ai nonni Giuseppe e Rosario

In ogni cosa ho voglia di arrivare
sino alla sostanza.
Nel lavoro, cercando la mia strada,
nel tumulto del cuore.
Sino all'essenza dei giorni passati,
sino alla loro ragione,
sino ai motivi, sino alle radici,
sino al midollo.
Eternamente aggrappandomi al filo
dei destini, degli avvenimenti,
sentire, amare, vivere, pensare
effettuare scoperte.

Boris Pasternak

RINGRAZIAMENTI

Un sincero ringraziamento alla Prof.ssa Sonia Bergamaschi per l'interesse e la disponibilità dimostrati durante il lavoro di tesi.

Ringrazio l'Ing.Mirko Orsini, l'Ing.Antonio Sala, l'Ing.Alberto Corni per i Loro consigli, che hanno facilitato la realizzazione di questo lavoro.

Un grazie di cuore alla mia famiglia che mi ha sempre incoraggiato e sostenuto lungo tutto il mio cammino universitario.

Infine, un sentito grazie a tutti gli amici, in particolare a Mauro, William e Daniele con i quali ho condiviso intensi momenti di vita universitaria.

Indice

Indice delle figure

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 impossibile 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***: utilizzo di modelli diversi per rappresentare informazioni simili.

In questo scenario, fortemente investigato 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*.

I *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) in modo virtuale 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 *I3* [2] dell'ARPA), che si distingue dagli altri modi di integrazione nel fatto che aggiunge valore alla conoscenza acquisita, ottenendo nuove informazioni dai dati ricevuti.

In generale, l'integrazione dell'informazione [3] è concepita come unificazione di fonti dati eterogenee provenienti da varie sorgenti. Nonostante i significativi sforzi fatti per sviluppare tools e soluzioni per l'automazione del processo di integrazione dati, queste attuali tecniche sono ancora molto manuali e richiedono un intervento significativo dell'utente, tramite programmazione di codice, per il settaggio e raffinamento dell' integrazione, in modo da migliorarne l'efficacia. Spesso la riusabilità del codice, creato per specifiche soluzioni di integrazione, è molto limitata. Questo approccio è in contrasto con la natura del processo di integrazione: ossia tutti i problemi legati all'elevato numero delle sorgenti coinvolte e la dinamicità e variabilità degli schemi di dati ed i loro contenuti. Naturalmente la sfida per il miglioramento dei sistemi di integrazione è un problema che va affrontato a diversi livelli di astrazione: dall'eterogeneità a livello di sistema (ossia dell'hardware), quella di livello strutturale (sintattica, ossia a livello di schema dei dati); considerando anche quella relativa al livello semantico (differenze di uso e significato di dati simili).

L'area di ricerca relativa all'integrazione dell'informazione è attiva sin dai primi anni '90, ed ha portato allo sviluppo di nuove tecniche e approcci; ma il compito più difficile è determinare la qualità e l'applicabilità di queste soluzioni; ciò infatti è obiettivo di molti studi ed uno dei principali limiti è che i dati utilizzati nei test sono poco ricchi e di volumi inadeguati per permettere un'adeguata e significativa valutazione del sistema di integrazione.

In questa tesi si è analizzato il modo in cui vengono affrontati i problemi dell'integrazione dell'informazione e di creazione di *datawarehouse* dei principali sistemi di integrazione dati presenti sul mercato:

- IBM WebSpher Information Integrator Content Edition [4];
- Oracle Data integrator [5];
- Microsoft SQL Server Integration Services [6];

Fra questi sistemi solo quello di IBM è un data integration system con approccio virtuale. Inoltre è stata effettuata un'analisi ed una valutazione comparativa, tra questi sistemi, ed il progetto MOMIS [7] (sviluppato dal gruppo di ricerca DBGroup [8] dell'università di Modena) attraverso il benchmark THALIA [9]. THALIA fornisce una raccolta di 25 fonti di dati diverse (sotto forma di file XML) che rappresentano i cataloghi dei corsi di diverse università di ingegneria informatica sparse nel mondo. Inoltre prevede un set di 12 query per il test del sistema di integrazione dati, ciascuna delle quali si prefigge l'obiettivo di testare come, effettivamente, il sistema di integrazione reagisce a particolari eterogeneità sintattiche e semantiche.

Lo scopo della presenti tesi è stato quello di analizzare le varie procedure di integrazione dei sistemi considerati e valutarle tramite l'implementazione delle query proposte dal benchmark THALIA al fine di effettuare un confronto generale tra i sistemi e scoprire come questi risolvano i vari problemi annessi ai processi di integrazione.

La tesi è articolata nel modo seguente:

Nel **Capitolo 1** viene presentata l'introduzione al benchmark THALIA ed alle sorgenti dati utilizzate.

Nel **Capitolo 2** sono analizzati i sistemi di integrazione IBM WebSpher Information Integrator Content Edition, Oracle Data Integrator e Microsoft SQL Server Integration Services, con le rispettive implementazioni delle query del benchmark su ogni sistema.

Nel **Capitolo 3** viene analizzato il progetto MOMIS e l'implementazione delle query del benchmark con tale sistema.

Nel **Capitolo 4** vengono analizzati i risultati ottenuti dall'applicazione del benchmark sui sistemi considerati in questo lavoro di tesi, e sviluppata un'analisi comparativa tra il sistema MOMIS ed i sistemi commerciali trattati.

Infine, vengono esposte le **conclusioni** di questa tesi.

Capitolo 1

Benchmark THALIA

1.1 Introduzione al benchmark e alle sorgenti dati

In questa capitolo verrà specificato lo strumento, di pubblico dominio, utilizzato come benchmark e test per la valutazione dei sistemi di integrazione dati presi in esame in questo lavoro di tesi: THALIA (Test Harness for the Assessment of Legacy information Integration Approaches). THALIA fornisce una raccolta di 25 fonti di dati diverse (sotto forma di file XML) che rappresentano i cataloghi dei corsi di diverse università di ingegneria informatica sparse nel mondo. Inoltre prevede un set di 12 query per il test del sistema di integrazione dati, ciascuna delle quali si prefigge l'obiettivo di testare come, effettivamente, il sistema di integrazione reagisce a particolari eterogeneità sintattiche e semantiche. Naturalmente, gli ideatori di questo benchmark hanno considerato le eterogeneità di rilevanza maggiore, in quanto, queste, mettono in luce i problemi principali che ancora oggi molti sistemi di integrazione non risolvono al meglio. Infine, THALIA, prevede una funzione di valutazione per il calcolo del punteggio, al fine di ottenere una classificazione finale dei vari sistemi di integrazione presi in esame. Il benchmark si focalizza sull'eterogeneità sintattiche e semantiche, croce e delizia dei sistemi di integrazione, in quanto queste sono oggetto di continue sfide tecniche per il miglioramento dei sistemi di integrazione stessi. THALIA non vuol essere solo un semplice strumento di valutazione, ma si prefigge di offrire un contributo importante ai ricercatori, al fine di migliorare l'accuratezza e la qualità dei futuri approcci alle tecniche di integrazione, attraverso un analisi completa e specializzata per queste tipologie di eterogeneità. THALIA oltre a fornire i dati con i quali provare il proprio sistema di integrazione, fornisce anche, come detto in precedenza, 12 query per il test: ed è proprio questo che distingue THALIA, dagli altri sistemi di valutazione esistenti. Le 25 sorgenti dati si riferiscono ad altrettanti dipartimenti universitari, ed offrono un'abbondante e pubblica risorsa per il test. I cataloghi delle rispettive università sono accessibili tramite il sito ufficiale [10] di THALIA, e permettono di avere

una visione completa di come può cambiare la rappresentazione dell'informazione, da un'università all'altra. In Figura 1 è rappresentato il catalogo dei corsi dell'università di Brown, con le relative informazioni su numero corso, nome corso, professore, orario, luogo):

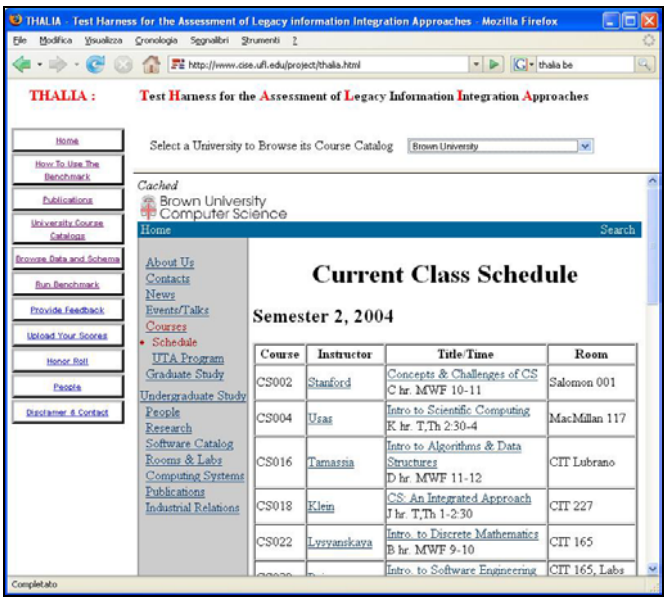


Figura 1. Snapshot del catalogo dei corsi dell'università di Brown

I dati relativi a tutti i cataloghi delle varie università vengono forniti sottoforma di file XML, uno per ogni catalogo relativo ad una specifica università. L'estrazione dell'informazione dal formato di ogni catalogo universitario e la trasformazione sotto forma di file XML è stata fatta tramite uno specifico wrapper: è stato usato il sistema Telegraph Screen Scraper (TESS) [11], che rimuove le eterogeneità a livello di sistema preservando, però, quelle semantiche e strutturali delle differenti sorgenti dati.

Per ogni fonte, TESS, ha un file di configurazione che specifica quali campi deve estrarre; i marcatori di inizio e fine campo sono identificati usando espressioni regolari. Nella sua prima versione originale, il sistema, estraeva con successo informazioni da cataloghi con semplici strutture, come quella dell'università Brown (Figura 1); il sistema TESS, però, non riusciva ad analizzare cataloghi complessi come quello dell'università di Maryland, mostrata in Figura 2. La combinazione di strutture libere da *form* e con *table* innestati richiese una modifica a TESS, che non era stato progettato per estrarre linee multiple da strutture innestate. Quindi sono state aggiunte informazioni supplementari nei file di configurazione, che specificano la struttura dei campi che contengono sottostrutture.

Quando TESS viene invocato su una fonte dati, per la quale esiste un file di configurazione, il corrispondente file XML è prodotto secondo le specifiche contenute nel file di configurazione. Naturalmente, è chiaro che qualsiasi variazione a livello sintattico di un particolare catalogo, porterà una corrispondente modifica nel file di configurazione di TESS.

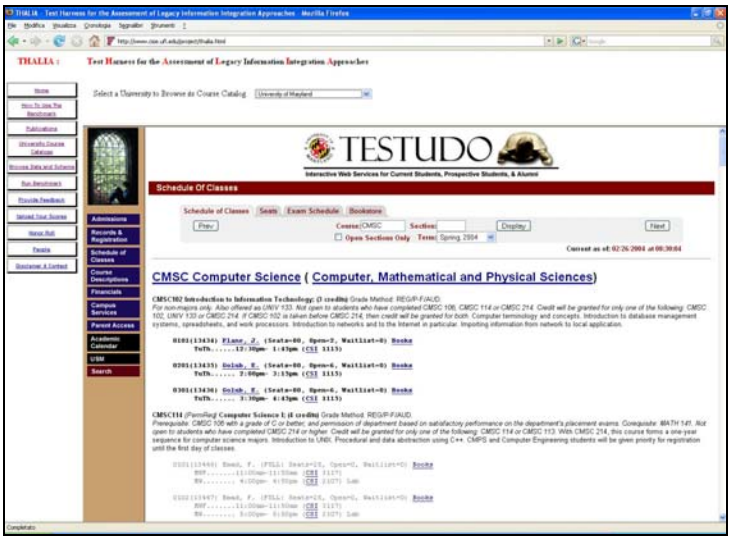


Figura 2. Snapshot del catalogo dei corsi dell'università del Maryland

Come nota finale, si sottolinea il fatto che TESS è in grado di estrarre informazione solo da una pagina web o da un documento, e non è in grado di estrarre contenuti da più pagine web attraverso i link contenuti in queste. Per far ciò, dovrebbero essere implementati più file di configurazione e un nuovo modo di estrarre l'informazione dalle pagine richiamate dai vari link; TESS restituisce solo il link, come un semplice valore di un qualsiasi campo.

Nella Figura 3 è rappresentato l'output XML ed il corrispondente schema per un particolare catalogo (università di Brown); come si può notare, si preservano tutte le eterogeneità semantiche tramite l'estrazione dei nomi degli elementi dello schema originale.

Per quanto riguarda l'esecuzione del benchmark, THALIA fornisce tutti questi file XML e le specifiche delle 12 query attraverso file zip, che si possono scaricare direttamente dal sito ufficiale. Inoltre fornisce anche gli schemi XML risultanti dall'integrazione, per l'esecuzione delle query proposte dal benchmark. Per ogni tipo di eterogeneità, il benchmark propone una serie di query, ed ognuna di queste agisce su due sorgenti dati diverse: uno schema di riferimento (reference schema) ed uno schema "*sfida*" (challenge schema), che contiene il tipo di eterogeneità da risolvere per il

sistema di integrazione. Proponendo queste 12 query al sistema di integrazione da analizzare, è possibile osservare come quest'ultimo reagisce ai vari tipi di eterogeneità semantiche, sintattiche e strutturali e di conseguenza apportare le giuste modifiche al mapping degli attributi dei vari schemi relativi alle varie sorgenti dati, quindi apportare i giusti raffinamenti.

