Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica – *Nuovo Ordinamento*

Progetto MOMIS: i wrapper DB2/ODLi3 e Oracle/ODLi3

Relatore:
Chiar.mo Prof. Sonia Bergamaschi

Candidato:
Verardi Mauro

Anno Accademico 2004-2005

RINGRAZIAMENTI

Ringrazio la Professoressa Sonia Bergamaschi per la grande disponibilità dimostrata durante la realizzazione della presente tesi.
Rivolgo un sincero ringraziamento all'Ing. Mirko Orsini per l'aiuto fornito e per i preziosi consigli.
Vorrei ringraziare anche la mia famiglia e miei amici che mi hanno sostenuto durante questo percorso di studi.

Un ringraziamento particolare ed Emanuela per aver sopportato noiosi discorsi di ambito informatico per aiutarmi nella stesura della tesi.

INDICE

Elenco delle figure

Appendici

Introduzione

Negli ultimi anni abbiamo assistito ad una crescita esponenziale del numero di fonti di informazione nelle aziende o su internet che rende possibile l'accesso ad un vastissimo insieme di dati distribuiti su macchine diverse. Contemporaneamente alla probabilità di trovare l'informazione desiderata sulla rete informatica, va aumentando la difficoltà di recuperarla in tempi e modi accettabili, essendo le fonti informative tra loro fortemente eterogenee per quanto riguarda i tipi di dati o il modo in cui sono descritte e presentate ai potenziali utenti. Si va così incontro al problema dell'*information overload* (sovraccarico di informazioni): il numero crescente di informazioni (e magari la loro replicazione) genera confusione, rendendo pressoché difficile isolare efficientemente i dati necessari a prendere determinate decisioni. Il reperimento delle informazioni desiderate, distribuite su sorgenti diverse, richiede inoltre competenze nei contenuti, le strutture e i linguaggi di interrogazione propri di queste risorse separate. L'utente deve quindi scomporre la propria interrogazione in sottointerrogazioni dirette alle varie sorgenti informative, e deve poi trattare i risultati parziali in modo da ottenere una risposta unificata, operazione non semplice se si considerano le trasformazioni che devono subire questi dati, le relazioni che li legano, le analogie o le diversità. Con un numero sempre maggiore di sorgenti a disposizione e dati da manipolare è difficile che l'utente medio abbia competenze necessarie a portare a termine compiti di questo tipo, diviene quindi indispensabile un processo di automazione a partire dalla fase di reperimento alla fase di integrazione.

Come descritto in [1] le maggiori difficoltà che si incontrano nell'integrazione di sorgenti diverse consistono principalmente in eterogeneità strutturali e implementative (basti pensare a differenze in piattaforme hardware, DBMS, modelli di dati e linguaggi di dati) e alla mancanza di una ontologia comune che porta ad una eterogeneità semantica. Tale eterogeneità semantica è dovuta sostanzialmente a:

- ***Differenti terminologie***: nomi differenti per rappresentare gli stessi concetti in sorgenti diverse;
- ***Differenti rappresentazioni strutturali***: utililizzo di modelli diversi per rappresentare informazioni simili.

In questo scenario, fortemente studiato e che coinvolge diverse aree di ricerca, si vanno ad inserire strumenti quali i DSS (Decision Support System), l'integrazione di basi di dati eterogenee, i *datawarehouse*.

L'integrazione di basi di dati, insieme ai *datawarehouse*, si occupano di presentare all'utente delle viste, cioè delle porzioni delle sorgenti, replicandone i contenuti, affidandosi a complicati algoritmi per assicurare la loro consistenza a fronte di cambiamenti nelle sorgenti originali.

Con *Integration of information*, si rappresentano invece tutti quei sistemi che combinano tra loro dati provenienti da sorgenti (o parti di sorgenti) senza le replicazione fisica dei dati, ma basandosi solo sulla loro descrizione. Se inoltre si fa uso di tecniche di intelligenza artificiale, sfruttando le conoscenza già acquisite, si può parlare di *Intelligent Integration of Information* (progetto I^3 [9] dell'ARPA), che si distingue dagli altri modi di integrazione nel fatto che aggiunge valore alla conoscenza acquisita, ottenendo nuove informazioni dai dati ricevuti.

Questa tesi si inserisce all'interno di un progetto più ampio denominato **MOMIS** [6] (**M**ediator Envir**O**nment for **M**ultiple **I**nformation **S**ources), una ricerca condotta dal Dipartimento di Scienze dell'Ingegneria dell'Università di Modena e Reggio Emilia e dal dipartimento dell'informazione dell'Università di Milano, sviluppato allo scopo di realizzare un tool semiautomatico per l'integrazione di sorgenti dati eterogenee e distribuite, siano esse strutturate o semistrutturate. La particolarità di questo progetto sta nell'approccio di tipo semantico adottato per l'integrazione, basata ora non solo sulla struttura delle sorgenti ma anche sul significato che il progettista assegna ai singoli campi delle sorgenti stesse. MOMIS [6] adotta una struttura a quattro livelli:

- livello ***sorgenti***;
- livello ***wrapper***;
- livello ***mediatore***;
- livello ***utente***;

Il mediatore è la parte fondamentale del sistema, cioè il tool semiautomatico che realizza l'integrazione. Esso è formato da due macro blocchi:

- ***Global Schema Builder***
  modulo che si occupa di effettuare l'integrazione grazie all'utilizzo del tool grafico SI-Designer [7] generando un unico schema globale;
- ***Query manager***
  gestore delle interrogazioni che dalla singola interrogazione dell'utente compone e invia le query alle interfacce con le sorgenti, ricombinando poi i risultati.

L'interfacciamento delle sorgenti avviene a livello dati grazie a moduli chiamati *Wrapper* che presidiano le sorgenti, hanno il compito di:

- descrivere in un linguaggio comune la sorgente alla quale è connesso;
- permettere l'esecuzione di query locali e presentare i risultati al mediatore.

Scopo della presente tesi è la realizzazione di wrapper per sorgenti relazionali quali IBM DB2 Universal Database [10] e Oracle Database [11], entrambi accessibili tramite le API JDBC [12]. Ulteriore fine è lo sviluppo dei relativi GUI da inserire nel framework SI-Designer [7].

Il punto di partenza per il perseguimento di tali obiettivi è rappresentato dai wrapper già implementati in MOMIS [6] per sorgenti accessibili tramite JDBC [12] (il wrapper JDBC generico [4] e il wrapper per Microsoft SQL Server) e dal GUI del wrapper per Microsoft SQL Server.

La tesi è organizzata nel seguente modo:

Il **capitolo 1** spiega come i wrapper sono inseriti nell'architettura di riferimento I^3 [9] e ne vengono illustrati i servizi.

Il **capitolo 2** riporta una panoramica sugli strumenti utilizzati per la connessione alle sorgenti da integrare.

Nel **capitolo 3** viene illustrato il progetto MOMIS [6], la sua architettura e i Wrapper già implementati.

Il **capitolo 4** spiega come sono stati realizzati e testati i wrapper DB2/ODL_{I3} e Oracle/ODL_{I3}.

Nel **capitolo 5** sono illustrate le problematiche affronatate per la realizzare le GUI utente dei wrapper.

I Wrapper nei sistemi di integrazione di dati

1.1 Architettura di riferimento I^3

L'architettura di riferimento (Figura 1) presentata in questo paragrafo è stata tratta dal sito web [9], e rappresenta una sommaria categorizzazione dei principi e dei servizi che possono e devono essere usati nella realizzazione di un integratore *intelligente* di informazioni derivanti da fonti eterogenee:

- ***Servizi di coordinamento***: servizi di alto livello che permettono l'individuazione delle sorgenti di dati interessanti, ovvero che probabilmente possono dare risposta ad una determinata richiesta dell'utente;
- ***Servizi di amministrazione***: servizi usati dai servizi di coordinamento per localizzare le sorgenti utili, per determinare le loro capacità, per creare e interpretare TEMPLATE (strutture dati che descrivono i servizi, le fonti ed i moduli da utilizzare per portare a termine un determinato task);
- ***Servizi di integrazione e trasformazione semantica***: servizi che supportano le manipolazioni semantiche necessarie per l'integrazione e la trasformazione delle informazioni. Tra questi servizi si distinguono quelli relativi alla trasformazione degli schemi (*metadati*) e quelli relativi alla trasformazione dei dati, sono spesso indicati come *servizi di mediazione*;
- ***Servizi di wrapping***: (vedi paragrafo 1.2) sono i servizi che rendono le diverse sorgenti di informazione conformi ad uno standard interno o esterno;
- ***Servizi ausiliari***: aumentano la funzionalità degli altri servizi.

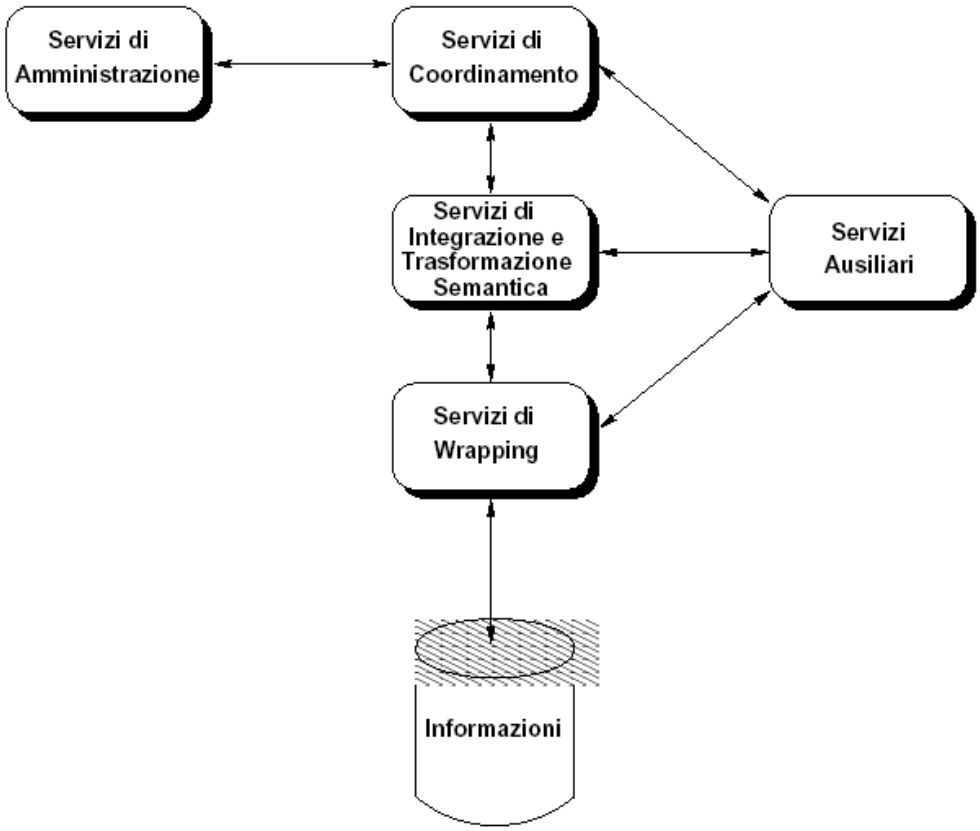


Figura 1: Architettura I^3

1.2 I servizi di Wrapping

I servizi di Wrapping [34] sono utilizzati per far aderire le fonti di informazioni ad uno standard per interfacciarsi con il mondo esterno. Si comportano quindi come traduttori dai sistemi locali ai servizi di alto livello dell'integratore e viceversa quando si interrogano le sorgenti di dato. In particolare si prefiggono due obiettivi:

- Permettere ai servizi di coordinamento e di mediazione di trattare le sorgenti locali in modo uniforme, anche se tali sorgenti non sono state concepite come facenti parte del sistema di integrazione;
- Essere il più riusabili possibile, devono quindi fornire interfacce che seguano gli standard più diffusi (ad esempio SQL come linguaggio di interrogazione di basi dati e CORBA [16,17] come protocollo di scambio oggetti). Questo renderebbe le sorgenti di questi wrapper accessibili dal maggior numero possibile di moduli mediatori.

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

I Servizi di Wrapping Primari includono Comunicazione, Ristrutturazione dei Dati e Comportamento. Riportiamo qui di seguito la gerarchia dei servizi di wrapping di alto livello, o primari, ed i loro sotto-servizi, unitamente ad una breve descrizione per ciascuno di essi.

W: Wrapping Services

W.1: Communication Wrapping Service

W.1.a: *Calling Interface*

questi servizi sono per la trasformazione dell'interfaccia delle chiamate tra due programmi scritti nel medesimo o in differenti linguaggi.

W.1.b: *Event Management*

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

W.1.c: *Method and Function Calling*

questi servizi gestiscono la trasformazione di chiamate di metodi e funzioni tra una sorgente di informazioni ed un servizio di più alto livello incompatibile.

W.2: Data Restructuring Wrapping

W.2.a: *Data Format Restructuring*

questi servizi sono per la traslazione tra differenti formati dei dati a livello macchina o file.

W.2.b: *Metadata Format Restructuring*

questi servizi sono per la trasformazione di formati metadata a livello macchina o file.

W.3: Behavioral Transformation Wrapping Service

W.3.a: *Application Program Modification*

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

W.3.b: *Protocol Modification*

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

W.3.c: *Data Language Transformation*

questi servizi permettono la traslazione tra differenti linguaggi di manipolazione dati, come tra SQL incompatibili.

I wrapper realizzati in questa tesi forniscono unicamente i servizi descritti al punto w.2, infatti il loro scopo principale è di descrivere in formato ODL_{I3} (paragrafo 3.2) elementi rappresentati in forma relazionale.

2 La tecnologia JDBC

La tecnologia JDBC [12] consiste in un insieme di API che permettono l'accesso ad ogni sorgente dati tabulare dal linguaggio di programmazione JAVA. Permette quindi non solo l'accesso ad una vasta gamma di database SQL ma anche ad altre sorgenti dati di tipo tabulare, come ad esempio i fogli elettronici. Il vantaggio di JDBC[12] rispetto ad altri approcci è che l'accesso alle sorgenti è di tipo virtuale ed avviene grazie alla Java Virtual Machine. In altre parole, con le API JDBC [12], non è necessario scrivere un'applicazione per il database Oracle, una per IBM DB2, una per SQL Server, ecc... Ogni programma che utilizzi questa tecnologia avrè la capacità di comunicare con una qualunque sorgente dati, a patto che per essa sia stato implementato il relativo driver JDBC. Semplificando, un driver JDBC rende possibili tre operazioni:

- stabilire una connessione con la sorgente dati;
- inviare Query;
- elaborare i risultati ottenuti.

LE API JDBC sono contenute nei package:

- java.sql;
- javax.sql.

L'utilizzo di JDBC [12] consente di realizzare il collegamento tra i Wrapper e la sorgente dati relazionale da presidiare (DB2 Universal Database [10] e Oracle Database [11] nel caso della presente tesi).

2.1 Stabilire una connessione

Per iniziare il dialogo con una sorgente dati è necessario innanzitutto caricare il driver compatibile con la fonte informativa utilizzando il metodo *Class.forName(String drivername)* e stabilire una connessione, creare cioè un canale di comunicazione tra il modulo software e la sorgente dati. L'oggetto *Connection* rappresenta la connessione al database, in ogni sessione di connessione vengono inviati statement SQL e ricevuti risultati. Un'applicazione può avere più connessioni con un singolo database oppure collegarsi a più database. Il modo

classico per stabilire una connessione con la sorgente dati è chiamare il metodo *DriverManager.getConnection* passandogli come parametro un URL.

2.2 Esecuzione di statements SQL

Per inviare statement SQL ad un database si utilizzano gli oggetti *Statement*, attualmente esistono tre tipi di oggetti *Statement*, ognuno dei quali permette di eseguire statement SQL su una data connessione:

- ***Statement***: usati per eseguire semplici statement SQL senza parametri;
- ***PreparedStatement***: usati per eseguire statement SQL precompilati con o senza parametri d'ingresso;
- ***CallableStatement*** usati per invocare una store procedure.

Per crearli si utilizza il metodo *createStatement* della classe *Connection*, come mostrato nell'esempio seguente:

```
Connection con = DriverManager.getConnection(URL);
Statement stmt = con.createStatement();
```

Gli statement SQL verranno inviati al database come argomenti di uno dei metodi *execute* di un oggetto Statement, ad esempio:

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table2");
```

La variabile rs si riferisce al ResultSet, cioè alla risposta del database, ovviamente una eventuale modifica ad rs non sortirà nessun effetto sulla base di dati. Ci sono tre tipi di metodi differenti per eseguire statement SQL:

- ***executeQuery***: serve per quei comandi che producono un singolo result set, come ad esempio una query di selezione (SELECT);
- ***executeUpdate***: viene utilizzato per eseguire statement del tipo INSERT, UPDATE o DELETE, oppure comandi per la definizione dei dati (SQL DDL) come ad esempio CREATE TABLE, DROP TABLE, ALTER TABLE. Il valore di ritorno di executeUpdate è un intero che indica il numero di righe modificate. Per comandi che non operano sulle righe (CREATE TABLE,...) il valore di ritorno è sempre zero;
- ***execute***: Si usa per quei comandi che producono più di un result set.

2.3 DatabaseMetaData

Finora si è parlato di come realizzare la connessione ad un database e di come interrogarlo, le descrizioni delle sorgenti dati, però, non contengono i dati memorizzati nelle sorgenti bensì informazioni riguardanti la struttura delle stesse. Le informazioni necessarie sono:

- nomi delle tabelle;
- nomi e tipi dei campi contenuti nelle tabelle;
- chiavi primarie;
- foreign key;

Le API JDBC mettono a disposizione un'interfaccia chiamata *DataBaseMetaData* che dà la possibilità di ottenere questi dati, o meglio questi *metadati*.

2.4 I driver Oracle/JDBC

Oracle fornisce quattro diversi tipi di driver per usi diversi, dalla versione 10.1 possono connettersi al server Oracle 8.1.7 (e alle versioni successive) [20]:

- ***JDBC OCI client-side driver*** usa metodi JAVA per utilizzare librerie C (OCI, Oracle Call Interface) che interagiscono con un database Oracle. Per funzionare richiede l'installazione client di Oracle della stessa versione del driver. JDBC OCI driver è dipendente dalla piattaforma;
- ***JDBC Thin client-side driver*** usa JAVA per connettersi direttamente con il server Oracle tramite TCP/IP implementato da socket java. Non richiede l'installazione di software client, ma solo di un listener TCP/IP configurato sul server. Poichè realizzato completamente in JAVA è indipendente dalla piattaforma;
- ***JDBC Thin server-side driver*** scritto anch'esso in JAVA offre le stesse funzionalità del precedente, ma può essere usato internamente a un server Oracle per collegarsi a database remoti;
- ***JDBC Server-Side Internal driver*** usa metodi JAVA per utilizzare delle librerie C che sono parte del processo server del database Oracle e comunica direttamente con il motore SQL interno in modo da evitare il traffico di rete. Accede così in modo ottimizzato e più veloce possibile, ma puo essere usato da codice JAVA solo se sullo stesso server.

È evidente che il driver più adatto ad essere inserito nel progetto MOMIS [6] è *JDBC Thin client-side*.

Per caricare il driver la stringa da passare al metodo *Class.forName()* è:

```
oracle.jdbc.driver.OracleDriver
```

Per aprire la connessione con *DriverManager.getConnection()* la stringa url è nel formato:

```
jdbc:oracle:<drivertype>:<user>/<password>@<host>[:<port>]//<service>
```

- **<drivertype>** può assumere, a seconda del tipo di driver i valori:

thin oci oci8 kprb

ad esempio usando il driver ***JDBC Thin client-side***:

```
jdbc:oracle:thin:scott/tiger@//myserver.com:5521/customer_db
```

2.5 I driver DB2/JDBC

IBM fornisce tre diversi driver [21]:

- ***DB2 Universal JDBC Driver***
  può connettersi a DB2 UDB for Linux, UNIX e Windows o a DB2 UDB per z/OS;
- ***JDBC/SQLJ 2.0 Driver for OS/390***
  può connettersi solo a DB2 UDB per z/OS;
- ***DB2 JDBC Type 2 Driver for Linux, UNIX and Windows***
  può connettersi solo a DB2 UDB per Linux, UNIX e Windows.

Il driver *DB2 Universal JDBC Driver* risulta compatibile con entrambe le versioni di DB2 Universal Databse ed è quindi ideale per il wrapper DB2/ ODL$_{I^3}$ di MOMIS [6].

Per caricare il driver la stringa da passare al metodo *Class.forName* è:

com.ibm.db2.jcc.DB2Driver

Per aprire la connessione con *Connection DriverManager.getConnection(String url)* la stringa url è nella forma:

jdbc:db2:<host>:<port>/<service>:user=<user>;password=<password>;

ad esempio:

```
jdbc:db2://ibm.com:5021/san_jose:user=db2adm;password=db2adm;
```

3 Il Progetto MOMIS

Il progetto MOMIS[1,2,6] è nato dalla collaborazione tra l'Università di Modena e Reggio Emilia e l'Università di Milano e Brescia, nell'ambito del progetto nazionale INTERDATA e del progetto D2I sotto la direzione della Professoressa Sonia Bergamaschi. Ora l'attività di ricerca prosegue all'interno del progetto di ricerca europeo SEWASIE [8,13].

MOMIS [6] è un framework per l'estrazione e l'integrazione di sorgenti di dati strutturate e semistrutturate. Il processo di integrazione produce una vista globale (*Global Schema*) contenente tutti i concetti che le diverse fonti informative vogliono condividere e che sarà l'unica con la quale l'utente dovrà interagire. Tale vista globale viene realizzata in modo semiautomatico a partire da un'ontologia comune (*Common Thesaurus* generato dal sistema) e su tale vista l'utente può richiedere il reperimento di informazioni da qualsiasi delle sorgenti sottostanti tramite un'architettura a mediatore [1]. Allo scopo di presentare ed integrare le informazioni estratte ed integrate in un linguaggio comune viene introdotto un linguaggio object-oriented, chiamato ODL$_{I3}$ (vedi paragrafo 3.2).

3.1 L'Architettura di MOMIS

MOMIS [6] è stato realizzato seguendo l'architettura di riferimento I^3 [3] (vedi capitolo 2), si possono distinguere quattro livelli durante la fase di integrazione:

1) ***Sorgenti***: sono al livello più basso e rappresentano le fonti di informazioni che devono essere integrate. I tipi di sorgenti alle quali MOMIS [6] può accedere sono database tradizionali (ad esempio relazionali, object-oriented), file system, sorgenti di tipo semistrutturato (lo standard de facto per sorgenti dati semistrutturate è oggi XML);
2) ***Wrapper***: sono i moduli più vicini alle sorgenti informative e rappresentano l'interfaccia tra il mediatore e le varie sorgenti locali di dati, per questo devono essere sviluppati appositamente per il tipo di sorgente che andranno a interfacciare. La funzione dei wrapper è duplice:
    - nella fase di integrazione, si occupano di descrivere gli schemi delle sorgenti nel linguaggio ODL$_{I3}$;

- nella fase di query processing, devono tradurre le query ricevute dal mediatore (espresse nel linguaggio comune OQL_{I^3}) in interrogazioni comprensibili dalla sorgente con cui ognuno di essi opera. Inoltre deve presentare al mediatore, attraverso il modello comune di dati utilizzato dal sistema, i dati ricevuti in risposta alla query presentata;

3) ***Mediatore***: può essere definito il cuore del sistema poichè combina, integra e ridefinisce gli schemi ricevuti dai wrapper tramite un'interfaccia ODL_{I^3}. È costituito da diversi sottomoduli:
   - ***Global Schema Builder***: è il modulo di integrazione degli schemi locali. Partendo dalle descrizioni delle sorgenti espresse in ODL_{I^3}, genera un unico schema globale da presentare all'utente. Questa fase di integrazione, realizzata in modo semi-automatico con l'interazione del progettista del sistema, utilizza gli ODB-Tools [5], le tecniche di clustering e quelle di fusione delle gerarchie;
   - ***Query manager***: è il modulo che gestisce le interrogazioni. Provvede a gestire la query dell'utente, prima deducendone da essa un'insieme di sottoquery da sottoporre alle sorgenti locali, poi ricomponendo le informazioni ottenute, ed inoltre ha anche il compito di provvedere all'ottimizzazione semantica delle interrogazioni utilizzando gli ODB-Tools [5];
4) ***Utente***: è colui che interagisce con il mediatore e al quale viene presentato lo schema globale, nel cui interno sono rappresentati tutti i concetti che possono essere interrogati. In pratica per l'utente è come se si trovasse di fronte ad un unico database da interrogare in modo tradizionale, mentre le sorgenti locali restano completamente trasparenti.

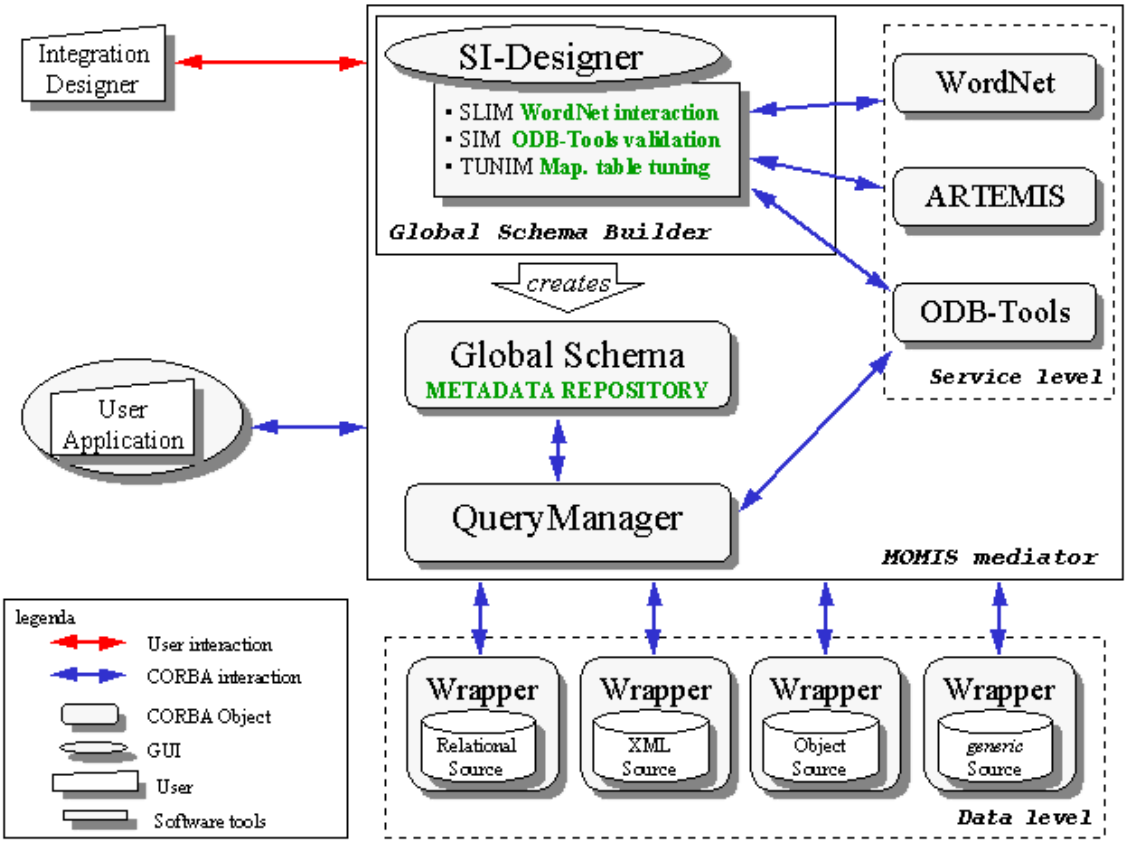


Figura 2: Architettura di MOMIS

In figura 2 si possono notare anche il tool SI-Designer [7] (vedi paragrafo 3.3) e altri tools software:

- ***ODB-Tools*** [5] è un sistema per l'acquisizione e la verifica di consistenza di schemi di basi di dati e per l'ottimizzazione semantica di interrogazioni nelle basi di dati orientate agli oggetti (OODB). ODB-Tools [5] e' stato sviluppato presso l'Universita' di Modena e Reggio Emilia, sotto la direzione della Professoressa Sonia Bergamaschi;
- ***ARTEMIS*** [14] è un tool basato sulle tecniche di affinità e di clustering che esegue l'analisi semantica ed il clustering delle classi ODL_{I^3}. L'analisi semantica ha come obiettivo l'identificazione degli elementi che hanno relazione semantiche nei diversi schemi. Il concetto di affinità viene introdotto per valutare il livello di relazione semantica tra gli elementi di un schema. ARTEMIS [14] è stato sviluppato presso l'Università di Milano e Brescia;
- ***WordNet*** [15] un database lessicale della lingua inglese, capace di individuare relazioni lessicali e semantiche tra termini.

- ***CORBA*** [16,17] non è un vero e proprio tool, ma il protocollo per lo scambio di oggetti tra i moduli adottato da MOMIS [6]. Con CORBA (Common Object Request Broker Architecture) [16,17] si indicano delle specifiche di un'architettura e infrastruttura che le applicazioni possono usare per interagire in rete indipendentemente dalla macchina, sistema operativo o linguaggio di programmazione. Le specifiche sono pubblicate da OMG (Object Management Group)[17], un consorzio di oltre 800 aziende.

3.2 Il linguaggio ODL$_{I3}$

ODL$_{I3}$ è il linguaggio di descrizione introdotto per la comunicazione tra wrapper e mediatore. Le caratteristiche di ODL, al pari di altri linguaggi basati sul paradigma ad oggetti, si possono riassumere in questo modo [2]:

- Definizione di tipi-classe e tipi-valore;
- Distinzione fra intensione ed estensione di una classe di oggetti;
- Definizione di attributi semplici e complessi;
- Definizione di attributi atomici e collezioni (set, list, bag);
- Definizione di relazioni binarie con relazioni inverse;
- Dichiarazione della signature dei metodi.

Il linguaggio ODL è certamente ben progettato e adatto alla descrizione di un singolo schema ad oggetti, risulta però incompleto se calato nel contesto di integrazione di basi di dati eterogenee di MOMIS [6]. Quindi ODL$_{I3}$, in accordo con gli standard ODMG e I^3, riprende le specifiche del linguaggio ODL aggiungendo estensioni utili per l'integrazione di informazioni e per la modellazione di dati semistrutturati [1,2]:

- **Costruttore *union***: con questo costrutto ogni classe può avere più strutture dati alternative.
- **Costruttore * (Optional)**: indica l'opzionalità degli attributi;

- **Relazioni terminologiche**:
  - *sinonimia (SYN)*
    definita tra due termini che sono considerati sinonimi, ovvero che possono essere interscambiati nelle sorgenti, identificando lo stesso concetto del mondo reale (ad esempio, `Worker SYN Employee`);
  - *ipernimia (BT)*
    definita tra due termini il cui il primo ha un significato più generale del secondo (ad esempio, `Organization BT Public_Organization`);
  - *iponimia (NT)*
    definita tra due termini è la relazione inversa di *ipernimia;*
  - *associazione (RT)*
    definita tra termini che generalmente sono usati nello stesso contesto (ad esempio, `Organization RT Employee`)
- **Regole**:
  - *If-then*: per esprimere in forma dichiarativa i vincoli di integrità inter-schema e intra-schema;
  - *Mapping*: per esprimere le relazioni tra lo schema integrato e gli schemi sorgente.

Ad esempio la descrizione in ODL_{I^3} della tabella TBL_DIPARTIMENTI e TBL_INSEGNAMENTITI_PROPEDEUTICITA dell'esempio di riferimento (capitolo 5):

