integrazione di sorgenti multilingua, ma è senz'altro possibile utilizzare quanto svolto come base di partenza per studi futuri. In tabella 4 si riporta un riepilogo riassuntivo dei risultati dei test.

Merita un discorso a parte la nuova funzionalità di *stemming* per la lingua Italiana. Come descritto nel capitolo 3.8.6, l'algoritmo di stemming implementato si basa su una versione "italiana" dell'algortimo di Porter, il quale mira a ricondurre un generico termine alla sua radice e quindi alla sua forma base. Se da un lato la radice dei termini in lingua inglese corrisponde nella maggior parte dei casi alla forma base stessa, non è possibile affermare la stessa cosa per la lingua italiana.

Infatti, prendendo come esempio la parola inglese "*cats*", plurale di "*cat*", l'algoritmo di Porter ricava con facilità la radice, ovvero "*cat-*". In questo caso la radice è uguale alla forma base ed è quindi semplice reperire il termine in un dizionario, o in questo caso, in un database lessicale. D'altra parte la grammatica inglese presenta una serie non elevata di casi riguardanti le forme flesse ed è quindi semplice ricondursi in ogni caso alla forma base.

Se però si prende in considerazione la parola italiana "*gatti*", plurale di "*gatto*", l'algoritmo di Porter implementato per la lingua italiana ricava correttamente la radice "*gatt-*", la quale però non corrisponde alla sua forma base. Di conseguenza per poter reperire con sicurezza il termine su un dizionario mancherebbe una funzionalità che dalla radice di una parola ricavi la sua forma base, prevedendo le numerose casistiche delle forme flesse della lingua italiana. Non essendo uno degli obiettivi di questa Tesi si è provveduto ad utilizzare una soluzione di comodo per eseguire i test. Partendo da ciò che l'algoritmo produce in uscita, che risulta comunque essere corretto, si provvede a fare una ricerca sul dizionario del *lemma* corrispondente alla radice. Qualora esista non si ha la necessità di proseguire oltre e il termine viene correttamente recuperato sul database lessicale MultiWordNet. Qualora invece non esista si provvede alla ricerca di tutti i *lemma* che iniziano con tale radice, scegliendo infine quello con lunghezza più corta, partendo da un presupposto volutamente "forzato" che il *lemma* corrispondente sia la parola più simile alla radice stessa. Tale forzatura presenta vantaggi e svantaggi. Infatti all'interno dello schema viene correttamente "*stemmato*" il termine "*richiedenti*", riconducendolo alla sua forma base "*richiedente*", sulla base della radice "*richiedent-*". D'altra parte però il termine "*fogli*" non viene correttamente "*stemmato*", in quanto la procedura identifica nel *lemma* "*foglia*" la forma base della radice "*fogl-*", piuttosto che ricondursi al *lemma* "*foglio*".

5.1.3 Creazione e popolamento di uno schema.

In questa serie di test vengono provate le nuove funzionalità messe a disposizione nella versione **DataRiver** di MOMIS, riguardanti la creazione e il popolamento di uno schema a partire da una GVV generata precedentemente. Per l'esecuzione dei test si è quindi preso in considerazione lo schema annotato nel Test 3 del precedente paragrafo e si è proceduto con la generazione della GVV tramite MOMIS. Essendo presente una sola sorgente il processo di integrazione non viene utilizzato in tutte le sue parti e la GVV generata è una rappresentazione fedele dello schema locale del SIAM. D'altronde, ciò che si propone in questa Tesi è innanzitutto realizzare un accoppiamento fra i due programmi e i test devono in primo luogo verificare la creazione e il popolamento di un database del tutto simile alla sorgente locale integrata.

Si è proceduto con 4 tipologie di test :

- Creazione parziale dello schema;
- Popolamento parziale dello schema;
- Creazione dell'intero schema;
- Popolamento dell'intero schema;