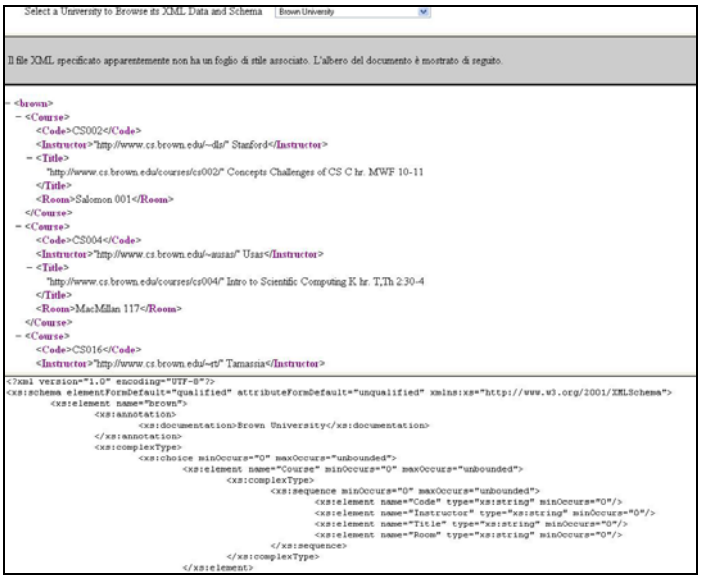


Figura 3. Rappresentazione XML del catalogo dell'università di Brown

1.2 Le 12 query del benchmark

In questa sezione si descriverà brevemente ogni tipo di eterogeneità e la relative query associata dal benchmark, includendo l'esempio dei dati sui quali questa va ad agire, ossia *reference schema* e *challenge schema*. Le dodici querce sono state raggruppate per il tipo di eterogeneità comune, ed in particolare vengono individuati tre gruppi principali: eterogeneità degli attributi, mancanza dei dati, eterogeneità strutturali.

Eterogeneità degli attributi: esistente tra due attributi correlati presenti in schemi differenti, si distinguono 5 casi:

- ***Sinonimi (query 1):*** Attributi di nome diverso che hanno lo stesso significato. Per esempio, "instructor" o "lecturer."

Benchmark Query: Lista dei corsi tenuti dal professore "Mark"
Reference University: Georgia Tech University
Challenge University: Carnegie Mellon University
Obiettivo: Determinare che nel catalogo dei corsi di CMU l'infomrazione su professore può essere trovata in un campo "Lecturer".

- ***Semplice mapping (query 2)****:* attributi correlati presenti in schemi differenti differiscono per una trasformazione matematica dei loro valori. Per esempio, l'orario può essere rappresentato in vaoli 0-24 oppure 0-12

  *Benchmark Query***:** Trovare tutti i corsi che iniziano alle ore 1:30 pm, in tutti i database.
  Reference University: Carnegie Mellon University
  Challenge University: University of Massachusetts
  Obiettivo: Conversione della rappresentazione dell'orario.

- ***Tipi unione (query 3):*** Attributi in schemi differenti utilizzano diversi tipi di dati per rappresentare la stessa informazione. Per esempio, la rappresentazione della descrizione di un corso può essere una semplice stringa, oppure un tipo di dato composto da una stringa ed un link URL relativo a dati esterni.

  Benchmark Query: Trovare tutti I corsi che nel loro titolo contengono la stringa "Data Structures"
  Reference University: University of Maryland
  Challenge University: Brown University
  Obiettivo: mappare un semplice attributo stringa come combinazione di stringa e link URL, visto come stringa. Inoltre, questa query, prevede anche un'eterogeneità di sinonimi (CourseName vs. Title).

- ***Mapping complesso (query 4)****:* Attributi correlati per una trasformazione complessa dei propri valori; la trasformazione non può essere sempre computabile a priori. Per esempio, l'attributo "Units" rappresenta il numero di lezioni per settimana, oppure un attributo "credits" esprime con descrizione testuale il carico di lavoro.

Benchmark Query: Lista di tutti i corsi che hanno più di 10 crediti in ore.
Reference University: Carnegie Mellon University
Challenge University: Swiss Federal Institute of Technology Zürich (ETH)
Obiettivo: Separatamente dai problemi della conversione di lingua, la sfida è sviluppare un mapping che estragga il valore numerico di crediti (in ore) dalla stringa che descrive il carico di lavoro (tedesco: "Umfang") del corso.

- ***Language Expression (query 5):*** I nomi o i valori di attributi che esprimono lo stesso significato sono espresso in differenti linguaggi. Per esempio, il termine inglese "Database" è chiamato "Datenbank" in lingua tedesca.

  Benchmark Query: Find all courses with the string 'database' in the course title.
  Reference University: University of Maryland
  Challenge University: Swiss Federal Institute of Technology Zürich (ETH)
  Obiettivo: Per ogni corso nel catalogo di ETH, convertire il termine Tedesco nel corrispondente termine inglese per l'attributo nome corso. Convertire il termine inglese 'Database' nel termine tedesco 'Datenbank' o 'Datei' o 'Datenbasis' e restituire i corsi ETH che contengono queste sottostringhe.

Dato mancante: derivano dalla mancanza di informazioni (struttura o valore) in uno dei due schemi.

- ***Valori nulli (query 6):*** l'attributo (valore) non esiste. Per esempio, alcuni corsi non hanno il campo "textbook" o il valore per questo campo è vuoto.

  Benchmark Query: Lista dei corsi che hanno verifica orale.
  Reference University: University of Toronto
  Challenge University: Carnegie Mellon University
  Obiettivo: Trattamento dei valori nulli. Per questa query, il risultato derivante dall'integrazione dovrà specificare che, di fatto, non ci sono informazioni su textbook per i corsi CMU.

- ***Attributi virtuali (query 7):*** L'informazione può essere rappresentata in maniera esplicita in uno schema, mentre può essere implicita in un altro, e può essere inferita da uno o più valori. Per esempio ci può essere il campo relativo ai "prerequisiti" di un corso in uno schema, ma può esistere sottoforma di commento differente attributo in un altro schema.

  Benchmark Query: Trovare i corsi che presentano prequisiti d'ingresso.
  Reference University: University of Michigan
  Challenge University: Carnegie Mellon University
  Obiettivo: Riuscire ad estrarre l'informazione sui prerequisiti di un corso che è presente come commento nell'attributo "nome" del corso.

- ***Semantic incompatibility (query 8):*** un concetto reale che è modellato tramite un attributo in uno schema, non esiste invece in un altro schema. Per esempio, la classificazione di studente ('freshman', 'sophomore', etc.) nell'università americana non esiste in quella tedesca.

  Benchmark Query: Lista di tutti I corsi che sono permessi a student "junior".
  Reference University: Georgia Tech
  Challenge University: The Swiss Federal Institute of Technology Zurich, ETH
  Obiettivo: Anche se si può ritornare un valore nullo dalla ricerca su ETH, la risposta può essere fuorviante. Per trattare intelligentemente questo tipo di query è bene distinguere tra diversi tipi di valori NULL che possono essere ritornati. Nello specifico, è bene distinguere i casi: "il dato non c'è ma può essere presente" (vedi query 6), da quello in cui "il dato non c'è e non può essere presente".

Eterogeneità strutturali: derivano dal modo in cui l'informazione può essere modellata o rappresentata nei differenti schemi.

- ***Stesso attributo in strutture differenti (query 9):*** *l*o stesso attributo può essere presente in posizioni diverse nello stesso schema. Per esempio, l'attributo "Room" fa parte dello schema "corso" in una sorgente dati, mentre è un attributo dello schema "sezione" in un'altra.

Benchmark Query: trovare tutte le aule nelle quali si tengono i corsi "Database".
Reference University: Brown University
Challenge University: University of Maryland
Obiettivo: Determinare che l'informazione sull'aula, nel catalogo dei corsi dell'università del Maryland, è disponibile come parte di informazione contenuta nell'attributo "time" nello schema "Section" .

- ***Trattamento di set di valori (query 10):*** in uno schema un attributo può rappresentare un set di valori, mentre in un altro schema questo set di valori può essere contenuto in diversi attributi organizzati gerarchicamente. Per esempio, un corso con più professori può avere un singolo attributo che li specifica tutti, oppure avere più attributi che raccolgono le informazioni dei vari professori.

  Benchmark Query: Lista dei professori che tengono corsi di "software systems".
  Reference University: Carnegie Mellon University
  Challenge University: University of Maryland
  Obiettivo: Determinare che l'informazione sul professore è memorizzata nello schema "Section" per l'università del Maryland. Nello specifico, l'informazione sul professore deve essere garantita tramite l'estrazione di quest'ultima come parte dell' attributo "title" dello schema sezioni, mentre l'informazione del professore è contenuta in un singolo attributo nel catalogo dei corsi dell'università di CMU.

- ***Il nome dell'attributo non ne descrive la semantica (query 11)*****:** il nome dell'attributo non descrive adeguatamente la semantica del valore che memorizza al suo interno.

  Benchmark Query: lista dei professori per i corsi "Database".
  Reference University: Carnegie Mellon University
  Challenge University: University of California, San Diego
  Obiettivo: nel caso del catalogo dei corsi dell'università di San California, San Diego, bisogna associare, nel mapping, gli attributi "Fall 2003", "Winter 2004" etc. con l'informazione sul professore.

- ***Composizione di attributi (query 12):*** La stessa informazione può essere rappresentata tramite un singolo attributo o tramite un set di attributi, possibilmente organizzati in maniera gerarchica.

  Benchmark Query: lista dei nomi e dell'orario dei corsi di reti di calcolatori.
  Reference University: Carnegie Mellon University
  Challenge University: Brown University
  Obiettivo: Determinare che l'informazione sul nome, giorno e ora del corso nel catalogo dell'università di Brown è rappresentata come parte dell'attributo "title" invece che in separati attributi come in CMU. Estrarre i valori del nome, giorno e ora dall'attributo "title" nel catalogo dell'università di Brown.
  L'insieme di queste 12 query non presentano in maniera esaustiva tutti i tipi di eterogeneità che si possono trovare in un dominio applicativo. Comunque, i casi presi in considerazione da questo benchmark ricoprono la maggior parte e le principali eterogeneità discusse in letteratura e le query rappresentano un ragionevole sottoinsieme di eterogeneità che, spesso, devono essere risolte dai vari sistemi di integrazione.

Capitolo 2

Sistemi di integrazione dati/ETL commerciali

2.1 Architettura WebSphere Information Integrator Content Edition 8.4

WebSphere Information Integrator Content Edition 8.4 (IICE) è il software di integrazione sviluppato da IBM, sviluppato interamente in Java per le aziende *businesses*. Questo software ha la capacità di integrare le applicazioni enterprise con varie tipologie di contenuti, come documenti, immagini, audio, video, dati strutturati e semi-strutturati memorizzati su più ambienti eterogenei, tipici proprio di queste aziende. Per utilizzare tutte le funzionalità ed i servizi di questo sistema di integrazione è necessario distribuirlo su un server delle applicazioni; per questo lavoro di tesi è stato utilizzato IBM WebSphere Application Server 6.1(WAS) [12] in quanto la configurazione della distribuzione dell'IICE sul WAS avviene tramite una procedura eseguita in automatico durante la fase di installazione dell'IICE. La scelta è stata effettuata in previsione di una maggiore facilità di installazione dei due prodotti IBM e compatibilità; infatti, utilizzando un server delle applicazioni diverso, la procedura di configurazione della distribuzione dell'IICE sull'application server deve essere programmata manualmente dall'utente.

L'architettura [13] (vedi Figura 4) di WebSphere Information Integrator Content Edition consiste di tre livelli principali: servizi per sviluppatore e utente, servizi di associazione e servizi di integrazione.

I servizi per lo sviluppatore e per l'utente includono i componenti utilizzati per visualizzare e utilizzare il contenuto. Questo livello include anche le API (Application Programming Interface) e le tecniche disponibili per la generazione di nuove applicazioni che consentono di visualizzare e utilizzare il contenuto:

- API dei Servizi Web che includono gran parte delle API WebSphere Information Integrator Content Edition tramite un'interfaccia SOAP. L'API

dei Servizi Web include un file WSDL (Web Services Description Language) che definisce in modo completo l'API e fornisce un metodo indipendente dal linguaggio di accedere al contenuto gestito, non strutturato in Internet e tramite firewall con un client che non richiede spazio su disco.

- API di integrazione che possono essere utilizzate per creare le applicazioni di gestione e integrazione del contenuto tramite un solo insieme di interfacce orientate agli oggetti. Espone un insieme comune di funzioni offerte dai repository di gestione del contenuto e dai sistemi di flusso di lavoro.
- Indirizzabilità URL e accesso http. In molte configurazioni, il server di integrazione del contenuto e i computer client potrebbero essere ubicati a migliaia di chilometri di distanza dai computer su cui è installato il repository nativo. Tale distanza può causare problemi di latenza. L'indirizzabilità URL e l'accesso HTTP forniscono metodi più veloci di richiamare il contenuto dai repository nativi.