```
interface TBL_DIPARTIMENTI(extent TBL_DIPARTIMENTI)
   {
      attribute string NOTE;
      attribute string DENOMINAZIONE;
      attribute string FAX;
      attribute float IDREFERENTE;
      attribute string SITOWEB;
      attribute string INDIRIZZO;
      attribute string EMAIL;
      attribute float ID;
      attribute string TELEFONO}}
```

```
interface TBL_INSEGNAMENTITI_PROPEDEUTICITA ( extent TBL_INSEGNAMENTITI_PROPEDEUTICITA
     key (IDINSEGNAMENTO, IDINSEGNAMENTOPROPEDEUTICO)
     foreign_key FK_PROP_INS(IDINSEGNAMENTO)references TBL_INSEGNAMENTI(ID)
     foreign_key FK_PROP_INS1(IDINSEGNAMENTOPROPEDEUTICO)references TBL_INSEGNAMENTI(ID))
  {
     attribute float TIPO;
     attribute float IDINSEGNAMENTO;
     attribute float IDINSEGNAMENTOPROPEDEUTICO;}
```

Questa descrizione ODL$_{I3}$ è stata ottenuta con SI-Designer su DB2 Universal Database (esempio di riferimento paragrafo 4.1).

3.3 SI-Designer

SI-Designer [18,7] è la GUI (Interfaccia Utente Grafica) che guida l'utente attraverso le varie fasi dell'integrazione, dall'acquisizione delle sorgenti fino alla messa a punto del *Common Thesaururs*. SI-Designer è composto da quattro moduli:

- **SIM** (*Source Integrator Module*): estrae le relazioni intra-schema sulla base della struttura delle classi ODL$_{I3}$ e delle sorgenti relazionali usando ODB-Tools [5]. Inoltre effettua la "validazione semantica" delle relazioni e ne inferisce delle nuove sfruttando sempre ODB-Tools [5].
- **SLIM** (*Sources Lexical Integrator Module*): estrae le relazioni inter-schema tra nomi di attributi e classi ODL$_{I3}$ sfruttando il sistema lessicale WordNet.
- **TUNIM** (*Tuning of the mapping Table*): questo modulo gestisce la fase di creazione dello schema globale.

La GUI di SI-Designer [7] è una sequenza di finestre, ognuna delle quali relativa ad una fase del processo di integrazione, e mette a disposizione l'interfaccia per interagire con i moduli SIM, SLIM ed ARTEMIS [14]. Il tool è stato progettato ed implementato in Java e ha un'architettura modulare, in modo da rendere molto semplice l'aggiunta di nuove fasi nel processo di integrazione.

3.4 I Wrapper nell'Architettura di MOMIS

I Wrapper in MOMIS [6] costituiscono il punto d'accesso alle sorgenti dati. Ogni sorgente dati (relazionale, object, XML, ...) dev'essere *presentata* a MOMIS [6] in modo standard, e questo è proprio quello che fa il wrapper. In MOMIS [6] il Wrapper è un oggetto CORBA [16,17] che è connesso ad una sorgente e che è in grado di descrivere la struttura di tale sorgente, inoltre permette di interrogare la sorgente usando il linguaggio OQL_{I3}.

Attualmente i wrapper implementati in MOMIS [6] sono:

- Wrapper JDBC generico;
- Wrapper per SQLServer;
- Wrapper XML;
- Wrapper JDBC-ODBC-MS_Access;
- Wrapper per Microsoft SQL Server;

3.4.1 L' interfaccia *Wrapper*

La classe *Wrapper* di MOMIS [6] è un'interfaccia CORBA [16,17] che tutti i wrapper in MOMIS [6] devono implementare, ad esempio i wrapper per sorgenti accessibili tramie JDBC [12] seguono la gerarchia in figura 3:

```
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_DB2Server
        |
        +--it.unimo.dbgroup.momis.communication.core.jdbc.WrapperJdbcCore_oracle
        |
        +--it.unimo.dbgroup.momis.communication.core.jdbc.WrapperJdbcCore_SQLServer
```

Figura 3 :gerarchia delle classi wrapper

L'interfaccia ***Wrapper*** dichiara I seguenti metodi*:*

- *public String getDescription()*
  ritorna la descrizione della sorgente in formato ODL*I3*;

- *public String getDescriptionXml()*
  ritorna la descrizione della sorgente in formato XML;

- *public String getSourceName()*
  ritorna la stringa rappresentante il nome della sorgente;

- *public String getType()*
  ritorna il tipo di sorgente (attualente può essere solo JDBC o XML);

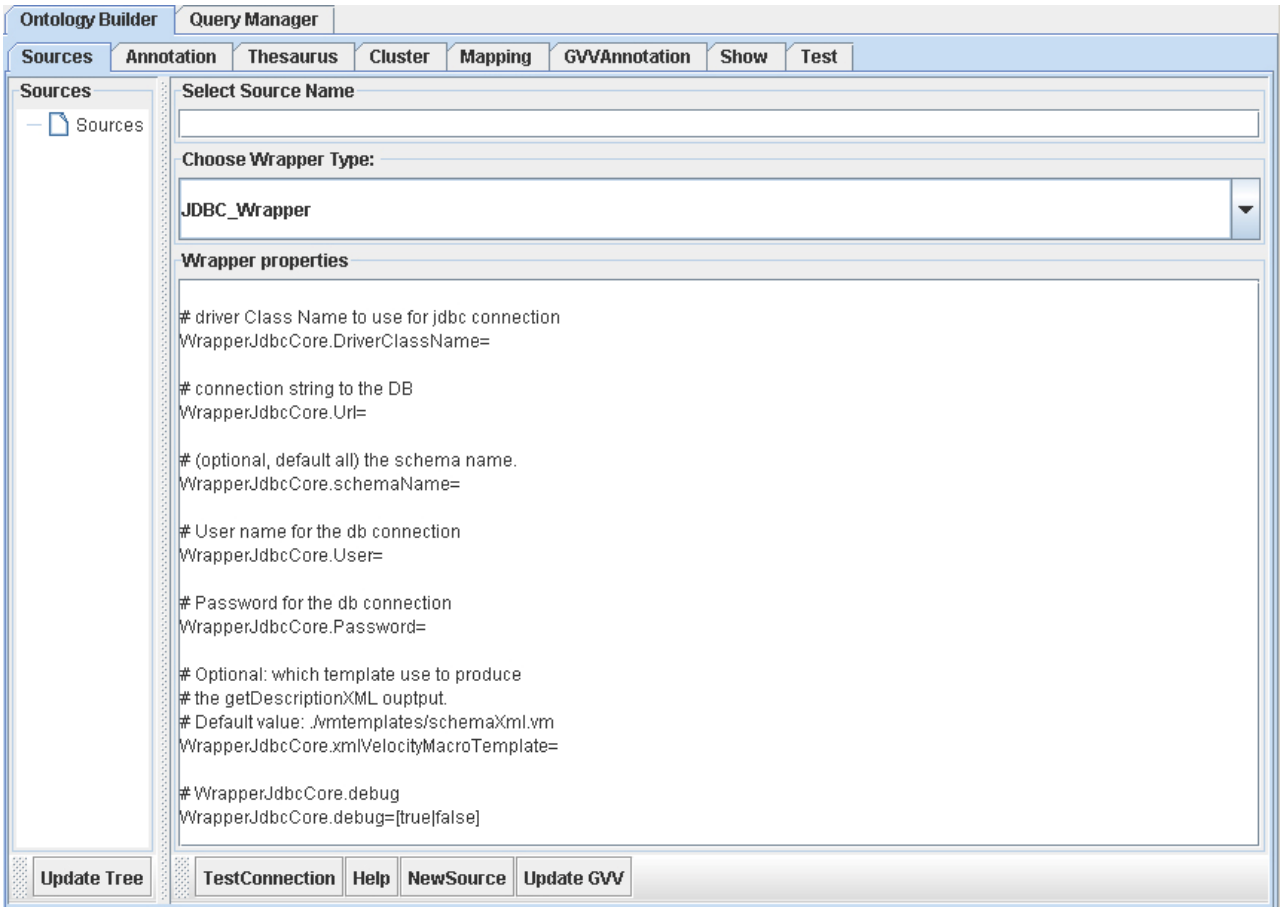
- *public WrapperRS runQuery (String oql)*
  esegue la query in formato OQL (Object Query Language);

- *public void close()*
  chiude la connessione del wrapper;

L'interfaccia ***WrapperCore*** estende la classe Wrapper dichiarando:

- *public void initialize(Map parameters)*
  inizializza i parametri del wrapper, tali parametri sono passati alla funzione in una classe che implementa l' interfaccia *Map*, la quale associa un valore ad una chiave.

- *public String getParamersAsPropertiesTemplate()*
  ritorna tutti i parametri che il wrapper necessita per funzionare, pronti per essere riempiti in un editor di testo, come è mostrato nella sessione SI-Designer in figura 4



.

Figura 4: parametri di connessione

3.4.2 Il Wrapper JDBC

Il wrapper JDBC [12] è stato concepito per le sorgenti relazionali interfacciabili tramite JDBC (vedi capitolo 2) ed è implementato nella classe *WrapperJdbcCore*. La classe *WrapperJdbcCore* implementa l'interfaccia *WrapperCore*, che estende a sua volta l'interfaccia *Wrapper*.

WrapperJdbcCore implementa tutti metodi dichiarati dalle interfacce *Wrapper* e *WrapperCore* e ne definisce degli altri. *Schema GetSchema()* è il metodo che estrae lo schema dal database indicato nei parametri di connessione, esso recupera i metadati dalla connessione instaurata e ritorna uno *Schema* ODL_{I^3} con tali informazioni.

Schema è la classe ODL_{I^3} che descrive una fonte di dati integrata in una sessione di MOMIS [6] tutte le entità definite in ODL_{I^3} sono accessibili da istanze di questa classe, in

particolare nel metodo *getSchema()* vengono istanziate e inserite nello *Schema* le seguenti classi ODL_{I^3}:

- ***Source***
  corrispondente alla sorgente informativa da estrarre e integrare;
- ***Interface***
  identifica una tabella;
- ***IntBody***
  è usato come "contenitore" di attributi (colonne) per *Interface;*
- ***SimpleAttribute***
  un attributo semplice o complesso è usato per descrivere le colonne.

La classe *WrapperJdbcCore* non presenta ancora le peculiarità di un particolate tipo di sorgente tabellare, dovrà quindi essere ancora specializzata. Poichè implementa l'interfaccia *WrapperCore* definisce tutte le funzioni ereditate da tale classe, anche quelle dichiarate nell'interfaccia *Wrapper*, ne definisce anche delle altre, tra le quali sono degne di nota:

- *public Schema getSchema()*
  recupera i metadati dalla connessione e inserisce in uno schema ODL_{I^3} i componenti dello schema relazionale estratti, cioè i nomi delle tabelle, le colonne, il tipo di dato di ogni colonna. Non estrae ancora le primary key e le foreign key, quindi dovrà essere ridefinita nelle classi gerarchicamente successive (**WrapperJdbcCore_DB2Server**, **WrapperJdbcCore_Oracle e WrapperJdbcCore_SQLServer**) data la dipendenza di tale operazione dalla sorgente.
- *public boolean theTableShouldBeDescribed(String tableName)*
  ritorna "true" se la tabella passata deve essere considerata nell'integrazione, dovrà essere ridefinita nelle classi gerarchicamente successive (*WrapperJdbcCore_DB2Server , WrapperJdbcCore_Oracle e WrapperJdbcCore_SQLServer*) perché le tabelle da non considerare sono diverse tra un tipo di sorgente e l'altro (vedi capitolo 4).

- *public static Type getAttributeType(int t)*
  prende in input un tipo JDBC [12] e ne ritorna il corrispondente tipo ODL_{I^3}, in figura 5 sono mostrate le corrispondenze tra i tipi JDBC - ODL_{I^3}:

JDBC	**ODL$_{I^3}$**
ARRAY	Unsupported
BIGINT	IntegerType(true,true) = Long
BINARY	Unsupported
BIT	BooleanType
BLOB	Unsupported
CHAR	CharType
CLOB	Unsupported
DATE	DateType
DECIMAL	FloatingType(false) = Float
DISTINCT	Unsupported
DOUBLE	FloatingType(true) = Double
FLOAT	FloatingType(true) =Double
INTEGER	IntegerType(false,true) = Short
LONGVARBINARY	StringType(unsupported)
LONGVARCHAR	StringType(unsupported)
NUMERIC	FloatingType(true) =Double
OTHER	AnyType (?)
REAL	FloatingType(true) =Double
REF	Unsupported
SMALLINT	IntegerType(false,true) = Short
STRUCT	Unsupported
TIME	DateType
TIMESTAMP	DateType
TINYINT	IntegerType(false,true) = Short
VARBINARY	StringType(unsupported?)
VARCHAR	StringType

Figura 5: Mapping tra tipi JDBC e ODL$_{I^3}$

In conclusione il wrapper JDBC [12] è in grado di estrarre da una sorgente lo schema del database (nome delle tabelle, nome e tipo delle colonne), ma non le primary key e le foreign key. Ad esempio, estraendo la tabella TBL_INSEGNAMENTI_AA dell'esempio di riferimento su un server DB2 (paragrafo 4.1) ha generato il seguente codice ODL [13] nel quale non compaiono primary key e foreign key:

```
interface TBL_INSEGNAMENTI_AA (extent TBL_INSEGNAMENTI_AA ){
   attribute float IDINSEGNAMENTO;
   attribute float ANNOCORSO;
   attribute string PROGRAMMA;
   attribute short DELETED;
   attribute char PASSWORD;
   attribute short PUBBLICATA;
   attribute string MATERIALEDIDATTICO;
   attribute char LOGIN;
   attribute string TESTICONSIGLIATI;
   attribute date ULTIMOAGGIORNAMENTO;
   attribute string NOTE;
   attribute float IDPERIODO;
   attribute string VODPROP;
   attribute short PUBBLICA;
   attribute float ID;}
```

4. I wrapper DB2/ODL$_{I3}$ e Oracle/ODL$_{I3}$

Le classi *WrapperJdbcCore_DB2Server* e *WrapperJdbcCore_oracle* (e anche *WrapperJdbcCore_SQLServer*) estendono la classe *WrapperJdbcCore* aggiungendo delle nuove funzionalità:

- estrazione di primary key;
- estrazione di foreign key;
- esclusione automatica delle tabelle di sistema del DBMS.

Per estrarre primary key e foreign key viene ridefinita la funzione *GetSchema()* senza però modificare il codice già esistente per l'estrazione dello schema.

Sia la prima che la seconda funzionalità sono implementate con lo stesso codice in tutti e tre le classi sopra citate.

Per descrivere primary key e foreign key ODL$_{I3}$ dispone delle seguenti classi:

- ***KeyList*** per le chiavi, un'istanza di questa classe contiene un insieme dei nomi delle colonne che formano la chiave. Nel caso di primary key è necessario impostare il nome dell'istanza *KeyList* come "`PrimaryKey`";
- ***ForeignKey*** per le foreign key, ogni istanza di tale classe è una foreign key, cioè una corrispondenza tra un'attributo locale e l'attributo della tabella a cui si referenzia.

Per quanto riguarda l'esclusione delle tabelle nelle tre classi prima citate viene reimplementato il metodo *boolean theTableShouldBeDescribed()*, tale metodo è definito già in *WrapperJdbcCore* ma non effettua nessuna effettiva selezione poichè la scelta delle tabelle da non estrarre dipende dalla sorgente associata al wrapper. Le tabelle da scartare sono le tabelle di sistema, cioè quelle che ogni DBMS crea e aggiorna automaticamente per mantenere informazioni sul sistema in generale.

In Oracle l'insieme delle tabelle di sistema è conosciuto come "***data dictionary***" [33]. Il data dictionary è parte di un database Oracle e fornisce informazioni riguardo:

- La definizione di tutti gli oggetti nel database (tabelle, viste, indici, cluster, sinonimi, sequenze, procedure, triggers...);
- Quanta memoria è allocata per tali oggetti;
- Valori di default delle colonne;

- Vincoli di integrità;
- Utenti di Oracle;
- Privilegi e ruoli ad essi concessi;
- Altre informazioni generali.

Data dictionary è strutturato in tabelle e viste, che sono memorizzate nel tablespace *SYSTEM* di ogni database. Poichè è impostato in sola lettura si possono eseguire solo delle query.

In DB2 l'insieme di tabelle di sistema è noto come "***catalog"*** [23,24]. In DB2 UniversalDatabase per Linux, Unix, Windows ogni database ha il proprio "catalogo", che contiene informazioni sugli oggetti del database di appartenenza ma non sul resto del server. Invece in DB2 Universal Database per z/OS il catalogo è relativo all'intero sistema ed è memorizzato nel database DSNDB06 che può essere interrogato tramite quey SQL.

Per implementare la funzione ***theTableShouldBeDescribed()*** per i wrapper DB2/ODL$_{I3}$ e Oracle/ODL$_{I3}$ inizialmente si è deciso di creare una lista dei nomi delle tabelle da escludere e salvarla su file di configurazione. Tale file è stato ottenuto connettendosi a due database "vuoti", uno su un server DB2 e l'altro su un server Oracle con un programma JAVA e utilizzando il metodo *GetTables()* [19] dell'interfaccia *DatabaseMetadata* fornita da JDBC [12].

La funzione *GetTables* restituisce un ResultSet contentente la descrizione di tutte le tabelle contenute nel database e rispondenti ai criteri di ricerca impostati tramite i parametri passati:

```
ResultSet getTables(String catalog,
                    String schemaPattern,
                    String tableNamePattern,
                    String[] types)
```

Nel *ResultSet* che si ottiene dall'esecuzione di questo metodo possono essere recuperate informazioni riguardo:

- **TABLE_SCHEM**: schema di appartenenza della tabella;
- **TABLE_NAME**: nome della tabella;

- **TABLE_TYPE**: tipo di tabella (tipici valori sono "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS" o "SYNONYM");

Al momento dell'estrazione della sorgente il file viene caricato su un *Vector* e i nomi delle tabelle processate dalla funzione *getSchema()* vengono confrontati con gli oggetti (*String*) in tale lista.

Questa soluzione è però ridondante per due motivi:

- molti dei nomi elencati nei file in realtà corrispondono a tabelle che non ritornano come TABLE_TYPE il valore "TABLE", e poichè *getSchema()* processa solo le tabelle di tipo "TABLE" (*WrapperJdbcCore.java* riga 325) tali nomi possono essere esclusi dai file di configurazione;
- le tabelle di sistema di DB2 hanno come valori possibili di TABLE_SCHEM quelli riservati "SYSIBM" o "SYSTOOLS", quindi tali tabelle si possono riconoscere in base al valore di TABLE_SCHEM.

  Analogamente si possono escludere le tabelle di Oracle che hanno come valore di TABLE_SCHEM uno tra: "MDSYS", "OLAPSYS", "SYS", "SYSTEM", "WKSYS" e "XDB".

Attualmente le funzioni *theTableShouldBeDescribed()* in entrambe le classi *WrapperJdbcCore_DB2Server* e *WrapperJdbcCore_oracle* ritornano il valore *true* o *false* in base al valore assunto da TABLE_SCHEM (ed esempio *WrapperJdbcCore_DB2Server.java* righe 95,112,114,253).

4.1 Esempio di riferimento

Per testare in modo completo i wrapper realizzati è stato usato un caso reale, cioè un database su SQL Server sul server apollo13.ing.unimo.it che si riferisce ad una realtà universitaria che gestisce corsi (categoria, classe MIUR...), appelli, insegnamenti (ambiti, attività, moduli, struttura organizzativa, propedeuticità, materiale didattico ...), facoltà, docenti, aule, avvisi, orari, pubblicazioni e altre entità, per un totale di 64 tabelle in relazione tra loro con 46 chiavi esterne (alcune tabelle e relazioni sono riassunte in forma grafica in figura 6).

Il codice SQL è stato generato da un tool di SQLServer sul server apollo13.ing.unimo.it, ma non sono stati utilizzati i valori di default che non vengono estratti dai wrapper, quindi inutili ai fini del test, e i tipi booleani (o bit) non esistenti ne in DB2 [28] ne in Oracle [29] sono stati sostituiti con dei valori interi.

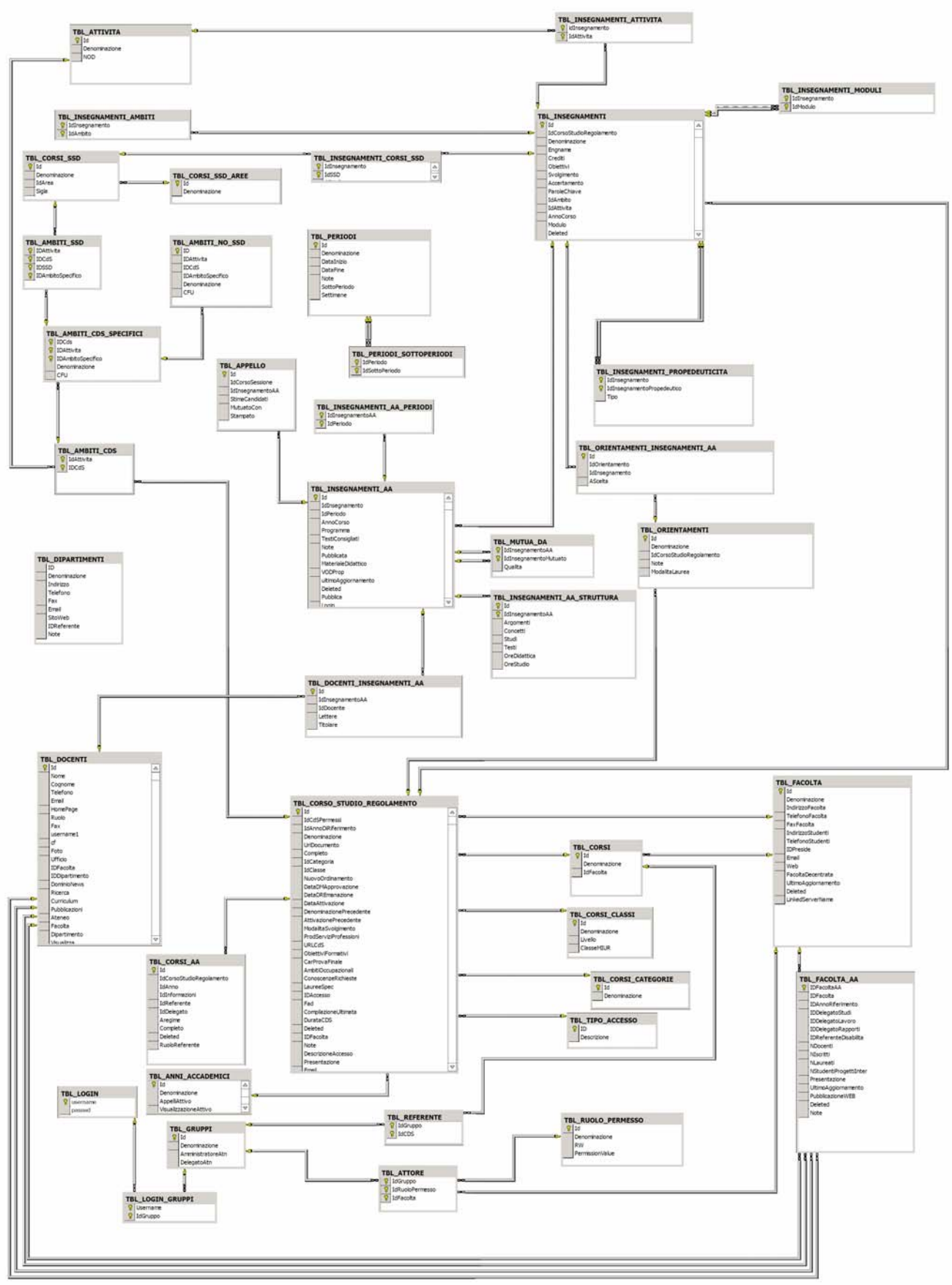


Figura 6: database di esempio

A titolo di esempio sono mostrati i risultati delle estrazioni della tabella **TBL_INSEGNAMENTI_AA:**

- codice SQL eseguito per la creazione(le colonne Pubblica e Deleted sono di tipo INT perchè il tipo booleano non è supportato da DB2 e da Oracle):

```
CREATE TABLE TBL_INSEGNAMENTI_AA (
      Id numeric(18, 0) NOT NULL,
      IdInsegnamento numeric(18, 0) NOT NULL,
      IdPeriodo numeric(18, 0),
      AnnoCorso numeric(18, 0),
      Programma varchar (50),
      TestiConsigliati varchar (50),
      Note varchar (50),
      Pubblicata int,
      MaterialeDidattico varchar (200),
      VODProp varchar (150),
      ultimoAggiornamento date,
      Deleted INT,
      Pubblica INT,
      Login char (50),
      Password char (50));
```

- ***WrapperJdbcCore_SQLServer*** ha prodotto il seguente codice ODL*I3*:

```
interface TBL_INSEGNAMENTI_AA ( extent TBL_INSEGNAMENTI_AA
     key (Id)
     foreign_key  FK_TBL_INSEGNAMENTI_AA_TBL_INSEGNAMENTI  (IdInsegnamento)
           references TBL_INSEGNAMENTI(Id))
{
      attribute boolean Pubblica;
      attribute double AnnoCorso;
      attribute double IdPeriodo;
      attribute double IdInsegnamento;
      attribute short Pubblicata;
      attribute date ultimoAggiornamento;
      attribute char Login;
      attribute char Password;
      attribute string Programma;
```

```
    attribute string MaterialeDidattico;
    attribute string Note;
    attribute string VODProp;
    attribute boolean Deleted;
    attribute string TestiConsigliati;
    attribute double Id;}
```

- ***WrapperJdbcCore_DB2Server*** ha prodotto il seguente codice ODL$_{I3}$:

```
    interface TBL_INSEGNAMENTI_AA ( extent TBL_INSEGNAMENTI_AA
          key ID)
          foreign_key FK_TBL_INS_AA_TBL_INS IDINSEGNAMENTO)
                references TBL_INSEGNAMENTI ID) )
{
    attribute float IDINSEGNAMENTO;
    attribute float ANNOCORSO;
    attribute string PROGRAMMA;
    attribute short DELETED;
    attribute char PASSWORD;
    attribute short PUBBLICATA;
    attribute string MATERIALEDIDATTICO;
    attribute char LOGIN;
    attribute string TESTICONSIGLIATI;
    attribute date ULTIMOAGGIORNAMENTO;
    attribute string NOTE;
    attribute float IDPERIODO;
    attribute string VODPROP;
    attribute short PUBBLICA;
    attribute float ID;}
```

- ***WrapperJdbcCore_oracle*** ha prodotto il seguente codice ODL$_{I3}$:

```
    interface TBL_INSEGNAMENTI_AA ( extent TBL_INSEGNAMENTI_AA
         key (ID)
         foreign_key FK_TBL_INS_AA_TBL_INS (IDINSEGNAMENTO)
          references TBL_INSEGNAMENTI (ID) )
    {
          attribute float IDINSEGNAMENTO;
```

```
        attribute float ANNOCORSO;
        attribute string PROGRAMMA;
        attribute float DELETED;
        attribute char PASSWORD;
        attribute float PUBBLICATA;
        attribute string MATERIALEDIDATTICO;
        attribute char LOGIN;
        attribute string TESTICONSIGLIATI;
        attribute date ULTIMOAGGIORNAMENTO;
        attribute string NOTE;
        attribute float IDPERIODO;
        attribute string VODPROP;
        attribute float PUBBLICA;
        attribute float ID;}
```

Sempre a titolo di esempio illustro quì di seguito l'estrazione della tabella TABLE **TBL_INSEGNAMENTI_AA_STRUTTURA:**

- codice SQL eseguito per la creazione:

```
CREATE TABLE TBL_INSEGNAMENTI_AA_STRUTTURA (
      Id numeric(18, 0) NOT NULL,
      IdInsegnamentoAA numeric(18,0) NOT NULL,
      Argomenti varchar (300),
      Concetti varchar (300),
      Studi varchar (300),
      Testi varchar (300),
      OreDidattica int,
      OreStudio int)
```

- ***WrapperJdbcCore_SQLServer*** ha prodotto il seguente codice ODL$_{I3}$:

```
      interface TBL_INSEGNAMENTI_AA_STRUTTURA ( extent
      TBL_INSEGNAMENTI_AA_STRUTTURA
         key (Id,IdInsegnamentoAA)
         foreign_key FK_TBL_INSEGNAMENTI_AA_STRUTTURA_TBL_INSEGNAMENTI_AA
          (IdInsegnamentoAA) references TBL_INSEGNAMENTI_AA (Id) )
    {
```

```
      attribute string Studi;
      attribute short OreDidattica;
      attribute string Argomenti;
      attribute string Concetti;
      attribute string Testi;
      attribute short OreStudio;
      attribute double Id;
      attribute double IdInsegnamentoAA;
}
```

- ***WrapperJdbcCore_DB2Server*** ha prodotto il seguente codice ODL_{I3}:

```
interface TBL_INSEGNAMENTI_AA_STRUTTURA ( extent
TBL_INSEGNAMENTI_AA_STRUTTURA
     key (ID, IDINSEGNAMENTOAA)
     foreign_key FK_TI_AA (IDINSEGNAMENTOAA)
      references TBL_INSEGNAMENTI_AA (ID) )
{
      attribute short OREDIDATTICA;
      attribute string STUDI;
      attribute string TESTI;
      attribute float IDINSEGNAMENTOAA;
      attribute string CONCETTI;
      attribute float ID;
      attribute string ARGOMENTI;
      attribute short ORESTUDIO;
}
```

- ***WrapperJdbcCore_oracle*** ha prodotto il seguente codice ODL_{I3}:

```
   interface TBL_INSEGNAMENTI_AA_STRUTTURA ( extent
TBL_INSEGNAMENTI_AA_STRUTTURA
     key (ID, IDINSEGNAMENTOAA)
     foreign_key FK_TBL_INS_AA_STR_TBL_INS_AA (IDINSEGNAMENTOAA})
      references TBL_INSEGNAMENTI_AA (ID) )
{
      attribute float OREDIDATTICA;
      attribute string STUDI;
```