I primi due test hanno avuto come scopo quello di verificare la correttezza dei metodi **`createPartialSchema()`** e **`populatePartialSchema()`** per la creazione / popolazione di tabelle mediante un Set di GlobalClasses. Le tabelle prese in esame sono **`AC_DEMANIOIDRICO`**, **`SINTRA_TE_ISTRUTTORIA`** e **`AC_CAMPSCAR`**. La scelta non è casuale poiché con queste 3 tabelle si riesce a testare un insieme abbastanza ampio di casi riguardo i tipi di dato degli attributi. Bisogna infatti sottolineare come si è dovuta porre particolare attenzione al mapping fra i tipi di dato ODL_I^3 e i tipi di dato JDBC. Solitamente questo mapping viene usato dai Wrapper in fase di creazione di una sorgente per stabilire che tipo di dato ODL_I^3 associare ad un attributo locale. Essendo ODL_I^3 un linguaggio descrittivo non era sinora stato approfondito tutto ciò che riguarda la definizione della dimensione di certi tipi di dato. Ad esempio, la gestione delle cifre decimali dei tipi di dato numerici o la dimensione di certi tipi di dato, come i **`BLOB`** e i **`CLOB`**. Quanto descritto nel Capitolo 4 spiega come questo problema sia stato risolto mediante l'estensione di alcuni tipi di dato ODL_I^3.

La creazione delle tabelle selezionate si è conclusa positivamente. Sul *target database* sono infatti state generate correttamente e un rapido controllo ha permesso di verificare l'intera struttura: nomi delle tabelle, tipo di dato degli attributi e quanto ne consegue, Primary Key della tabella.

Anche il popolamento delle tre tabelle selezionate precedentemente non ha dato particolari problemi, se non un'opportuna gestione dei campi `CLOB` in fase di memorizzazione. L'RDBMS PostgreSQL infatti effettua un controllo sulla dimensione massima di alcuni tipi di dato e in questo caso si è dovuto provvedere al ridimensionamento di alcuni dati, pur senza perderne la coerenza.

Le ultime due tipologie di test sono invece servite sia per testare le funzionalità riguardanti i metodi **`createSchema()`** e **`populateSchema(),`** sia per rilevare le prestazioni del programma durante l'intera procedura. Si è trattano infatti della creazione automatica e del popolamento di 90 tabelle comprendenti 1711 attributi e risulta essere uno "*stress test*" di discreta rilevanza.

Durante il test il programma non ha risentito di particolari problemi prestazionali se non durante la creazione e il popolamento della tabella **`AC_POZZOIDRICO`**, che presenta 114 attributi e un numero di record pari a 28417. Altra piccola nota riguarda invece i tipi di dato **`BINARY`** (caso sfortunato: un caso su 1711!) sfuggito al caso di test precedente e durante il mapping dei tipi di dato JDBC nello sviluppo del Wrapper PostgreSQL. Sinora non erano stati gestiti né in MOMIS, né in DataRiver ed è stato necessario risolvere questo caso prima di poter proseguire. Per modellare il tipo di dato `BINARY` è stato scelto di utilizzare il tipo di dato ODL_I^3 `StringType`. La soluzione scelta si è rivelata corretta in quanto si è potuto analizzare correttamente quella particolare tipologia di dato e provvedere alla creazione automatica del relativo attributo e il suo seguente popolamento.

Anche il risultato di questi test è stato pienamente soddisfacente essendo state create e popolate correttamente 86 tabelle su 90 GlobalClass di partenza, oltre il 95 % dei casi. Una successiva analisi sulle 4 tabelle per le quali la procedura ha dato esito negativo ha evidenziato come vi siano stati problemi durante la creazione per un errato reperimento del parametro *scale* di alcuni attributi numerici. Una semplice correzione ha permesso di poter rilanciare la procedura solamente per questo Set di GlobalClass, testando ulteriormente quanto già sviluppato.

Conclusioni e sviluppi futuri

In questi Tesi si è proposto l'accoppiamento fra il *data integration system* MOMIS e SIAM, un' applicativo web utilizzato dall'Ufficio Ambiente di diverse Province italiane. Tale accoppiamento è stato realizzato avendo come obiettivo lo sviluppo di alcune nuove funzionalità per il progetto MOMIS, che hanno riguardato diverse sezioni del processo di integrazione. L'accoppiamento dei due programmi dipendeva dal conseguimento di 3 obiettivi:

- Realizzazione di un Wrapper per sorgenti PostgreSQL;
- Integrazione di un database lessicale multilingua;
- Materializzazione della GVV;

In primo luogo si è sviluppato con successo un Wrapper per sorgenti localizzate su un RDBMS PostgreSQL. Sulla base degli sviluppi dei Wrapper già esistenti è stata creata un'interfaccia grafica e un modulo per la connessione e il reperimento delle sorgenti locali e per consentire la creazione di una sorgente ODL_I^3. Non ha presentato particolari problemi in fase di sviluppo ed è stato testato con risultati completamente soddisfacenti.