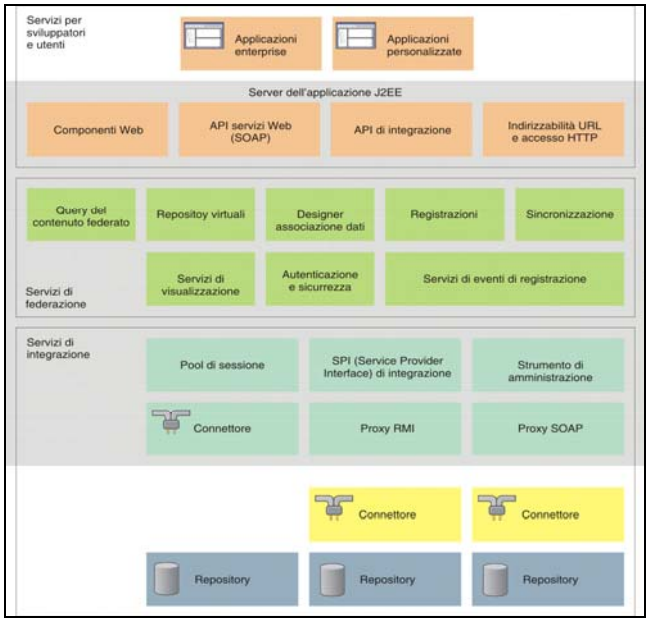


Figura 4. Architettura IBM WebSphere IICE 8.4

I servizi di federazione, invece, comprendono funzioni che combinano e aggiungono valore al contenuto integrato:

- **Query federate**, un insieme di opzioni di ricerca da utilizzare per individuare il contenuto in un gruppo di repository nativi;
- **Repository virtuali**, un insieme di funzioni API che gli utenti possono utilizzare per creare e salvare le ricerche e le viste del contenuto federato. Gli utenti

possono creare intere strutture di directory per organizzare il contenuto federato in conformità a processi e pratiche aziendali. I repository virtuali non richiedono che il contenuto sia replicato, ma forniscono una vista personalizzabile del contenuto federato tramite collegamenti dinamici basati su ricerca o diretti. I dati dei repository virtuali vengono salvati nell'archivio dati dell'IICE;

- **Designer delle associazioni dati**, che estende le funzionalità di ricerca collegando elementi di diversi repository che presentano lo stesso tipo di contenuto ma nome diverso. Ad esempio, in un repository gli indirizzi possono essere associati al nome "Indirizzo" e in un altro repository al nome "Residenza";
- **Servizi evento di sottoscrizione**, ossia servizi di notifica evento basati sulla sottoscrizione, che consentono di notificare agli utenti le modifiche effettuate a un elemento di repository dotato di indirizzo, ad esempio il contenuto, le query e i flussi di lavoro. I servizi evento di sottoscrizione sono in grado di rilevare le modifiche effettuate utilizzando le API del repository nativo di WebSphere IICE, gli strumenti del repository nativo e i sistemi di flusso di lavoro;
- **Servizi di visualizzazione**, che convertono dinamicamente i documenti e le immagini in formati visualizzabili in un browser Web. Consentono la distribuzione di soluzioni basate sul contenuto in ambienti thin-client senza richiedere visualizzatori o software specifici del fornitore. Un'applet del visualizzatore viene inclusa nei servizi di visualizzazione come opzione per la visualizzazione di immagini dal lato del client, la manipolazione dell'immagine e il supporto delle annotazioni;
- **Autenticazione e sicurezza**, ossia funzionalità che consentono agli utenti di accedere simultaneamente a più repository con un solo insieme di criteri di autenticazione utilizzando un archivio dati integrato o un archivio LDAP;

I servizi di integrazione funzionano in combinazione per fornire un accesso efficace al contenuto dei repository nativi. Gli elementi utilizzati per questo servizio sono:

- *Pool di sessione*, che consente alle applicazioni di riutilizzare le connessioni di repository all'interno di un'applicazione o tra più applicazioni, di limitare le connessioni di repository utilizzate da un'applicazione durante l'attività di picco, cancellare le connessioni residue stabilite delle applicazioni client che terminano in modo imprevisto;

- *Strumento di amministrazione*, un'applicazione grafica che consente di configurare i componenti dell'IICE, ad esempio i connettori, i servizi evento di sottoscrizione ecc. Con lo strumento di amministrazione è anche possibile impostare le preferenze di accesso;
- *Connettori*, adattatori che convertono le chiamate dell'API in chiamate dell'API specifica del repository. Questa conversione consente alle applicazioni WebSphere IICE e alle applicazioni personalizzate di utilizzare il contenuto del repository senza interruzioni. Attualmente sono disponibili numerosi connettori ed è incluso un framework di generazione connettori che consente di generare ulteriori connettori;
- *SPI di integrazione*, che fornisce un framework che può essere utilizzato dagli sviluppatori per personalizzare i connettori e per generare nuovi connettori per i repository o altre origini di contenuto.
- *Proxy RMI*, consente la connettività con i repository nativi utilizzando RMI (Remote Method Invocation) Java. RMI consente a un oggetto in esecuzione in una JVM (Java Virtual Machine) di richiamare un metodo su un oggetto in esecuzione in un'altra JVM.
- *Proxy dei Servizi Web*, fornisce la connettività ai repository nativi utilizzando SOAP (Simple Object Access Protocol). Sfruttano SOAP, ad esempio, le applicazioni che accedono alle funzionalità e al contenuto utilizzando un linguaggio diverso dal linguaggio di programmazione Java e gli utenti che accedono alle funzionalità e al contenuto utilizzando un client Microsoft .NET.

WebSphere IICE segue specifici principi di progettazione per mantenere un ambiente in cui gli utenti possano utilizzare il contenuto sia con gli strumenti del repository nativo che con l'IICE. Questi principi guida per la progettazione sono alla base dell'intera architettura:

- ✓ I repository gestiscono autonomamente autorizzazione e sicurezza.Vengono aggiunte ulteriori funzioni di autorizzazione e sicurezza, che non possono però sostituire quelle dei repository nativi. Questo principio consente agli utenti di utilizzare il contenuto con o senza l'IICE preservando la sicurezza;
- ✓ Le funzionalità del repository di base non vengono mai integrate o migliorate. Tutti i repository hanno funzioni e limitazioni diverse relative a ricerca, memorizzazione ecc. WebSphere Information Integrator Content Edition non compensa tali limitazioni. Ad esempio, nelle query federate è possibile

utilizzare solo le funzioni di ricerca comuni ai repository in cui viene eseguita la ricerca;

- ✓ Il contenuto e i metadati del repository non vengono mai memorizzati dall'IICE. Gli utenti che accedono ai repository con strumenti nativi hanno lo stesso accesso al contenuto e ai metadati degli utenti dell'IICE;
- ✓ Per gli utenti di IICE sono esposti solo i repository astratti. Gli utenti che utilizzano il contenuto devono conoscere la posizione esatta del contenuto utilizzato. Da una prospettiva di sviluppo, non vi sono API specifiche del repository in WebSphere IICE. Ciò consente agli utenti di evitare le diversità di specifici repository richiedendo loro la conoscenza di un solo insieme di comportamenti. Analogamente, per gli sviluppatori è necessario conoscere un solo insieme di API.

2.1.1 Ricerca federata

La ricerca federata è il metodo con il quale è possibile accedere alle varie sorgenti collegate all'IICE, attraverso la formulazioni di query . In pratica, viene specificata una singola query ed inviata al sistema di ricerca federata, questo la traduce nel formato appropriato per ogni sorgente del contenuto federato (le sorgenti dati collegate attraverso i connettori) ed invia a queste le richieste effettuate dalla query principale. I risultati ottenuti dalle varie sorgenti vengono poi aggregati dal sistema federato. La ricerca federata include anche esecuzioni post-ricerca, come l'ordinamento del result set finale della query. I risultati finali vengono presentati all'utente come se provenissero da una solo sorgente. Il meccanismo previsto dall' IICE per eseguire la ricerca federate ed elaborare i risultati non richiede la creazione ed il mantenimento di un indice sui metadati, ma viene sfruttato la capacità di ricerca dei repository stessi, delle varie sorgenti, collegati al sistema federato. Nella ricerca federata i servizi d'accesso ricevono le richieste di ricerca, determinano i repository chiamati in causa dalla query, ed invia le query "personalizzate" alle varie sorgenti dati in parallelo. I risultati vengono poi memorizzati in una "cache dei riusltati finali", lato server, per consentire ulteriori elaborazioni sui dati, post-interrogazione, tramite un meccanismo simile ad un cursore, che scorre le righe del reault set finale.

Un approccio complementare alla ricerca federata è usare una tecnologia di ricerca che crea un indice centralizzato del contenuto integrato e dei metadata che possono essere

ricercati. Questo approccio è implementato in un motore di ricerca sviluppato sempre dall'IBM, WebSphere Information Integrator OmniFind™ Edition [14].

2.1.2 Pannello di amministrazione

Il pannello di amministrazione (vedi Figura 5) del server di integrazione del contenuto viene utilizzato per configurare i server di integrazione del contenuto per l'ambiente specifico. Questo componente è un'applicazione Java autonoma in grado di comunicare direttamente con i componenti del server di integrazione del contenuto. Può funzionare in tre modalità differenti:

- ✓ *Modalità apertura file*, lo strumento di amministrazione viene eseguito senza un server delle applicazioni J2EE e richiama le informazioni di configurazione in locale dal filesystem. In questa modalità non è possibile eseguire svariate funzioni nello strumento di amministrazione.
- ✓ *Modalità connessa*, lo strumento di amministrazione viene eseguito con un server delle applicazioni J2EE e richiama le informazioni di configurazione dal server di configurazione dell'IICE.
- ✓ *Modalità diretta*, lo strumento di amministrazione viene eseguito senza un server delle applicazioni J2EE e richiama le informazioni di configurazione in locale dal filesystem.

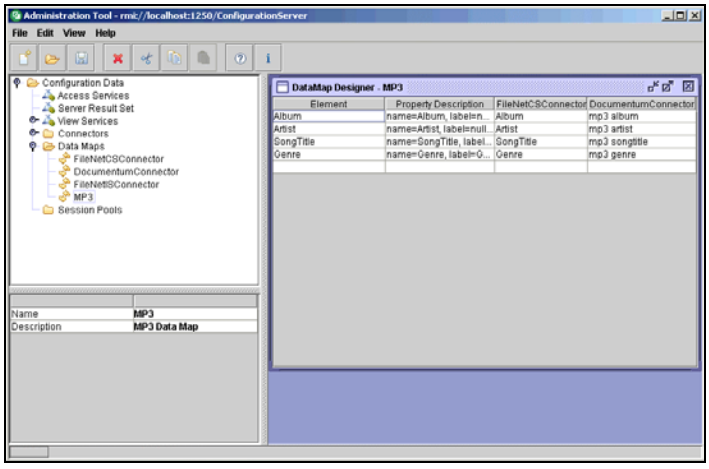


Figura 5. Strumento di amministrazione IICE

Tramite questo pannello è possibile svolgere alcune importanti funzioni come configurare i connettori per collegarsi alle varie sorgenti dati, importare informazioni

delle strutture dati e sui metadati delle sorgenti, creare il dizionario degli attributi globali tramite il Designer associando i dati tra i repository e gestendo gli archivi dati virtuali.

2.1.3 Connettori

I connettori convertono le chiamate alle API di integrazione in chiamate alle API dei repository delle sorgenti. Ciascun connettore deve essere configurato per l'ambiente in cui viene eseguito. Essi consentono alle applicazioni di accedere alle origini di contenuto senza dipendenze specifiche del repository. Vengono forniti connettori per molti repository di contenuto, sistemi di flussi di lavoro, filesystem e sistemi di database. Questi connettori possono anche essere modificati per supportare le diverse implementazioni. È possibile anche sviluppare connettori per repository che non sono ancora supportati. Ciascun repository supportato dispone di un connettore corrispondente che gestisce la comunicazione con quel repository specifico ed ogni connettore presenta proprietà che è necessario impostare utilizzando lo strumento di amministrazione. L'IICE mette a disposizione una serie di connettori per supportare le varie tecnologie presenti sul mercato:

- DB2® Content Manager
- DB2 Content Manager OnDemand
- WebSphere MQ Workflow
- WebSphere Portal Document Manager
- Lotus Notes®
- Lotus Domino® Document Manager
- Documentum
- FileNet® Content Services
- FileNet Images Services
- FileNet ISRA
- FileNet P8 Content Manager
- FileNet P8 Business Process Manager
- OpenText Livelink
- Microsoft Index Server
- Microsoft SharePoint
- TeamSite
- Hummingbird DM
- Relational Database Managemt (RDBMS)

È possibile configurare i connettori da eseguire come *bean enterprise* in esecuzione su un server delle applicazioni J2EE; in modalità diretta se il server di integrazione del

contenuto non è in esecuzione in un server delle applicazioni; attraverso un server del connettore RMI per fornire accesso remoto al connettore; come servizio Web che utilizza SOAP.