```
        attribute string TESTI;
        attribute float IDINSEGNAMENTOAA;
        attribute string CONCETTI;
        attribute float ID;
        attribute string ARGOMENTI;
        attribute float ORESTUDIO;
}
```

Le descrizioni prodotte in linguaggio ODL$_{I3}$ dai wrapper DB2/ODL$_{I3}$, Oracle/ODL$_{I3}$ e SqLServer/ODL$_{I3}$ sono le stesse meno ti alcuni tipi di dato (in tabella), e sono in accordo con il codice SQL usato per generare le tabelle. In base a queste osservazioni si può quindi affermare che *WrapperJDBCCore_oracle* e *WrapperJDBCCore_DB2Server* funzionano correttamente.

tipo SQL	**tipo ODL*I3* estratto con *WrapperJdbcCore_SQLServer***	**tipo ODLI3 estratto con *WrapperJdbcCore_DB2Server***	**tipo ODLI3 estratto con *WrapperJdbcCore_oracle***
INT	Short	Short	float
NUMERIC	Double	Float	float

Queste diversità sono causate dai diversi mapping tra i tipi (vedi APPENDICI).
Ad esempio, poichè Oracle non ha il tipo numeric, le colonne NUMERIC(18,0) estratte dal wrapper Oracle/ODL$_{I3}$ sono state create di tipo NUMBER(18,0) come mostrato in APPENDICE C, come si vede in APPENDICE A il tipo NUMBER corrisponde a diversi tipi JDBC, in questo caso è stato mappato con DECIMAL o FLOAT, vale a dire il tipo Float di ODL$_{I3}$.

5 Le Wrapper GUI

Il tool SI-Designer [7] è una sequenza di finestre, ognuna delle quali relativa ad una fase del processo di integrazione. Tale framework è stato progettato ed implementato in Java ed ha un'architettura modulare, che semplifica l'aggiunta di nuove fasi nel processo di integrazione. È possibile aggiungere in SI-Designer [7] un pannello per ogni wrapper come interfaccia utente per l'inserimento dei parametri di connessione. Ad esempio nella figura 7 è mostrato *SQLServerWrapperGUI* durante una sessione di SI-Designer [7] con la classe GUI (senza GUI sarebbe come in figura 4):

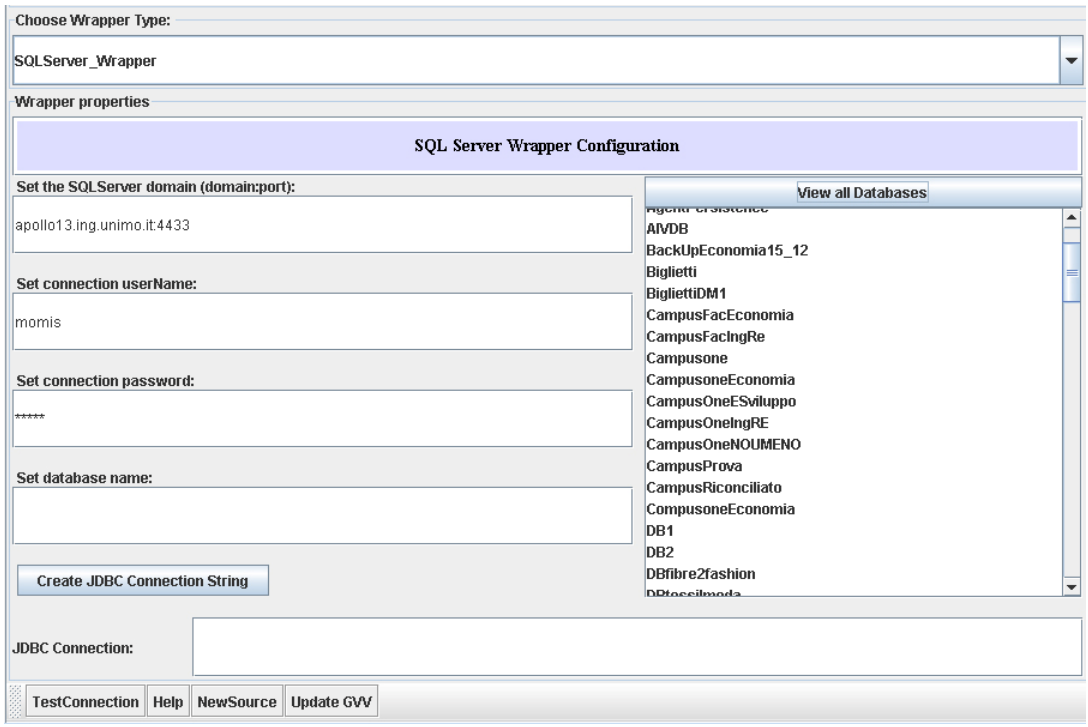


Figura 7: sessione *SQLServerWrapperGUI*

Il pannello GUI offre all'utente un'interfaccia più semplice da usare in quanto non solo non necessita la conoscenza del nome del driver per aprire il collegamento con la sorgente, ma presenta anche una funzionalità aggiuntiva che consiste nel recuperare i nomi di tutti database da un server.

Questo servizio è implementato dal metodo *getAllDatabase()* nella classe *SQLServerWrapperGUI*.

In ogni installazione di Microsoft SQLServer è presente di norma un database di sistema contenente informazioni su tutto il server, compresa la lista di database disponibili. Tale informazione è contenuta nel database di nome "`master`" nella tabella "`master.dbo.sysdatabases`", *getAllDatabase()* non fa altro che instaurare una connessione utilizzando l'URL del database, username e password inseriti nei campi del pannello, ed eseguire la query SQL "`select name from master.dbo.sysdatabases order by name`".

5.1 La classe *DB2ServerWrapperGUI*

L'interfaccia del wrapper per DB2 UDB [10] è del tutto simile a quella del wrapper per SQL Server, tutto il codice utilizzato per l'inizializzazione degli elementi grafici JAVA è uguale, la differenza sta nel metodo *String generateJDBCConnectionString()*. Tale funzione ha il compito di combinare le stringhe inserite nei campi in modo tale da ottenere la stringa richiesta per la connessione. Nel caso di *DB2 Universal JDBC Driver* tale stringa deve essere nel seguente formato:

```
jdbc:db2:<host>:<port>/<service>:user=<user>;password=<password>;
```

Nell'implementare la funzione ***getAllDatabase()*** si è trovata qualche difficoltà nel cercare una tabella o una vista che contenesse la lista dei database disponibili sul server.

È necessario fare una distinzione tra le due versioni di DB2 Universal Database sviluppate da IBM:

- ***DB2 Universal Database per z/OS***
- ***DB2 Universal Database per Linux, Unix, Windows***

In **DB2 Universal Database per z/OS** [24] esiste un "catalogo" di sistema, cioè un insieme di tabelle contenenti informazioni riguardo il sistema DB2, incluse le table space, gli indici, le tabelle, gruppi di memorizzazione e altre informazioni caratteristiche di un server DB2. Tale catalogo è contenuto nel database di sistema DSNDB06. Quando si crea, si modifica o si cancella una qualsiasi struttura tabelle del catalogo che ne contengono la descrizione e le relazioni con le altre tabelle vengono aggiornate.

Poiche il catalogo può essere esplorato con semplici query SQL, quello che la funzione *getAllDatabase()* dovrebbe fare per recuperare la lista dei database sul server è connettersi al database DSNDB06 ed eseguire la query "`select name from SYSIBM.SYSDATABASE`", dove SYSIBM.SYSDATABASE è la tabella del catalogo contenente informazioni su ogni database nel sistema. Tale funzione è stata implementata ma non è stato possibile testarne il funzionamento non avendo a disposizione un server DB2 Universal Database per z/OS.

DB2 Universal Database per Linux, Unix, Windows [23] è invece organizzato in maniera diversa. Ogni istanza di DB2 può avere più database che sono però "isolati" tra di loro, di fatto non esiste un catalogo di sistema ma ogni database ha il proprio catalogo che descrive la struttura logica e fisica dei dati memorizzati sul database, tale catalogo è formato da tabelle che contengono informazioni sugli oggetti del database quali tabelle, viste, indici e le autorizzazioni degli utenti su questi oggetti. Il catalogo nasce nel momento stesso in cui viene creato il database e viene aggiornato automaticamente con il normale uso del server. Per avere l'elenco dei nomi dei database su un' istanza di DB2 è reso disponibile il comando "`list database directory`". Tale comando va a recuperare questa lista sul *system database directory file* "`sqldbdir`" contenuto nella sottodirectory omonima della home directory dell' istanza, da distinguere dal *local database directory file* definito in ogni percorso (o drive) nel quale è definito un database. Purtroppo "list database directory" non può essere eseguito con le API JDBC [12] ne è possibile leggere dai file sul server remoto, quindi attualmente non è possibile implementare la funzione *getAllDatabase()* per DB2 Universal Database per Linux, Unix, Windows.

5.2 La classe *OracleWrapperGUI*

L'interfaccia utente del wrapper per Oracle è la stessa dei wrapper per DB2 e per SQLServer. È necessario implementare la funzione *String generateJDBCConnectionString()* per far si che combini i parametri inseriti dall'utente in modo tale da ottenere la stringa necessaria al collegamento in una forma predefinita, che nel caso del driver *JDBC Thin client-side* è:

```
jdbc:oracle:<drivertype>:<user>/<password>@<host>[:<port>]//<service>
```

Per quanto riguarda l'implementazione della funzione ***getAllDatabase()*** non è stato trovato alcun modo per recuperare i nomi dei database da un server Oracle. Infatti quest'informazione non è accessibile da remoto poichè nei sistemi Unix tale lista è memorizzata nel file /etc/oratab (/var/opt/oracle se sistemi Solaris) [25] e nei registri di sistema in sistemi Windows in

HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE\KEY_*HOME_NAME*\ [26].

Inizialmente si era pensato alle tabelle DBA_DB_LINKS, ALL_DB_LINKS e USER_DB_LINKS, ma contengono solo i links ad altri database che un'amministratore può concedere agli utenti.

6 Conclusioni

I wrapper DB2/ODL$_{I3}$ e Oracle/ODL$_{I3}$ sono stati realizzati con successo, anche per merito della classe WrapperJdbcCore_SQLServer già esistente, dalla quale ho riciclato parte del codice. Tra gli obiettivi non raggiunti figura invece la mancata implementazione della funzione *getAllDatabase()* per il recupero di una lista dei database su un server nelle classi GUI. D'altro canto non esiste modo attendibile per effettuare tale operazione nel caso di Oracle e nel caso di DB2 Universal Database per Linux,Unix,Windows.

Prima di creare il database di figura 6, si è cercato di estrarre un database che opzionalmente crea Oracle come esempio. L'esito di questo tentativo non è stato positivo a causa dei tipi di dato non supportati da ODL$_{I3}$ (figura 5), si suggerisce quindi di prendere in considerazione il completamento di questo aspetto per non precludere la possibilità di integrare alcune sorgenti. Un altro miglioramento che si potrebbe attuare è considerare anche i valori di default come metadato utile all'integrazione di sorgenti.

L'uso delle risorse sul web è stato di notevole importanza per lo svolgimento di questa tesi. In particolare è stato fatto uso di forum [30] per la ricerca di informazioni sul reperimento dei nomi di tutti i database su server DB2 e Oracle. Tra tutti i forum consultati gli unici validi sono stati:

- **Oracle DBA Forum [31]**
  utilissimo per ottenere consulenze sui sistemi Oracle da specialisti; ho subito ricevuto risposta da esperti sul topic da me inserito consultabile all'indirizzo http://dba.ipbhost.com/index.php?showtopic=1767
- **DUGI Listserv [32]**
  gestito dal ***DB2 User Group Italia(DUGI)*** permette di iscrivere la propria e-mail a un servizio per comunicare con tutti gli altri iscritti; su listserv ho ricevuto risposta sul problema del reperimento della lista dei database.

Naturalmente tutte le informazioni ricevute sono state confermate dalle documentazioni ufficiali.

Questa esperienza mi è stata utile per conoscere, anche se non molto a fondo, due tra i DBMS commerciali più diffusi quali sono DB2 [10] e Oracle [11]. Inoltre ho approfondito la conoscenza di JDBC [12], le API di JAVA dedicate allo sviluppo di applicazioni per l'accesso a database.

APPENDICE A

Mapping tra i datatypes Oracle_SQL e JDBC [27]

SQL Datatypes	JDBC Typecodes
	STANDARD JDBC 1.0 TYPES:
CHAR	java.sql.Types.CHAR
VARCHAR2	java.sql.Types.VARCHAR
LONG	java.sql.Types.LONGVARCHAR
NUMBER	java.sql.Types.NUMERIC
NUMBER	java.sql.Types.DECIMAL
NUMBER	java.sql.Types.BIT
NUMBER	java.sql.Types.TINYINT
NUMBER	java.sql.Types.SMALLINT
NUMBER	java.sql.Types.INTEGER
NUMBER	java.sql.Types.BIGINT
NUMBER	java.sql.Types.REAL
NUMBER	java.sql.Types.FLOAT
NUMBER	java.sql.Types.DOUBLE
RAW	java.sql.Types.BINARY
RAW	java.sql.Types.VARBINARY
LONGRAW	java.sql.Types.LONGVARBINARY
DATE	java.sql.Types.DATE
DATE	java.sql.Types.TIME
TIMESTAMP	java.sql.Types.TIMESTAMP
	STANDARD JDBC 2.0 TYPES:
BLOB	java.sql.Types.BLOB
CLOB	java.sql.Types.CLOB
user-defined object	java.sql.Types.STRUCT
user-defined reference	java.sql.Types.REF
user-defined collection	java.sql.Types.ARRAY
	ORACLE EXTENSIONS:
BFILE	oracle.jdbc.OracleTypes.BFILE
ROWID	oracle.jdbc.OracleTypes.ROWID
REF CURSOR type	oracle.jdbc.OracleTypes.CURSOR
TIMESTAMP	oracle.jdbc.OracleTypes.TIMESTAMP
TIMESTAMP WITH TIME ZONE	oracle.jdbc.OracleTypes.TIMESTAMPTZ
TIMESTAMP WITH LOCAL TIME ZONE	oracle.jdbc.OracleTypes.TIMESTAMPLTZ

APPENDICE B

Mapping tra i datatypes DB2_SQL e JDBC [28]

JDBC data type	SQL data type
BIT	SMALLINT
TINYINT	SMALLINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
REAL	REAL
FLOAT	REAL
DOUBLE	DOUBLE
NUMERIC	DECIMAL
DECIMAL	DECIMAL
CHAR	CHAR
CHAR	GRAPHIC
VARCHAR	VARCHAR
VARCHAR	VARGRAPHIC
LONGVARCHAR	VARCHAR
VARCHAR	CLOB(*n*)
LONGVARCHAR	CLOB(*n*)
CLOB	CLOB(*n*)
BINARY	CHAR FOR BIT DATA
VARBINARY	VARCHAR FOR BIT DATA
LONGVARBINARY	VARCHAR FOR BIT DATA
VARBINARY	BLOB(*n*)
LONGVARBINARY	BLOB(*n*)
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
BLOB	BLOB
CLOB	CLOB
CLOB	DBCLOB
None	BLOB(*n*)
None	CLOB(*n*)
None	CLOB(*n*)
com.ibm.db2.jcc.DB2Types.ROWID	ROWID
DATALINK	DATALINK

APPENDICE C

Mapping tra ANSI SQL datatypes e Oracle_SQL Datatypes [29]

ANSI SQL Datatype	Oracle SQL Datatype
CHARACTER(n)	CHAR(n)
CHAR(n)	
CHARACTER VARYING(n)	VARCHAR(n)
CHAR VARYING(n)	
NATIONAL CHARACTER(n)	NCHAR(n)
NATIONAL CHAR(n)	
NCHAR(n)	
NATIONAL CHARACTER VARYING(n)	NVARCHAR2(n)
NATIONAL CHAR VARYING(n)	
NCHAR VARYING(n)	
NUMERIC(p,s)	NUMBER(p,s)
DECIMAL(p,s)[a]	
INTEGER	NUMBER(38)
INT	
SMALLINT	
FLOAT(b)[b]	NUMBER
DOUBLE PRECISION[c]	
REAL[d]	

APPENDICE D

Mapping tra i tipi ODL$_{I3}$/JDBC/DB2 SQL/Oracle SQL

ODL$_{I3}$ datatypes	**JDBC datatypes**	**DB2 SQL datatypes**	**Oracle SQL datatypes**
IntegerType(true,true)	BIGINT	BIGINT	NUMBER
IntegerType(false,true) (Short)	INTEGER	INTEGER	NUMBER
	SMALLINT	SMALLINT	NUMBER
	TINYINT	SMALLINT	NUMBER
BooleanType	BIT	SMALLINT	NUMBER
CharType	CHAR	CHAR GRAPHIC	CHAR
DateType	DATE	DATE	DATE
	TIME	TIME	DATE
	TIMESTAMP	TIMESTAMP	TIMESTAMP
FloatingType(false) (Float)	DECIMAL	DECIMAL	NUMBER
	FLOAT	REAL	NUMBER
FloatingType(true) (Double)	DOUBLE	DOUBLE	NUMBER
	NUMERIC	DECIMAL	NUMBER
	REAL	REAL	NUMBER
StringType	LONGVARBINARY	VARCHAR FOR BIT DATA	LONGRAW
	LONGVARCHAR	CLOB(n)	LONG
	VARBINARY	BLOB(n)	RAW
	VARCHAR	CLOB(n)	VARCHAR2

RIFERIMENTI BIBLIOGRAFICI

[1] D.Beneventano, S.Bergamaschi, A.Corni, M.Vincini
"Creazione di una vista globale d'impresa con il sistema MOMIS basato su Description Logics"
Article for the AIIA journal on the for Gestione d'impresa 2000

[2] Giovanni Lorandini
"Progetto MOMIS: il wrapper Access/ODLi3"
Tesi di laurea

[3] Gianni Pio Grifa
"Analisi di Affinita' Strutturali fra Classi ODL(I3) nel Sistema MOMIS"
Tesi di laurea

[4] http://dbgroup.unimo.it/Momis/prototipo/modules/wrappers/

[5] http://dbgroup.unimo.it/ODB-Tools.html

[6] http://dbgroup.unimo.it/Momis/

[7] http://dbgroup.unimo.it/Momis/prototipoPub/modules/SIDesigner/

[8] http://dbgroup.unimo.it/Sewasie/

[9] http://www.isse.gmu.edu/I3_Arch/

[10] http://www-306.ibm.com/software/data/db2/

[11] http://www.oracle.com/database

[12] http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/

[13] http://www.sewasie.org/

[14] http://islab.dico.unimi.it/artemis/index.php

[15] http://wordnet.princeton.edu/

[16] http://www.omg.org/gettingstarted/corbafaq.htm

[17] http://www.omg.org/

[18] D.Beneventano, S. Bergamaschi, A. Corni, R. Guidetti, G. Malvezzi
"SI-Designer un tool di ausilio all'integrazione intelligente di sorgenti di informazione"
SEBD Sistemi Evoluti di Basi di Dati L'Aquila, Italy 2000

[19] JAVA 2 SDK, Standard Edition Documentation Version 1.4.2

[20] http://www.oracle.com/technology/tech/java/sqlj_jdbc

http://download-west.oracle.com/docs/cd/B14117_01/appdev.101/b10795/adfns_en.htm#1013330

[21] download-west.oracle.com/otn_hosted_doc/toplink/1013/MAIN/_html/optimiz004.htm

[22] publib.boulder.ibm.com/infocenter/db2help

[23] http://www-306.ibm.com/software/data/db2/udb/support/manualsv8.html
SQL Reference, Vol. 1

[24] http://www-306.ibm.com/software/data/db2/zos/v8books.html
SQL Reference

[25] http://download-west.oracle.com/docs/html/B10811_01/ch6.htm#sthref1822

[26] http://download-west.oracle.com/docs/cd/B14117_01/win.101/b10113/registry.htm#sthref844

[27] http://download-west.oracle.com/docs/cd/B14117_01/java.101/b10979/basic.htm#sthref238

[28] publib.boulder.ibm.com/infocenter/db2help
Developing>Database application>Programming application>
>Java>Java, JDBC, and SQL data types

[29] http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10759/sql_elements001.htm#i54335

[30] http://www.tek-tips.com
http://www-106.ibm.com/developerworks/forums/dw_jforums.jsp
http://www.dbforums.com
http://forums.oracle.com/forums/

[31] http://dba.ipbhost.com

[32] http://www.gruppo.technites.it/db2ug/home.htm

[33] http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10743/datadict.htm

[34] http://www.isse.gmu.edu/I3_Arch/X0008_7.WrappingServices.html