L'accoppiamento fra i due programmi ha posto in evidenza un fattore che sinora era stato solo argomento di ricerca, ovvero la possibilità di poter annotare le sorgenti locali mediante un database lessicale multilingua, nello specifico relativo alla lingua inglese e alla lingua italiana. Previo uno studio su due database lessicali oggetto di ricerca di diversi istituti di ricerca europei, MultiWordNet ed EuroWordNet, si è proceduto con la scelta ponderata di uno dei due database per l'integrazione con il database lessicale presente, Princeton WordNet 2.0. Scelto MultiWordNet come risorsa lessicale da integrare a causa dei vantaggi forniti dal modello *expand model*, l'intera procedura di integrazione ha comportato come primo step un aggiornamento dei record contenti in esso, in quanto allineati ad una versione obsoleta di Princeton WordNet, la 1.6, e di conseguenza non totalmente coerenti. In seguito si è potuto procedere con l'integrazione vera e propria, provvedendo a importare i dati secondo una logica che parte dal presupposto di poter sempre stabilire all'interno del database lessicale se un dato record, sia esso un *synset*, un *lemma* o una relazione, appartiene alla lingua inglese o alla lingua italiana. Come ultimo passo si è provveduto a inserire in MOMIS la logica che consente all'utente di poter stabilire quale lingua utilizzare durante il processo di integrazione. Queste modifiche hanno riguardato sia diverse sezione dei tool SI-Designer, WordNet Editor e delle interfacce dei Wrapper, sia ampie sezioni del "Backend" di MOMIS per ciò che riguarda la gestione interna di diverse lingue e la creazione dello Schema

Globale.

In stretta relazione con l'integrazione di un database lessicale multilingua si è collocato un altro degli sviluppi affrontati in questa Tesi. Poiché MOMIS consente un' annotazione semi – automatica delle sorgenti all'interno del processo di integrazione, si è provveduto ad apportare alcune variazioni a diverse funzionalità già presenti. In particolare sono stati modificati due algoritmi riguardanti l'Annotazione Automatica ed è stato creato uno *stemmer* per la lingua italiana, sulla base di un algoritmo per la lingua inglese già presente in MOMIS. Gli algoritmi modificati sono il WordNet First Sense Algorithm e il WordNet Domains Algorithm. Per lo *stemmer* si è fatto riferimento ad un progetto che si occupa dell'implementazione di algoritmi in diverse lingue sulla base del noto Porter Algorithm Stemmer. Inoltre, di pari passo con la creazione del nuovo *stemmer* si è proceduto con un totale refactoring del package e delle classi che si occupano di questa funzionalità. Come riportato nel capitolo 5 i risultati dei test effettuati sulla base di queste modifiche possono ritenersi soddisfacenti per ciò che concerne le procedure di annotazione automatica e manuale, mentre non completamente soddisfacenti per ciò che concerne l'algoritmo di stemmer per la lingua italiana.

Quanto fatto può essere considerato come una buona base di partenza per future implementazioni. Infatti, con la possibilità di avere un database multilingua in MOMIS si aprono una serie di scenari riguardo le scelte di progetto e gli sviluppi futuri per l'integrazione di sorgenti eterogenee di diverse lingue. Nella corso della Tesi si sono proposte alcune soluzioni sulla base di assunzioni atte a verificare il caso di studio preso in considerazione. Sono quindi soluzioni specifiche del caso, non definitive, e non viene preclusa la possibilità di considerare casi d'uso differenti e procedere con una differente implementazione di quanto svolto.