Alcuni connettori richiedono la disponibilità della libreria Java del vendor del repository, ossia dei driver specifici per la particolare tecnologia ospitante le sorgenti dati al quale verrà connesso il connettore. Per fornire al connettore accesso alle librerie Java del repository bisogna includere la libreria richiesta nel file *EAR*, VeniceBridge, utilizzando packager *EAR*. Il programma di utilità di packaging *EAR*, fornito con l'IICE, fornisce un modo per revisioare il contenuto del file, aggiungendo nuove librerie. Il packaging *EAR* include alcuni parametri:

- ✓ `-libraryJars`, per specificare nomi file relativi o completi per i file JAR dipendenti che devono essere inclusi. In genere, si specifica *IICE_HOME*/lib/vbr.jar. È anche possibile includere file JAR specifici del connettore. I file JAR inclusi vengono aggiunti all'archivio e specificati nel manifesto e nel descrittore di distribuzione;
- ✓ `-appDirs`, per specificare percorsi relativi delle directory che contengono il bean enterprise e i file *WAR* che si desidera includere;
- ✓ `-updateEJBManifests`, questo parametro facoltativo indica al programma di packaging *EAR* di aggiornare i file manifest del bean enterprise e i file *WAR* in modo da fare riferimento alle librerie fornite (`-libraryJars`) prima di inserirli nel file EAR finale;
- ✓ `-appName`, per specificare il nome del file EAR creato. Il file del programma di utilità di packaging *EAR* utilizza l'estensione "*.ear*".

È possibile utilizzare lo script "`rebuild_ear`" incluso con WebSphere Information Integrator Content Edition per impostare il percorso delle classi Java appropriato prima di eseguire il programma di packaging, e per la generazione del file *EAR* VeniceBridge.

Per il lavoro di tesi sono state utilizzate le funzionalità del connettore RDBMS (Relational Database Management System) che consente al server di integrazione di connettersi ad un sistema RDBMS utilizzando l'API Java Database Connectivity (JDBC).

Utilizzando il connettore RDBMS, è possibile accedere al contenuto strutturato oppure ai metadati memorizzati nei database relazionali presenti nella sorgente alla quale è collegato. Il connettore rende possibile solo l'accesso in sola lettura ed espone le righe

che contengono metadati del contenuto come “elementi”; supporta tabelle, viste o sinonimi. Il contenuto nativo (documenti o immagini) a cui fa riferimento un *ID* (identificativo univoco) esterno oppure un *URL* viene esposto come allegato. È possibile rappresentare il contenuto nativo richiamando il contenuto memorizzato in una colonna RDBMS oppure richiamando il contenuto memorizzato in un filesystem esterno. Per il contenuto memorizzato in una colonna RDBMS, sono supportati i seguenti tipi di dati standard:

- BLOB
- CLOB
- CHAR
- VARCHAR
- LONGVARCHAR
- BINARY
- LONGBINARY

È possibile utilizzare lo strumento di amministrazione per configurare le proprietà del connettore RDBMS per consentire al server di integrazione di accedere a un repository RDBMS. In particolare per la connessione vanno specificate le seguenti proprietà:

- **URL JDBC**, l'URL del database sottostante, nel formato `jdbc:subprotocol:subname`. Ad esempio, `jdbc:odbc:MyDBName`.
- **Classe driver JDBC**, ossia il nome della classe del driver JDBC che si connette all'URL del database.

Il connettore RDBMS utilizza la sicurezza fornita dal sistema del database per accedere al contenuto che è memorizzato nel database. Per il contenuto memorizzato nel filesystem, il connettore utilizza le autorizzazioni dell'utente collegato. Se il connettore non è in esecuzione attraverso il servizio proxy del connettore RMI, il connettore utilizza le autorizzazioni dell'utente che ha avviato la JVM per il servizio proxy connettore RMI.

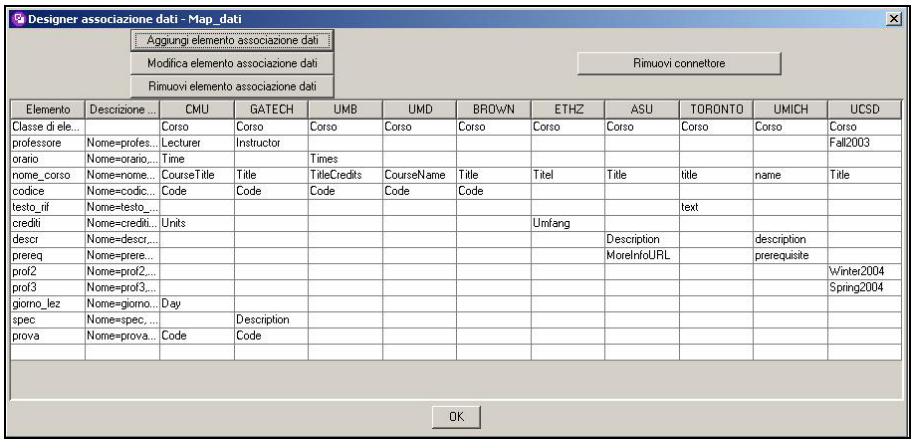
2.1.4 Designer di classi di elementi

Una volta configurato il connettore RDBMS e testato la connessione con la sorgente dati esterna, lo strumento di amministrazione richiederà un nome utente e una password per il database connesso. Terminata la fase di autenticazione è possibile utilizzare l’interfaccia “Designer RDBMS” per creare e definire una classe di elementi

basata su una tabella, vista o sinonimo presenti nel database della sorgente dati. Quando si richiama un elenco di nomi di entità è possibile filtrare le entità in base al nome e selezionare una o più colonne contenute nella struttura dati selezionata. Per fare un esempio, se viene selezionata la tabella "Corsi", presente sul database al quale si è collegati, il Designer richiede l'inserimento di un "nome" da associare all'entità selezionata (es. CorsiStudenti) e successivamente tramite un'altra interfaccia "Descrizione proprietà" sarà possibile specificare i valori del contenuto della tabella "Corsi" da aggiungere alla classe di elementi CorsiStudenti.

2.1.5 Associazione dati

Una volta create le classi di elementi, è possibile utilizzare il "Designer Associazione dati" che offre la funzionalità di generare istanze di attributi globali, chiamate "Associazioni dati" sui quali mappare i metadati delle varie sorgenti. In pratica dà la possibilità di creare un dizionario dei dati, permettendo il mapping dei diversi schemi, di ogni sorgente, in un modello comune di dati. Le associazioni di dati consentono, inoltre, di eseguire le query su più repository (query federate) utilizzando nomi di elemento comuni. Ogni associazione di dati (vedi Figura 6) include le proprietà per i diversi connettori e le classi di elementi contenute nel mapping associazioni di dati. Si possono creare più istanze di associazioni dati.



Elemento	Descrizione ...	CMU	GATECH	UMB	UMD	BROWN	ETHZ	ASU	TORONTO	UMICH	UCSD
Classe di ele...		Corso	Corso	Corso	Corso	Corso	Corso	Corso	Corso	Corso	Corso
professore	Nome=profes...	Lecturer	Instructor								Fall2003
orario	Nome=orario,...	Time		Times							
nome_corso	Nome=nome...	CourseTitle	Title	TitleCredits	CourseName	Title	Titel	Title	title	name	Title
codice	Nome=codic...	Code	Code	Code	Code	Code					
testo_rif	Nome=testo_...								text		
crediti	Nome=crediti...	Units					Umfang				
descr	Nome=descr,...							Description		description	
prereq	Nome=prere...							MoreInfoURL		prerequisite	
prof2	Nome=prof2,...										Winter2004
prof3	Nome=prof3,...										Spring2004
giorno_lez	Nome=giorno...	Day									
spec	Nome=spec, ...		Description								
prova	Nome=prova...	Code	Code								

Figura 6. Interfaccia grafica "Designer Associazione Dati"

2.1.6 Servizi di accesso

Il componente dei servizi di accesso configura il *bean enterprise* dei servizi di accesso, distribuito nel server J2EE, gestendo le richieste dall'API e instradandole al componente appropriato. È possibile modificare le proprietà di questo componente dallo strumento di amministrazione, per personalizzare alcune funzionalità. In particolare, nell'implementazione del lavoro di tesi, è stata sfruttata la proprietà "Classe trasformatore query associata" che dà la possibilità di specificare una classe Java "transformer" query associata: cioè una classe che viene richiamata in automatico, ed in modo trasparente all'utente, durante l'esecuzione delle query federate e può compiere operazioni come la trasformazione di valori di attributi della query o la modifica dei risultati di query.

Questa classe "*transformer*" implementa una classe *interface* Java, `IFederatedQueryTransformer` (presente nelle SPI fornite dall'*IICE*), che permette di applicare varie trasformazioni alla query federata, quando questa viene eseguita sulle varie sorgenti. In particolare, `IFederatedQueryTransformer` mette a disposizione 4 metodi principali, che opportunamente programmati permettono di agire sulla query federata:

- `preProcessMultiQuery(MultiQuery original)`: prevede l'accesso alla Multiquery, prima che questa venga iterata sulle varie sorgenti dati. In questo metodo si permette la modifica della query federata, la creazione di una nuova o semplicemente non apporta nessuna modifica. Il parametro `original` è la query passata dall'applicazione client, nel caso specifico dall'applicazione `ExecMultiQuery`. Il metodo ritorna il parametro `original` stesso.
- `preProcessQuery(Query query)`: prevede l'accesso alla singola query prima che questa venga eseguita su una specifica sorgente, passata tramite il parametro `query`. Permette la modifica della query, qualora si voglia intervenire sulle proprietà di selezione o sui requisiti di ricerca. Il metodo ritorna il parametro `query` stesso.
- `postProcesResults(Query query,QueryResults queryResults)`: prevede l'accesso al `QueryResults`, ossia ai risultati che la query restituisce dopo essere eseguita su una particolare sorgente dati.

Offre la possibilità di intervenire sui risultati della query e di restituirli tramite il parametro `queryResults`.

- `postProcessResultSet(IServerResultSet serverResultSet)`: prevede l'accesso ai risultati restituiti da tutte le query eseguite sulle varie sorgenti dati chiamate in causa dalla query federata. Si possono eseguire modifiche o aggiungere anche delle tuple sui risultati finali.

Quindi è possibile creare una classe Java, che implementa l'SPI `IFederatedQueryTransformer` e settando il valore di configurazione, ossia il nome della classe transformer, nelle proprietà dei servizi di accesso dallo strumento di amministrazione, illustrata in, questa classe verrà eseguita in background dal processo di integrazione quando viene eseguita una query federata.

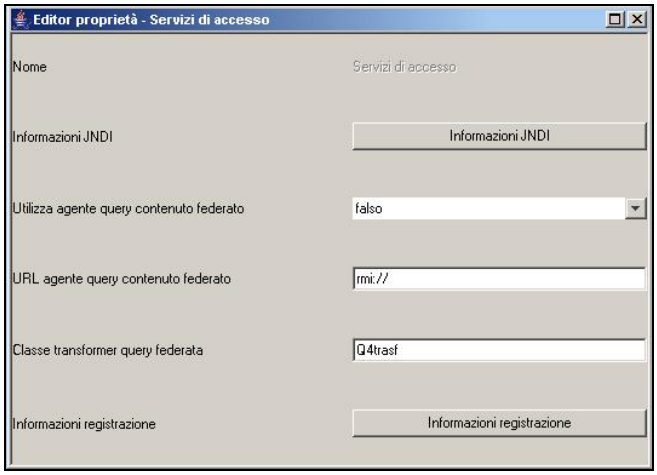


Figura 7. Interfaccia grafica per impostare il nome della classe transformer

2.1.7 Query federate

Una query federata è una semplice query specificata sugli attributi globali definiti nell'associazione dati, quindi l'interrogazione avrà come target tutte le sorgenti dati che hanno un attributo, di una loro struttura dati, mappato con l'attributo globale richiamato dalla querry.

La query federata viene lanciata tramite uno script "*`run_sample`*" dall'ambiente riga di comandi messo a disposizione dall'IICE; questo script non fa altro che richiamare un' API, `ExecMultiQuery`, che accetta diversi parametri:

```
run_sample commandline.ExecMultiQuery <listaSorgenti> <user>
<pwd> <associazione dati> <listaProrietà> [select=<criteri di ricerca>]
```

Ogni parametro ha il seguente significato:

- *lista sorgenti*, con questo parametro si inserisce l'elenco delle sorgenti che si vogliono interrogare;
- *user e pwd*, una coppia di username e password per ogni sorgente inserita in lista;
- *associazione dati*, in questo parametro si specifica un'istanza di un'associazione dati, contenente il mapping tra gli attributi delle varie sorgenti (si può inserire una sola istanza di associazione dati);
- *lista proprietà*, lista degli attributi di selezione della query;
- *criteri di ricerca*, elenco di attributi con relative espressioni di ricerca della query.

Una volta lanciata questa query, l'IICE sfrutta il server delle applicazioni Websphere Application Server per inviare la query federata, personalizzata per ogni sorgente, in parallelo. I risultati dell'interrogazione verranno visualizzati a video dallo script lanciato in precedenza per generare la query federata.