Infine, per completare l'accoppiamento si è provveduto alla realizzazione di una funzione di materializzazione dello Schema Globale mediante l'utilizzo della versione open-source di MOMIS: "DataRiver". In seguito all'importazione del Wrapper implementato nel capitolo 2, e sulla base delle stesse interfacce grafiche per la connessione ad un server presenti in DataRiver, è stato realizzato un modulo che consente all'utente la connessione ad un server di destinazione sul quale materializzare sia l'intero Schema Globale ricavato dalla GVV, sia un set di GlobalClasses opportunamente scelte. In parallelo con questi sviluppi è stata realizzata una funzione che a partire dall'intero Schema Globale, o da un Set di GlobalClasses, consente la materializzazione di tutti i record ricavati con il Processo di Integrazione in un database di destinazione. Anche in questo caso si crea un insieme di possibilità di sviluppo parecchio ampio. La creazione di uno schema, il popolamento dello stesso e, non meno importante per un'applicazione come il SIAM, l'aggiornamento dei dati ricavati da una GVV non sono fattori per i quali c'è una rapida soluzione.

Come descritto nel capitolo 4, lasciare ad un *data integration system* come MOMIS il compito di svolgere operazioni che solitamente sono in dotazione ai vari client SQL comporta una serie di casi per i quali servirebbe sicuramente uno studio approfondito. Anche in questo caso quanto svolto in questa parte di Tesi ha comportato alcune scelte di progetto effettuate sulla base del caso di studio in oggetto e non costituisce un vincolo su futuri sviluppi.

Nel complesso l'accoppiamento fra i due programmi non può dirsi concluso in quanto dei 3 macro-obiettivi che ci si è posti l'ultimo non risulta ancora pienamente soddisfacente. Si è provveduto infatti con successo alla creazione e al popolamento di un database tramite la GVV. Non è possibile tuttavia ritenere questo risultato come definitivo in quanto sarebbe necessario verificare innanzitutto la coerenza dei dati contenuti rispetto quelli originali e interfacciare il database con tutte le Web Application da cui il SIAM dipende. Inoltre sarebbe opportuno realizzare delle funzioni che consentano un aggiornamento costante dello schema del database e dei dati in esso contenuti. Infine sarebbe un interessante caso di studio poter generare una GVV tramite l'integrazione del database del SIAM con sorgenti aventi schemi differenti, si pensi ad esempio ad un Registro delle Imprese Provinciale, e in seguito procedere con la materializzazione di tale GVV. Si avrebbe una versione del database del SIAM ampliata, contenente non solo i dati propri ma anche i dati integrati. Tale operazione sarebbe un enorme valore aggiunto durante le frequenti importazioni di dati appartenenti a terze parti, finora effettuate mediante delle procedure spesso lunghe e onerose a carico degli sviluppatori.

Bibliografia

[1] Gio Wiederhold et al. Intergrating artificial intelligence and database technology. Journal of Intelligent Integration System, 2/3 Giugno 1996

[2] Data integration involves combining data residing in different sources and providing users with a unified view of these data. Maurizio Lenzerini (2002). "Data Integration: A Theoretical Perspective". PODS 2002. pp. 233–246

[3] R. Hull and R. King et al. ARPA I3 reference architecture, 1995. http://www.isse.gmu.edu/I3 Arch/index.html

[4] 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

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

[6] D.Calvanese, S.Castano, F.Guerra, M.Vincini et al. Towards a Comprehensive Methodological Framework for Semantic Integration of Heterogeneous Data Sources.

[7] S. Bergamaschi, D.Beneventano, S. Castano and M. Vincini. Integrazione di informazione: il linguaggio ODL_I^3 e la logica descrittiva OLCD. Tecnical report T3-R03, 16 Luglio 1998.

[8] J.Yang Introduction to OQL. 1999.

[9] http://java.sun.com/developer/onlineTraining/corba/corba.html#co1

[10] D.Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. Synthesizing an Integrated Ontology. *IEEE Internet Computing*, 7(5):42{51, 2003.

[11] S. Bergamaschi, S. Castano, and M. Vincini. Semantic Integration of Semistructured and Structured Data Sources. *SIGMOD Record*, 28(1):54{59, 1999

[12] From Aspect-Oriented Model to Implementation: Watch Out for Impedance Mismatch (2003) Johan Ovlinger.

[13] Evaluation of the Object--Relational DBMS Kim Nelson Rossiter

[14] Large Object Support in PostgreSQL 1993, Michael Stonebraker, Michael Olson

[15] PostgreSQL 2000 Bruce Momjian Ch. 17 - 18 – 19

[16] “SI-Designer: un tool per l’integrazione di sorgenti distribuite ed eterogenee.” Tesi di Laurea di Rossano Guidetti 1999/2000

[17] A. G. Miller. WordNet: A lexical database for english. Communications of the ACM, 38(11):39 - 41, 1995. http://www.cogsci.princeton.edu/~wn/

[18] Vossen P. (ed.). EuroWordNet, General Document. E’ disponibile all’indirizzo
http://www.illc.uva.nl/EuroWordNet/docs.html (Deliverable D032D033/2D014).

[19] “Ontologie Lessicali Multilingua: MultiWordNet ed EuroWordNet” Tesi di Laurea di Roberto Rasi A.A. 2002 - 2003

[20] “Intelligent Information Integration systems: extending lexicon ontology” A.A 2001-2002 Tesi di Laurea di Veronica Guidetti

[21] B.Magnini, C.Strapparava, G.Pezzullo e A.Ghiozzo. Comparing Ontology-based and Corpus-Based Domain Annotation in WordNet. In Proceeding of first International WordNet Conference, 2002

[22] B.Magnini, S.Strapparava et. al.. The role of domain Information in Word Sense Disambiguation. Natural Language Engineering, 25 luglio 2002.

[23] E. Pianta, L. Bentivogli, C. Girardi. MultiWordNet, developing an aligned multilingual database. Proceedings of the First International Conference on Global WordNet, Mysore, India, January 21-25, 2002. http://multiWordNet.itc.it/english/publications.html

[24] Pubblicazioni Rada Mihalcea: http://www.cse.unt.edu/~rada/papers.html

[25] "Metodi di disambiguazione del testo ed estensioni di WordNet nel sistema MOMIS" Tesi di Laurea di Serena Sorrentino A.A. 2005 – 2006

[26] Bergamaschi S., Po L., Sorrentino S., Automatic annotation for mappings discovery in data integration systems, In Proceedings of the Sixteenth Italian Symposium on Advanced Database Systems, SEBD 2008, 22-25 June 2008, Mondello, PA, Italy.

[27] M.F. Porter, 1980, An algorithm for suffix stripping, *Program*, pp 130−137.

[28] "Datariver: un sistema di integrazione dati Open Source" Tesi di laurea di Francesco Nigro A.A. 2006 – 2007

Ringraziamenti

Ringrazio la Professoressa Sonia Bergamaschi per l'opportunità di studio e di ricerca fornitami e i consigli durante la realizzazione di questa Tesi.

Rivolgo un sincero e sentito ringraziamento agli ingegneri Mirko Orsini e Laura Po. Grazie per la pazienza(grande) e la disponibilità mostrata in ogni occasione e per i preziosi aiuti durante gli sviluppi.

Un ringraziamento va anche a Quix s.r.l., in particolar modo a Claudio Masini, per l'occasione di ricerca proposta, e a Simo, Angelina, Marco, Emanuela, Daniele e ai Mattia per il sostegno, tecnico ma soprattutto Morale, indispensabile di questi mesi.

E senza dimenticare che "nessuno cade se non ci stringiamo forte le mani" non posso escludere da questi ringraziamenti la Grande Famiglia che mi accompagna nei weekend in giro per l'Italia un grazie a:

> *Zannina e Nunzio, Giouà e Tizziolina, Akka, Bubu, Teo e la Vale, al Gabbo, a quei "tòpi" dei Modena City Ramblers per avermi fatto studiare nei locali più malfamati di Italia, Uazza per il significato intrinseco di una laurea, Danito e l'Abesipage, Guido Foddis e la grandissima Polisportiva Italia Gangbang, Luca Lanzi e la Casa del Vento, a Matley e alla Cagnina, Faseux lo spagnolo, a tutti coloro che mi sto sicuramente dimenticando.*

Grazie per essere continua fonte di ispirazione a : Amelie Poulain, Yann Tiersen, la Guinness, Foer, Bukowski e Benni, De Andrè e Dave Matthews Band, Eugene Hutz e a "Santa Marinella" nelle notti passate a sviluppare, tin wisthles e fise, Einaudi, Renoir e Monet.

E infine un grazie a tutta la mia famiglia, quella vera, per la costante vicinanza nonostante la reale distanza.

A mio Padre che rimarrà sempre il mio costante punto di riferimento.

A mia Madre perché non senza di Lei non sarei mai arrivato da nessuna parte.

A mia Sorella per la nostra piccola ma indispensabile complicità.

E infine a Wendy. Per Tutto e per Noi.