2.2 Configurazione e connessione alle sorgenti dati

Una volta installato l'IICE, tramite il pannello di amministrazione, si procede con la configurazione dei connettori. I database utilizzati come sorgenti dati sono memorizzati su un server diverso da quello in cui è installato l'IICE, e sono gestiti tramite il DBMS di Microsoft SQL Server 2000. Per questo motivo come tipologia di connettori si usa "Connettore RDBMS". Per ogni database al quale ci si deve collegare, si crea un nuovo connettore, e si configurano i parametri relativi all' URL JDBC e si specifica la classe del driver da utilizzare per la particolare tecnologia che ospita i database. In questo caso è necessario utilizzare i driver JDBC per MSQL Server; tramite lo script `rebuild_EAR.bat`, aggiungendo il codice mostrato in Figura 8, si imposta il percorso relativo alle classi Java del driver e si rigenera il file EAR VeniceBridge , incudendo le librerie Java per i driver JDBC di Microsoft. A questo punto si impostano i valori della classe dei driver da utilizzare, nelle opzioni di configurazione del connettore, e la stringa JDBC per la connessione al database:

- ✓ `com.microsoft.jdbc.sqlserver.SQLServerDriver`
- ✓ `jdbc:microsoft:sqlserver://apollo13.ing.unimo.it:4433; selectMethod=cursor;DatabaseName=cmu;User=XXX;Password =XXX`

Una volta configurato i connettori, e testato la connessione, per ognuno di questi, tramite il "Designer connettore RDBMS" si creano le classi di elementi in base alla configurazione della struttura dati alla quale è collegata il connettore, come descritto in precedenza.

```
java -classpath "%VBR_CLASSPATH%" ^
-Dvbr.home="%VBR_HOME%" ^
com.venetica.vbr.util.EarPackager ^
-libraryJars
./lib/vbr.jar,./lib/msbase.jar,./lib/mssqlserver.jar,./li
b/msutil.jar -appDirs ./ejb,./war -appName VeniceBridge
```

Figura 8. Codice dello script "rebuild_EAR.bat"

2.3 Implementazione delle query del benchmark THALIA in IICE

In questa sezione verrà illustrato il modo in cui è stato implementato il benchmark THALIA sul sistema di integrazione dati dell'IICE.

In primo luogo si procede col mapping degli attributi globali tramite l'interfaccia grafica "Designer Associazione dati", vedi Figura 6. Tramite questa è possibile creare delle associazioni di dati per definire i nomi di attributi globali che possono essere associati a uno o più elementi dati dei vari repository. Inoltre per eseguire la maggior parte delle query del benchmark THALIA si è dovuto implementare, per ognuna di queste, una particolare classe transformer che risolvesse i problemi di eterogeneità, sfruttando i metodi `preProcessQuery()` e `postProcesResults()`.

Di seguito verrà descritto il raffinamento compiuto per ogni singola query:

Query 1:

- ✓ Sorgenti dati: CMU,GATECH;
- ✓ Semplice mapping degli attributi: (instructor,lectur) con "professore", (code,code) con "codice", (CourseTitle,Title) con "nome_corso";
- ✓ No classe transformer;

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery CMU,GATECH XXX XXX Map_dati "codice,nome_corso,professore" select="professore like '*Clark*'"
```

Query 2:

- ✓ Sorgenti dati: CMU,UMB;
- ✓ Semplice mapping degli attributi: (CourseTitle,TitleCredits) con "nome_corso", (time,times) con "orario";
- ✓ Classe transformer: nel `preProcessQuery()`, per la query sulla sorgente UMB, vengono apportate modifiche sulla formattazione dell'attributo orario nei criteri di ricerca, trasformandolo dal formato Am/Pm in quello 0-24. Nel `postProcesResults()` si è intervenuto sui risultati restituiti dalla query effettuata sulla sorgente UMB, trasformando i valori del campo orario dal formato 0-24 a quello Am/Pm, cioè nella formattazione specificata nella query originale.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery CMU,UMB XXX XXX Map_dati "nome_corso,orario" select="orario like '1:30*'"
```

Query 3:

- ✓ Sorgenti dati: UMD,BROWN;
- ✓ Semplice mapping degli attributi: (CourseName,Title) con nome_corso, (code,code) con "codice";
- ✓ No classe transformer;

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery BROWN,UMD XXX XXX Map_dati "codice,nome_corso" select=" nome_corso like '*Data Structures*'"
```

Query 4:

- ✓ Sorgenti dati: ETHZ,CMU;
- ✓ Semplice mapping degli attributi: (CourseTitle,Titel) con "nome_corso", (Units,Umfang) con "crediti", mappato come tipo dati `int`, a differenza degli attributi "Units" e "Umfang" (di tipo `String`), per consentire l'inserimento degli operatori di confronto '<' '>' '=' nella formulazione della query;
- ✓ Classe transformer: nel `preProcessQuery()`, per la query sulla sorgente CMU, vengono apportate modifiche sui criteri di ricerca ed in particolare sul campo "crediti", facendo diventare il criterio di ricerca "crediti IsNotNull". E' stata necessaria questa modifica perché, se la query viene inoltrata sulla sorgente CMU nella sua forma originale viene restituito un errore di incompatibilità di tipo dati, in quanto l'operatore '>', nella query presa in considerazione, va ad agire sul tipo di dati "Units" (nella sorgente CMU) che è di tipo `String`. Quindi per aggirare il problema si modifica il criterio di ricerca, come illustrato in precedenza. Questo comporterà che, l'effettivo filtro (> 5) sul campo crediti verrà implementato nel metodo `postProcesResults()`. Inoltre, questo procedimento risulta poco efficiente, perché nel result set della query verranno restituite tuple che effettivamente non fanno parte dei risultati finali, ossia verranno restituiti dalla sorgente tutti i corsi con nome 'Database' che hanno il numero di crediti non maggiori di 5 (questo inconveniente verrà poi mascherato nel metodo `postProcesResults()`); per di più, se l'utente modifica il criterio di ricerca sul campo crediti nella query originale, bisogna implementare una classe transformer ad hoc che applichi il filtro di ricerca sui risultati finali. Sempre nel metodo `preProcessQuery()`, per la sorgente ETHZ, verranno riportate, sulla query originale, le stesse modifiche per la sorgente CMU, con l'aggiunta della modifica nei criteri di ricerca sul campo "nome corso", trasformandolo in: `nome_corso like '*Datei*' OR nome_corso like '*Datenbasis*' OR nome_corso like '*Datenbank*'`, per risolvere il problema della conversione in lingua tedesca dei criteri di

ricerca sul campo “nome_corso”. Nel metodo `postProcesResults()`, per i risultati restituiti dalla sorgente CMU viene implementato il controllo sul campo “Units”, parsando il suo valore come intero (essendo di tipo `String` alla sorgente); mentre per la sorgente ETHZ viene estratta l’informazione sui crediti parsando il valore del campo “Umfang” (che nella sua forma originale è: #u+#v) e calcolando i crediti tramite la formula #u+#v+1, aggiornando poi i valori in uscita del result set. Ricordando il fatto che sia per la sorgente ETHZ, sia per CMU nel metodo `postProcesResults()` vengono “oscurati” (annullando i campi “crediti” e “nome_corso” nelle righe del result set finale) tutti i risultati, che comunque sono restituiti in uscita dalle query effettuate su entrambe le sorgenti, ma che non rispettano la condizione sul campo “crediti”.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery CMU,ETHZ XXX XXX Map_dati "crediti,nome_corso" select="crediti > ‘5’ AND nome_corso like ‘*Database*’"
```

Query 5:

- ✓ Sorgenti dati: ETHZ,UMD;
- ✓ Semplice mapping degli attributi: (CourseName,Titel) con “nome_corso”;
- ✓ Classe transformer: nel metodo `preProcessQuery()`, per la sorgente ETHZ, si sono modificati i criteri di ricerca sul campo “nome corso”, trasformandolo in: `nome_corso like '*Datei*' OR nome_corso like '*Datenbasis*' OR nome_corso like '*Datenbank*’`, per risolvere il problema della conversione in lingua tedesca dei criteri di ricerca sul campo “nome_corso”.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery ETHZ,UMD XXX XXX Map_dati "nome_corso" select=" nome_corso like ‘*Database*’"
```

Query 6:

- ✓ Sorgenti dati: TORONTO,CMU;
- ✓ Semplice mapping degli attributi: (CourseTitle,title) con "nome_corso", (text) con "testo_rif". Si può notare che viene mappato solo l'attributo "text" della sorgente TORONTO, in quanto nella sorgente CMU non esiste un attributo che contenga questa informazione.
- ✓ Classe transformer: nel metodo `preProcessQuery()`, per la sorgente CMU, si è eliminato dalle proprietà di selezione il campo "testo_rif", non esistendo un mapping di quest'ultimo con un attributo della sorgente CMU. Nel metodo `postProcesResults()`, nei risultati della query eseguita sulla sorgente CMU, si è aggiunto un nuovo campo sul result set (con valore `Null` impostato per default) per far sì di rendere compatibili i risultati della query eseguita su questa sorgente con i risultati della query eseguita sulla sorgente TORONTO, nonché con quelli della query originale.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery TORONTO,CMU XXX XXX Map_dati "nome_corso,testo_rif" select=" nome_corso like '*Verification*'"
```

Query 7:

- ✓ Sorgenti dati: ASU,UMICH;
- ✓ Semplice mapping degli attributi: (name,title) con "nome_corso", (Description,description) con "descr", (MoreInfoUrl,prerequisite) con prereq.
- ✓ Classe transformer: per quanto riguarda la sorgente ASU, se non è presente la parola 'Prerequisite' nel campo "Description" significa che per quel particolare corso non esistono prerequisiti. Di conseguenza nel `preProcessQuery()`, per la sorgente ASU, si modificano i criteri di ricerca della query trasformandoli nel seguente modo:
  `nome_corso IsNotNull AND NOT descr like '*Prerequisite*'`. Mentre, sempre per la sorgente ASU, nel metodo `postProcesResults()`, considerato il fatto che la sorgente ASU non ha un campo esplicito dove viene memorizzata l'informazione sui prerequisiti di un corso, forzo la scrittura del valore 'none' nel campo "MoreInfoUrl" (mappato con l'attributo globale "prereq"), per aggiornare il

result set finale ed indicare che per quel particolare corso non vi sono prerequisiti.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery ASU,UMICH XXX XXX Map_dati "nome_corso,descr,prereq" select=" prereq like '*none*'"
```

Query 8:

- ✓ Sorgenti dati: ETHZ,GATECH;
- ✓ Semplice mapping degli attributi: (Title,Titel) con "nome_corso", (Description) con "spec".
- ✓ Classe transformer: nella sorgente ETHZ non è presente un attributo che memorizzi l'informazione relativa alla requisiti di accesso ai corsi di informatica (nella fattispecie 'Junior' o 'Senior'), quindi nel `preProcessQuery()`, per la sorgente ETHZ, si modificano sia le proprietà di selezione che i criteri di ricerca della query trasformandoli nel seguente modo:
  `nome_corso like '*inform*' OR nome_corso like '*computer*'`. Successivamente nel metodo `postProcesResults()`, considerato il fatto che la sorgente ETHZ non ha un campo esplicito dove viene memorizzata l'informazione sui requisiti di accesso, forzo la scrittura del valore 'none' nell'attributo globale "spec" (presente nel result set finale della query federata.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery ETHZ,GATECH XXX XXX Map_dati "nome_corso,spec" select=" spec like '*JR*'"
```

Query 9 e query 10:

Non è stato possibile implementare queste query, perché l'IICE non consente operazioni di Join quando si effettuano query federate. Infatti per risolvere queste query è necessario effettuare l'operazione di Join tra due diverse tabelle sulla stessa sorgente dati. Questa risulta essere una forte limitazione.

Query 11:

- ✓ Sorgenti dati: CMU,UCSD;
- ✓ Semplice mapping degli attributi: (CourseTitle,Title) con "nome_corso", (Instructor,Fall 2003) con "professore", (Winter 2004) con "prof2", (Spring 2004) con "prof3". In questo caso la sorgente UCSD, al contrario di quella CMU, possiede tre campi diversi (Fall 2003, Winter 2004, Spring 2004) che memorizzano l'informazione relativa ai professori che tengono un tipo di corso; per questo motivo, non potendo mappare tre attributi di una sorgente sullo stesso attributo globale (l'IICE consente di mappare un solo attributo per sorgente su un attributo globale), l'alternativa è stata quella di creare altri due attributi globali ("prof2", "prof3") per mappare i rimanenti due campi di UCSD relativi alle informazioni sui professori.
- ✓ Classe transformer: proprio sul problema del mapping degli attributi citato prima, nel `preProcessQuery()`, per la sorgente UCSD, si modificano sia le proprietà di selezione che i criteri di ricerca della query federata originale trasformandoli nel seguente modo: `nome_corso,professore,prof2,prof3` (propr. di selezione), `nome_corso like '*Database*'` (criteri di ricerca). Successivamente nel metodo `postProcesResults()` è necessario modificare il result set restituito dalla query su UCSD per renderlo conforme al result set della query federata; è stato quindi necessario unificare il valori contenuti nei tre attributi relativi alle informazioni dei professori, inserendoli nel campo "professore" del result set finale della query federata.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery CMU,UCSD XXX XXX Map_dati "nome_corso,professore" select="nome_corso like '*Database*'"
```

Query 12:

- ✓ Sorgenti dati: CMU,BROWN;
- ✓ Semplice mapping degli attributi: (CourseTitle,Title) con "nome_corso", (Time) con "orario", (Day) con "giorno_lez". In questo caso la sorgente BROWN, al contrario di quella CMU, memorizza l'informazione relativa al giorno e l'ora di lezione di un dato corso nell'attributo "Title".

- ✓ Classe transformer: proprio per il problema del mapping degli attributi citato prima, nel `preProcessQuery()`, per la sorgente BROWN, si modificano le proprietà di selezione della query federata originale escludendo dalle proprietà di selezione "orario" e "giorno_lez". Successivamente nel metodo `postProcesResults()` sempre per la sorgente BROWN è necessario estrarre i valori relativi al giorno e l'ora del corso contenuti nell'attributo "Title" e modificare il result set della query, aggiungendo i due campi relativi a "orario" e "giorno_lez" immettendo i valori estratti prima, per renderlo conforme al result set della query federata di partenza.

Comando per eseguire la query federata:

```
run_sample commandline.ExecMultiQuery CMU,BROWN XXX XXX
Map_dati "nome_corso,giorno_lez,orario" select="nome_corso
like '*Computer*Networks*'"
```

2.4 Architettura Oracle Data Integrator (ODI)

Oracle Data Integrator [ODI][15] è il sistema di integrazione sviluppato da Oracle che permette si sviluppare soluzioni di data warehouse evolute con un approccio innovativo E-LT, eseguendo le trasformazioni direttamente sul database "target".

Oracle Data Integrator ha un'architettura modulare organizzata su repository (vedi Figura 9), i quali sono accessibili in modalità client-server dai componenti (moduli grafici e agenti) scritti interamente in Java, sia a runtime sia in fase di progettazione.

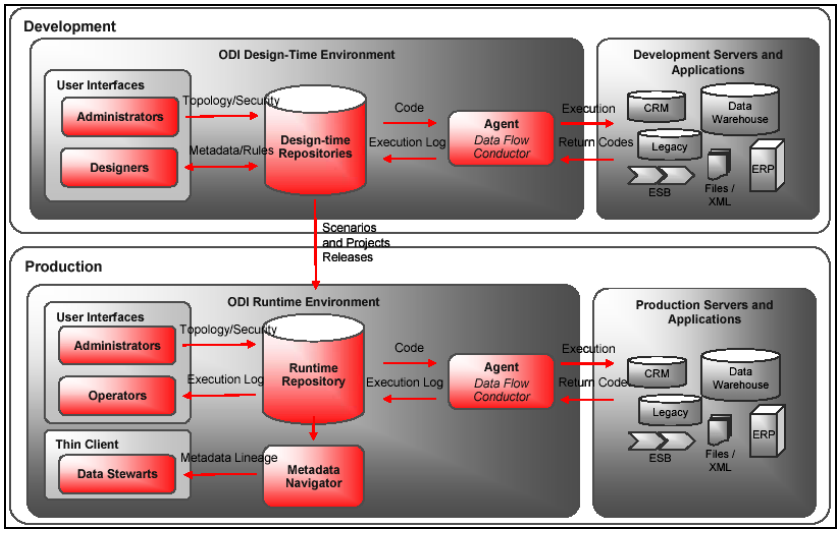


Figura 9. Architettura ODI

I quattro moduli grafici, che compongono il sistema di integrazione sono: Designer, Operator, Topology Manager e Security Manager, che possono essere installati su qualsiasi piattaforma che supporti la Java Virtual Machine 1.5 (J2SE), come Windows, Linux, HP-UX, Solaris, AIX, e Mac OS.

Il Designer, modulo "portante" del sistema di integrazione, definisce regole dichiarative per la trasformazione e l'integrità dei dati, è in questo modulo che vengono importati i metadati dei database e delle applicazioni. L'Operator è il modulo progettato per gestire e monitorare il flusso di integrazione, segnala tramite file di log gli eventuali errori nel processo ed i dati interessati, molto utile in fase di debug. Il Topology Manager definisce l'architettura fisica e logica di tutta l'infrastruttura di integrazione. I server, gli schemi e gli agenti sono registrati nel *master repository* attraverso questo modulo, ed amministrati tramite esso. Infine vi è il Security Manager per la gestione dei profili

utenti e dei privilegi di accesso, in pratica è il modulo per la gestione della sicurezza. Tutti questi quattro moduli memorizzano le loro informazioni nei repository.

Vi è un altro componente fondamentale nell'architettura dell'ODI, che agisce a tempo di esecuzione: lo Scheduler Agent, che coordina l'esecuzione di tutti gli scenari sviluppati in fase di integrazione. L'esecuzione può essere lanciata da uno dei quattro moduli descritti in precedenza o avviato direttamente dallo scheduler interno o di terze parti. Nel processo (E-LT) lo Scheduler Agent non esegue trasformazioni, ma semplicemente recupera il codice dai repository di esecuzione ed invia le richieste ai server dei database, al sistema operativo, o al motore di integrazione per far eseguire il codice recuperato. Quando l'esecuzione è completata, lo Scheduler Agent aggiorna i log di esecuzione nei repository e fa i report dei messaggi d'errore e le statistiche di esecuzione che possono essere controllati dall'utente tramite il modulo Operator. Di fatto lo Scheduler Agent coordina il processo di integrazione.

I repository consistono di un *master repository* and molti *work repositories*: questi sono dei Database memorizzati in RDBMS e contengono tutti gli oggetti configurati e sviluppati dai moduli, accessibili in modalità client-server dai vari componenti dell'architettura. Di solito si crea un solo master repository, che contiene le informazioni sulla sicurezza (profile utenti e privilegi) e sulle definizione delle tecnologie e dei server utilizzati. Le informazioni mantenute nel master repository sono gestate dal Topology Manager e dal Security Manager.

Invece gli oggetti create in fase di progettazione sono memorizzati nei work repositories: archivi dati, colonne, vincoli di integrità, regole dichiarative, procedure, cartelle e variabili, informazioni di scheduling e log. L'utente può gestire il contenuto dei work repositories tramite il Designer and l'Operator. Inoltre tutti i work repositories sono collegati ad un solo master repositories.

2.4.1 L'approccio E-LT

Il tradizionale approccio ETL viene eseguito estraendo prima i dati dalle varie sorgenti, portandoli nel "motore" ETL proprietario e caricando i dati trasformati nei data warehouse o server di integrazione. Lo step della trasformazione dei dati è un punto cruciale del processo, in quanto è molto oneroso a livelllo computazionale, ed è svolto interamente dal motore ETL sul server proprietario di integrazione. Questo processo di trasformazione viene svolto riga per riga e spesso risulta essere il collo di bottiglia nel processo generale. In più, i dati creano un sovraccarico del traffico di rete in quanto prima vengono passati dalle sorgenti al server di integrazione, poi dal server ETL alle destinazioni.

Invece con un approccio E-LT [16] (vedi Figura 10) la trasformazione dei dati avviene direttamente sull'RDBMS target, cambiando l'ordine delle operazioni: prima avviene l'estrazione dei dati dalle sorgenti, poi vengono caricati sui database target, ed è qui che vengono trasformati utilizzando le operazioni SQL native.

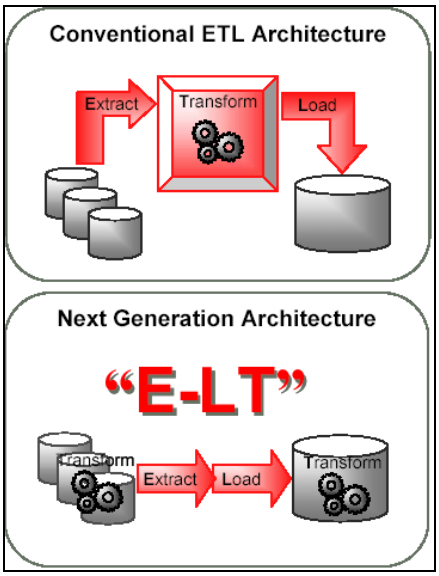


Figura 10. Funzionamento processo "E-LT"

L'approccio E-LT sfrutta la Potenza del motore RDBMS ed elimina la presenza di un server proprietario di integrazione, riducendo così il traffico di rete ed aumentando le performance e la scalabilità di tutta l'infrastrutttura di integrazione. Il processo di trasformazione non viene svolto riga per riga, ma con una singola query sfruttando la potenzialità del join sul server target e aumentando le prestazioni.

Inoltre, per progettare un processo si integrazione ETL tradizionale, lo sviluppatore deve implementare ogni step del processo, richiedendo particolari requisiti per i singoli step del processo; vi è un maggiore dispendio in fase di progettazione, perché le sequenze di processi ripetitivi, quali la gestione degli inserimenti e degli aggiornamenti, nei server target ne richiedono l'implementazione in ogni singolo processo. In pratica, con l'approccio ETL gli aspetti logici e tecnici dell'integrazione non sono separati. Mentre con il metodo di progettazione "dichiarativa" (vedi Figura 11), messo in pratica con l'approccio E-TL, bisogna solo progettare colo "cosa" deve fare il processo di integrazione e non "come" deve essere fatto.

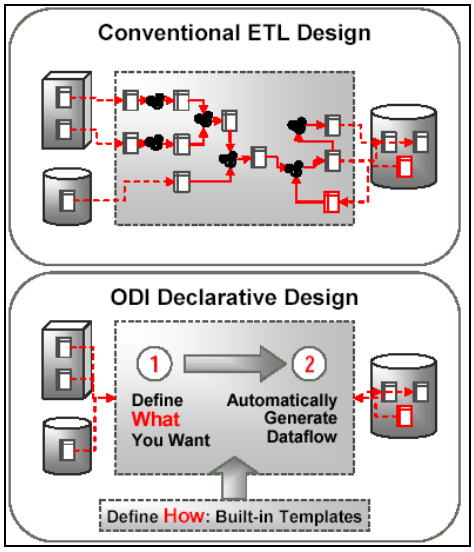


Figura 11. Progettazione dichiarativa ODI

2.4.2 Interfacce

La progettazione dichiarativa sfrutta il paradigma relazionale per definire, tramite delle "Interfacce" (vedi Figura 12), le regole dichiarative per un processo di integrazione, come la definizione delle sorgenti dati, del target e delle espressioni di trasformazione.

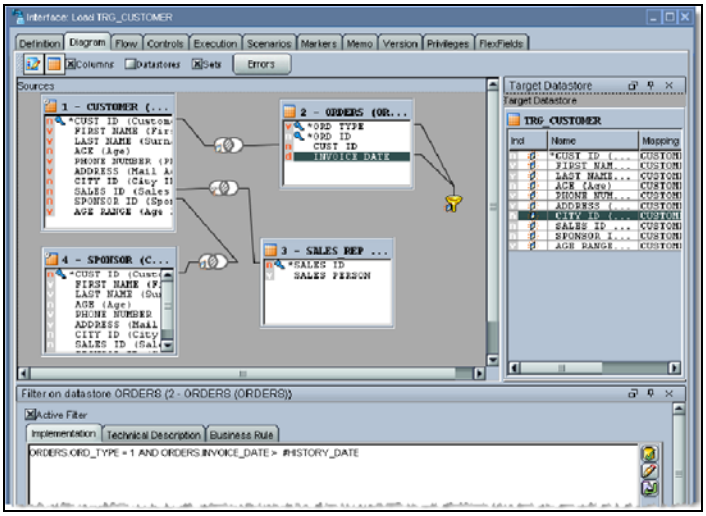


Figura 12. Modulo di progettazione "Interfaccia"

Un "Interfaccia" consiste di un insieme di regole che definiscono il processo di caricamento dei dati da una particolare sorgente ad una destinazione; nel caso in esame le interfacce sono state utilizzate per caricare i dati dalle tabelle sorgenti (definite nei modelli) alla tabella "target" (creata in precedenza, con gli opportuni campi necessari alla logica di integrazione) nella quale ci saranno effettivamente solo i dati che rispondono alle regole di integrazione. La tabella target può essere creata in automatico dal processo di caricamento dati in un'area di memoria temporanea dell'ODI, chiamata "staging area", oppure creata manualmente in un database e caricata in un modello tramite il *reverse_engineering*, prima del processo ETL. Una volta specificate le tabelle sorgenti e la tabella destinazione, all'interno dell'Interfaccia, è possibile specificare le condizioni di join tra le tabelle sorgenti (se più di una). Un passaggio fondamentale, nella programmazione dell'Interfaccia, è la creazione dei mapping tra gli attributi della tabella, o tabelle, sorgente e la tabella target, con la possibilità di definire per ogni associazione particolari funzioni di trasformazione, utili per implementare la strategia di estrazione dei dati. In particolare, l'ODI mette a disposizione un set di funzioni per ogni tecnologia supportata ospitante le tabelle sorgenti. Nel caso in esame sono state

utilizzate funzioni *like SQL92*, considerando il fatto che si è utilizzato MSQL Server 2000. Tramite queste è stato possibile definire delle "funzioni filtro", per definire il flusso dati durante il caricamento dalla tabella sorgente al quella target.

2.4.3 Knowledge Modules

Gli aspetti tecnici del processo di integrazione sono implementati tramite i cosidetti Knowledge Modules, un codice indipendente dall'interfaccia di progettazione, il quale gestisce le sorgenti, le destinazioni dati e le trasformazioni processate. In fase di progettazione vengono creati metadati, che descrivono il processo di integrazione; questi vengono elaborati dal Knowledge Module, che a sua volta genera del codice pronto per essere eseguito a runtime sulle sorgenti e destinazioni del processo di integrazione. Ognuno di questi esegue uno specifico compito di integrazione: il reverse-engineering dei metadati di sorgenti eterogenee; gestisce il Changed Data Capture (CDC) su un sistema; carica i dati da un sistema all'altro utilizzando ottimizzazioni; integra i dati in un sistema target, utilizzando specifiche strategie; controlla l'integrità del flusso dei dati.

L'ODI mette a disposizione, per le varie tecnologie supportate, varie tipologie di Knowledge Modules che effettuano letture da files sorgenti:

- ✓ LKM File to SQL;
- ✓ LKM File to DB2 UDB;
- ✓ LKM File to MSSQL;
- ✓ LKM File to Netezza;
- ✓ LKM File to Oracle (SQLLDR);
- ✓ LKM File to SalesForce;
- ✓ LKM File to SAS;
- ✓ LKM File to Sybase IQ;
- ✓ LKM File to Teradata;

ed esportazione di dati su files target:

- ✓ IKM SQL to File Append;
- ✓ IKM Netezza To File (EXTERNAL TABLE);
- ✓ IKM SalesForce to File;
- ✓ IKM Teradata to File (FASTEXPORT).

Il codice di questi moduli è “open” e può essere personalizzato per implementare nuovi metodi di integrazione. Oracle Data Integrator possiede più di 100 Knowledge Modules (vedi Figura 13) per i maggiori DBMS commerciali.

Il Changed Data Capture è il modulo che si occupa dell’identificazione e cattura degli eventi di inserimento, modifica cancellazione nelle sorgenti dati interessate in un progetto di integrazione. Oracle Data Integrator può utilizzare due metodi diversi per tracciare i cambiamenti nelle sorgenti dati: i trigger e i log RDBMS.

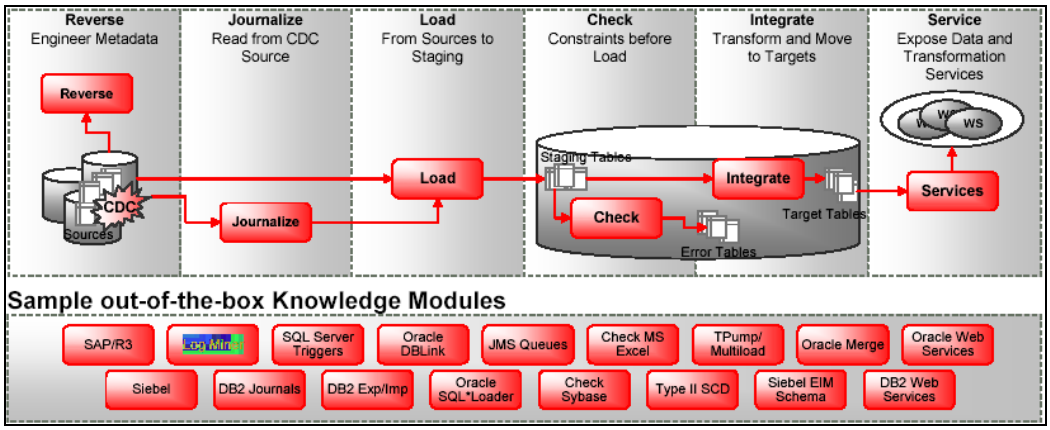


Figura 13. Knowledge Modules

Il primo metodo può essere utilizzato nella maggior parte degli RDBMS che supportano I trigger. Questo metodo è ottimizzato per diminuire il sovraccarico computazionale sul sistema ove risiedono le sorgenti. Per esempio, i cambiamenti catturati dal trigger non sono duplicati, minimizzando così il numero delle operazioni di input/output. Il second metodo invece si base sull’analisi dei file di log degli RDBMS, dove vengono memorizzate tutte le operazioni svolte all’interno dei database. Questo metodo provoca una leggera diminuzione delle prestazioni del sistema transazionale ed è supportato solo per Oracle e IBM DB2/400.

2.5 Configurazione e connessione alle sorgenti dati

Prima di utilizzare il moduli Topology Manager ed il Designer è necessario creare due database nel DBMS utilizzato come contenitore dei progetti di integrazione dati sviluppati attraverso l'ODI (nel caso preso in esame è stato utilizzato Microsoft SQL Server 2000), denominati *Master repository* e *Work repository* [17], e configurarne i parametri di accesso tramite l'interfaccia grafica messa a disposizione dal Topology Manager e dal Designer. In pratica questi database sono utilizzati dall'ODI per mantenere le informazioni di configurazioni dei diversi progetti: in particolare il *Master repository* memorizza le informazioni relative alla particolare tecnologia utilizzata per contenere i dati che serviranno poi per l'integrazione, informazioni sulla sicurezza e sulle versioni dei vari progetti e modelli sviluppati in fase di sviluppo del progetto E-LT; mentre il *Work repository* contiene informazioni di configurazione dei modelli dei dati (creati durante la fase di integrazione), dei progetti e del loro uso (come la pianificazione dei processi, lo scheduling ed i report di esecuzione). Attraverso l'interfaccia grafica mostrata in Figura 14 è possibile specificare, tra le varie proprietà, i driver per la particolare tecnologia RDBMS utilizzata per la memorizzazione del *Master repository* e i parametri della connessione JDBC.

Successivamente, tramite una simile interfaccia grafica, messa a disposizione dal Designer, va effettuata la stessa configurazione per il *Work repository*. A questo punto il Topology Manager ed il Designer sono correttamente configurati.

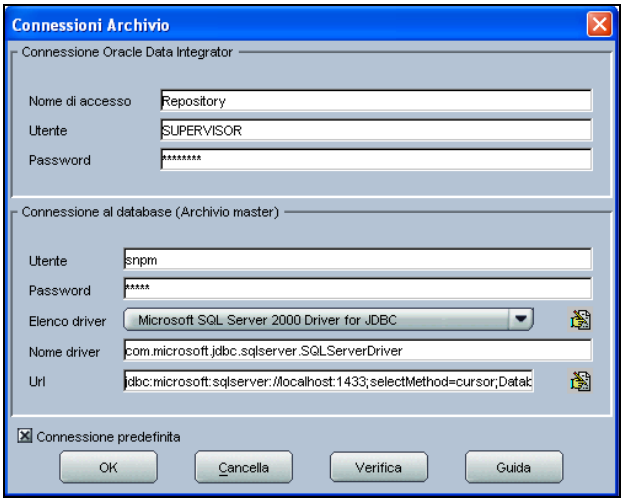


Figura 14. Interfaccia grafica per la connessione all'archivio Master

Usando il modulo Topology Manager è possibile configurare le connessioni alle sorgenti dati e gestire le relative informazioni. L' ODI supporta molte tecnologie che

possono ospitare le sorgenti dati, ed inoltre il repository dei metadati (schema globale/integrato/target) può essere installato su diversi RDBMS (vedi Figura 15).

Figura 15. Tecnologie supportate

Attraverso l'interfaccia del "Topology Manager" (vedi Figura 16), una volta selezionato la particolare tecnologia, è possibile definire un *server di dati*: tramite un'interfaccia vengono impostati i driver che verranno utilizzati per la particolare tecnologia alla quale ci si dovrà connettere (nel caso in esame sono stati impostati i driver JDBC per MSQL Server) e si definisce la stringa di connessione JDBC relativa al particolare host che ospita la tecnologia contenente i database nei quali sono memorizzati i dati che serviranno poi nella fase di integrazione. Una volta definito il server dei dati, ad esso verrà associata un'istanza, denominata "schema fisico", per ogni database contenente gli schemi dei dati da integrare; in particolare, per ogni istanza, si specifica il nome del database ed il nome dello schema logico (univoco) da associare alle strutture dati contenute nel database. In questo modo si crea uno schema fisico per ogni database sorgente.

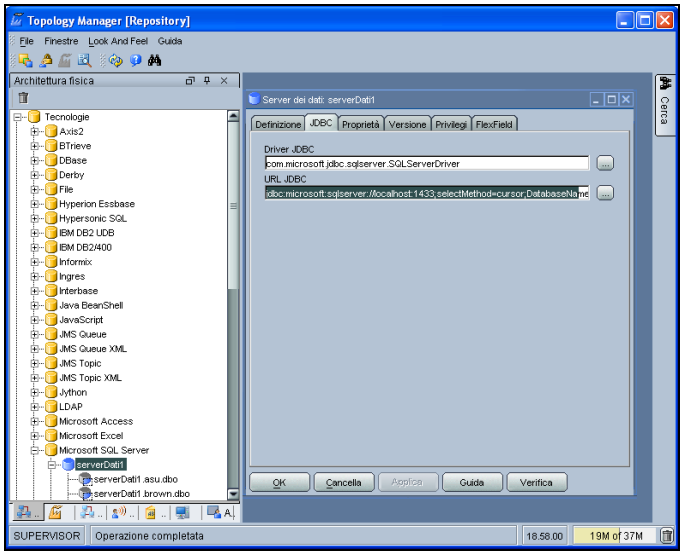


Figura 16. Interfaccia del Topology manager

Una volta terminata la configurazione delle connessioni alle varie sorgenti, è possibile passare alla fase di integrazione vera e propria utilizzando il modulo "Designer". Questo permette innanzitutto di definire dei *modelli*, che contengono l'insieme delle informazioni relative alle strutture dati contenute negli schemi fisici precedentemente configurati. Di per sé un modello, non contiene i dati veri e propri da integrare, in pratica è una diretta associazione allo schema logico, definito per un particolare schema fisico nel Topology Manager. Successivamente, con l'opzione *reverse_engineering* vengono caricati, effettivamente, nei modelli le tabelle o i file contenuti nelle strutture dati associate ad un particolare schema fisico. In questo modo, all'interno del particolare modello, si avrà una visione completa delle strutture dati. Successivamente alla creazione dei modelli, è possibile dar vita al progetto E-LT, programmando le "Interfacce". Come descritto nel paragrafo 2.4.2 tramite questi moduli è possibile definire il processo E-LT, definendo la tabelle sorgenti da integrare nel processo e la tabella destinazione sulla quale effettuare l'integrazione dei dati. Si definiscono inoltre altre opzioni relative al controllo e gestione del processo.

In Figura 17 si può notare il mapping definito tra l'attributo "Time" della sorgente "Section" e l'attributo "aula" della tabella target "Table09", ed inoltre su quest'associazione è definita anche una funzione di trasformazione.

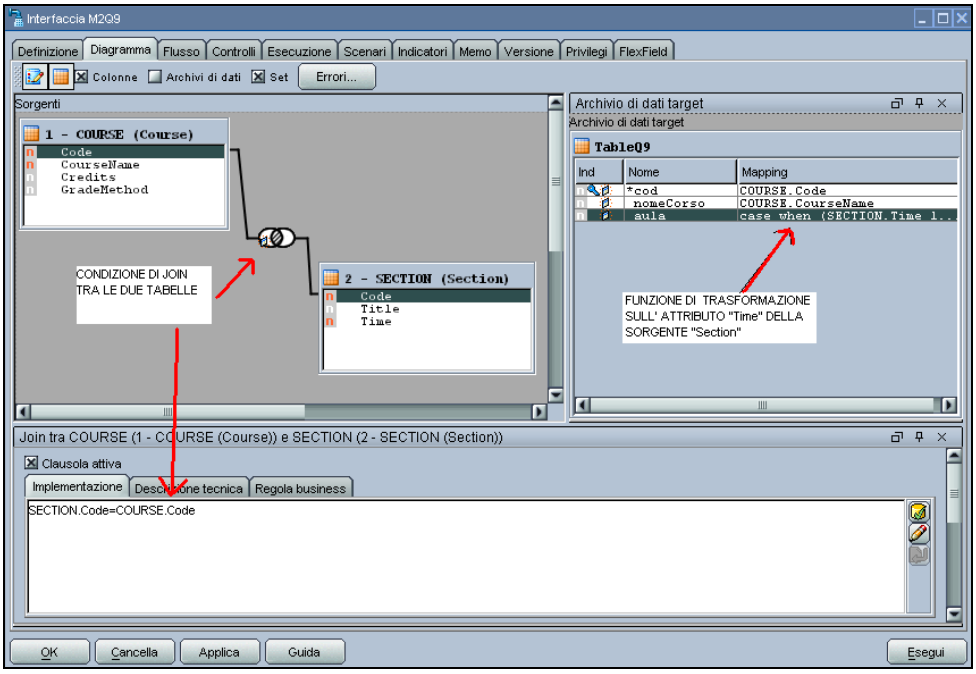


Figura 17. Snapshot dell'Interfaccia della query 9

Una volta definito il mapping tra gli attributi, nella sezione *Flusso* dell'Interfaccia, è possibile definire, nella strategia di integrazione, la gestione delle eventuali modifiche nelle sorgenti per tenere aggiornato il database integrato senza bisogno di tornare ad eseguire il processo di integrazione. Questo è reso possibile da moduli sviluppati ad-hoc, chiamati Integration Knowledge Module (IKM) [18], che seguono gli eventi di modifica delle sorgenti e li processano regolarmente in batch (pull mode) o in tempo reale (event-driven, push mode). In particolare, per la tecnologia MSQL Server, vi sono due tipi di IKM che permettono, rispettivamente, di applicare due strategie differenti per il controllo del processo di integrazione: IKM SQL Incremental Update, crea una tabella nella *staging area* contenente il flusso dei dati della fase di caricamento e confronta il contenuto di questa tabella con la tabella "Target" per capire quali record aggiornare e quali inserire come nuovi; mentre IKM Slowly Changing Dimension controlla, durante la fase di caricamento, i record da inserire o da aggiornare direttamente sulla tabelle target. Inoltre, il controllo sui dati estratti dei vincoli e delle regole definite nell'Interfaccia vengono eseguiti da un'altra tipologia di moduli, Check Knowledge Module (CKM): tramite questi moduli è possibile recuperare, in apposite tabelle accessibili sempre dal Designer, i record che non soddisfano i vincoli o le trasformazioni definite prima della fase di caricamento; ciò aumenta la facilità di controllo dell'integrazione, e permette di individuare possibili defaiance dovute alla

logica implementata dalla funzioni di trasformazione o derivanti dall'inadeguatezza di alcuni record ai vincoli imposti in fase di estrazione.

2.6 Implementazione delle query del benchmark THALIA in ODI

In questo paragrafo verrà descritto il procedimento di integrazione eseguito con l'ODI per implementare le query del benchmark THALIA. Prima di tutto è stato necessario definire,nel Topology Manager, uno schema fisico per ogni database relativo ai dati di ogni università, definendo i parametri di collegamento ai vari database. Successivamente nel Designer si è creato un modello per ogni sorgente (o schema fisico) e si è proceduto con il *reverse engineering* per l'importazione delle strutture dati relative alle diverse università. Una volta terminata questa prima fase, per ogni query è stato necessario creare dapprima la tabella target nel quale inserire i dati sui quali effettuare l'interrogazione; ciò ha determinato la conoscenza, a priori, della struttura che dovrà assumere la tabella necessaria per l'integrazione dei dati. Inoltre, ogni query, coinvolge due sorgenti dati distinte, quindi per implementare ognuna di queste, sono state programmate due Interfacce per l'estrazione, trasformazione e caricamento dei dati delle rispettive sorgenti nella tabella target di integrazione. Una volta creato ed eseguito le due Interfacce ed aver popolato la tabella target con i dati necessari per soddisfare la particolare query, si è passati alla fase finale, interrogando la tabella target con la query stessa. Di seguito verrà illustrato, per ognuna delle dodici query del benchmark, le sorgenti chiamate in causa, la definizione delle Interfacce relative alle singole query (mapping attributi e funzioni di trasformazione applicate), la struttura della tabella target.

Query_1:

Sorgenti: CMU, GATECH

Tabella dei mapping:

ATTRIBUTI DELLE SORGENTI	ATTRIBUTI TABELLA TARGET (TableQ1)
CRN (gatech) Code (cmu)	codice [tipo dati: `string`]
Title (gatech) CourseTitle (cmu)	nomeCorso [tipo dati: `string`]
Instructor (gatech) Lecturer (cmu)	prof [tipo dati: `string`]

Nessuna condizione di Join, solo una tabella per ogni sorgente.

Nessuna funzione di trasformazione necessaria, semplice mapping tra attributi.

Query eseguita:

```
select codice, nomeCorso, prof
from dbIntegrazione.dbo.TableQ1 where prof like '%Clark%'
```

Query_2:

Sorgenti: CMU, UMB

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
Code (cmu) Code (umb)	cod [tipo dati: `string`]
CourseTitle (cmu) TitleCredits (umb)	nomeCorso [tipo dati: `string`]
Time (cmu) **F1** (umb)	orario [tipo dati: `string`]

Nessuna condizione di Join, solo una tabella per ogni sorgente.

Funzione di trasformazione definita sul campo “Times” di UMB (**F1**):

```
CASE WHEN ISNUMERIC(SUBSTRING(COURSE.Times, 1, 2)) = 1 THEN
   CASE WHEN CAST(SUBSTRING(COURSE.Times, 1, 2) AS int) > 12
        THEN CAST(CAST(SUBSTRING(COURSE.Times, 1, 2) AS
integer)- 12 AS nvarchar(2))
        ELSE SUBSTRING(COURSE.Times, 1, 2)
   END
  +  SUBSTRING(COURSE.Times, 3, 4)  +
   CASE WHEN CAST(SUBSTRING(COURSE.Times, 7, 2) AS int) > 12
        THEN CAST(CAST(SUBSTRING(COURSE.Times, 7, 2) AS
integer)- 12 AS nvarchar(3))
        ELSE  SUBSTRING(COURSE.Times, 7, 2)
   END
+ SUBSTRING(COURSE.Times, 9, 3)
END
```

Query eseguita:

```
select cod,nomeCorso,orario from dbIntegrazione.dbo.TableQ2
where orario like '?1:30%'
```

Query_3:

Sorgenti: BROWN, UMD

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
Code (brown) Code (umd)	cod [tipo dati: `string`]
Title (brown) CourseName (umd)	nomeCorso [tipo dati: `string`]

Nessuna condizione di Join, solo una tabella per ogni sorgente.

Nessuna funzione di trasformazione necessaria, semplice mapping tra attributi.

Query eseguita:

```
select cod,nomeCorso from dbIntegrazione.dbo.TableQ3 where
nomeCorso like '%Data Structures%'
```

Query_4:

Sorgenti: ETHZ, CMU

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
CourseTitle (cmu) Titel (ethz)	nomeCorso [tipo dati: `string`]
Units (cmu) **F2** (ethz)	crediti [tipo dati: `int`]

Nessuna condizione di Join, solo una tabella per ogni sorgente.

Funzione di trasformazione definita sul campo "Umfang" di ETHZ (**F2**):

```
CAST (SUBSTRING (UNTERRICHT.Umfang, CHARINDEX('V',
UNTERRICHT.Umfang) - 1, 1) AS int)
+  CAST(SUBSTRING(UNTERRICHT.Umfang, CHARINDEX('U',
UNTERRICHT.Umfang) - 1, 1) AS int)
+1
```

Query eseguita:

```
select nomecorso,crediti from dbIntegrazione.dbo.TableQ4
where nomeCorso like '%Database%'AND crediti >= '5'
```

Query_5:

Sorgenti: ETHZ, UMD

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
CourseName (umd) **F3** (ethz)	nomeCorso [tipo dati: `string`]
Code (umd) Typ (ethz)	cod [tipo dati: `string`]

Nessuna condizione di Join, solo una tabella per ogni sorgente.

Funzione di trasformazione definita sul campo “Titel” di ETHZ (**F3**):

```
CASE WHEN ((UNTERRICHT.Titel like '%Datei%')
OR(UNTERRICHT.Titel like '%Datenbasis%')OR (UNTERRICHT.Titel
like '%Datenbank%'))
      THEN 'Database'
            ELSE UNTERRICHT.Titel
END
```

Query eseguita:

```
select cod,nomeCorso
from dbIntegrazione.dbo.TableQ5
where nomeCorso like '%Database%'
```

Query_6:

Sorgenti: CMU, TORONTO

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
title (toronto) CourseTitle (cmu)	nomeCorso [tipo dati: `string`]
text (toronto) “no map” (cmu)	testoRif [tipo dati: `string`]

Nessuna condizione di Join, solo una tabella per ogni sorgente. Dalla tabella riassuntiva si può notare che sull’attributo globale “testoRif” non è mappatto nessun attributo della sorgente CMU, in quanto non è presente il campo contenente l’informazione richiesta. Questo è un primo esempio di gestione dei valori `null`.

Nessuna funzione di trasformazione necessaria, semplice mapping tra attributi.

Query eseguita:

```
select nomeCorso,testoRif from dbIntegrazione.dbo.TableQ6
where nomeCorso like '%Verification%'
```

Query_7:

Sorgenti: ASU, UMICH

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
title (asu) name (umich)	nomeCorso [tipo dati: `string`]
Description (asu) description (umich)	descr [tipo dati: `string`]
F4 (asu) prerequisite (umich)	prereq [tipo dati: `string`]

Nessuna condizione di Join, solo una tabella per ogni sorgente. In questa query vi è un altro caso di gestione dei valori `null`: infatti non è presente un attributo contenente solo le informazioni sui prerequisiti di un corso nella sorgente ASU, ma questa informazione è stata ricavata, quando presente, dall'attributo "Description" tramite la funzione di trasformazione (**F4**); mentre quando non presente, la "**F4"** assegna il valore 'none' come valore per l'attributo globale "prereq".

Funzione di trasformazione definita sul campo "Description" di ASU (**F4**):

```
CASE WHEN (COURSE.Description LIKE '%Prerequisite%')
     THEN  SUBSTRING(COURSE.Description,
PATINDEX('%Prerequisite%',COURSE.Description ) + 1,
LEN(COURSE.Description) -
PATINDEX('%Prerequisite%',COURSE.Description ))
ELSE  'none'
END
```

Query eseguita:

```
select nomeCorso,descr,prereq from dbIntegrazione.dbo.TableQ7
where prereq like '%none%'
```

Query_8:

Sorgenti: ETHZ, GATECH

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
Titel (ethz) Title (gatech)	nomeCorso [tipo dati: `string`]

Typ (ethz) CRN (gatech)	cod [tipo dati: `string`]
'none' (ethz) Description (gatech)	spec [tipo dati: `string`]

Nessuna condizione di Join, solo una tabella per ogni sorgente. In questa query vi è un altro esempio di gestione dei valori `null`: infatti non è presente nessun attributo nella sorgente ETHZ contenente le informazioni sulla specializzazione di un corso. Nella fase di integrazione il problema è stato risolto assegnando per default il valore 'none' nel mapping con l'attributo globale "spec".

Nessuna funzione di trasformazione necessaria, semplice mapping tra attributi.

Query eseguita:

```
select cod,nomeCorso,spec from dbIntegrazione.dbo.TableQ8
where spec like '%JR%'
```

Query_9:

Sorgenti: UMD, BROWN

Per risolvere questa query è necessario definire nell'Interfaccia la condizione di `JOIN` tra due tabelle presenti nel database UMD (vedi Figura 18), in particolare l'informazione sul nome del corso è sulla tabella "Course" mentre l'informazione relativa all'aula è memorizzata nel campo "Time" del tabella "Section". La condizione di `JOIN` è: "`SECTION.Code = COURSE.Code`".

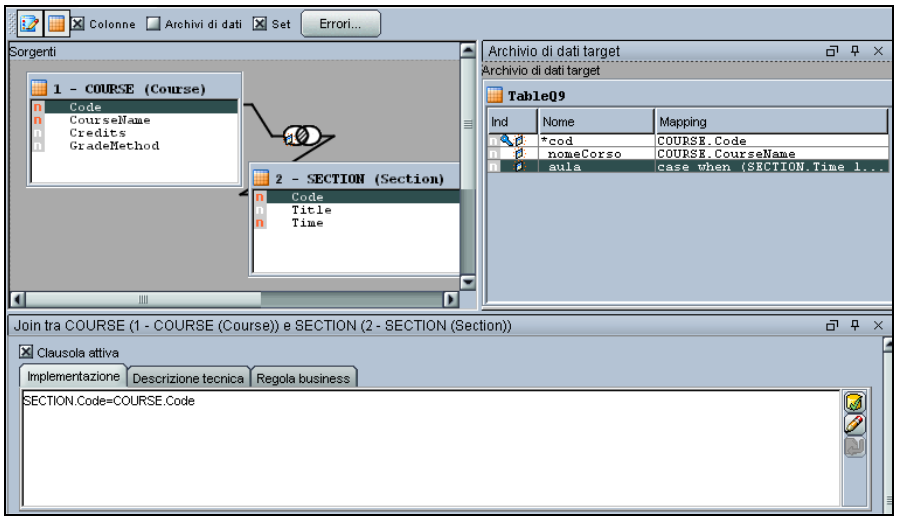


Figura 18. Definizione JOIN, sulla sorgente UMD, tra la tabella Section e Course

Tabella dei mapping:

ATTRIBUTI DELLE SORGENTI	ATTRIBUTI TABELLA TARGET (Table Q9)
Title (brown) CourseName (umd)	nomeCorso [tipo dati: `string`]
Code (brown) Code (umd)	cod [tipo dati: `string`]
Room (brown) **F5** (umd)	aula [tipo dati: `string`]

Inoltre è necessario estrarre l'informazione relativa all'aula tramite la funzione di trasformazione **F5** sull'attributo "Time", nella sorgente UMD:

```
CASE WHEN (SECTION.Time like '%(%')
     THEN SUBSTRING(SECTION.Time, PATINDEX('%(%',
SECTION.Time), 30)
          ELSE  'no info room'
END
```

Query eseguita:

```
select cod,nomeCorso,aula
from dbIntegrazione.dbo.TableQ9
where nomeCorso like '%Software Engineering%'
```

Query_10:

Sorgenti: UMD, CMU

Anche per questa query è stata definita, nell'Interfaccia, la condizione di `JOIN` tra le due tabelle presenti nel database UMD, in particolare l'informazione sul nome del corso è sulla tabella "Course" mentre l'informazione relativa ai professori è memorizzata nel campo "Title" del tabella "Section". La condizione di `JOIN` è sempre: "`SECTION.Code = COURSE.Code`".

Tabella dei mapping:

ATTRIBUTI DELLE SORGENTI	ATTRIBUTI TABELLA TARGET (Table Q10)
CourseTitle (cmu) CourseName (umd)	nomeCorso [tipo dati: `string`]
Code + Sec (cmu) Code (umd)	cod [tipo dati: `string`]

Lecturer (cmu) **F6** (umd)	professore [tipo dati: `string`]

È necessario estrarre l'informazione relativa ai professori tramite la funzione di trasformazione **F6** sull'attributo "Title", nella sorgente UMD:

```
SUBSTRING(SUBSTRING(SECTION.Title, 1,
PATINDEX('%.%',SECTION.Title)), PATINDEX('%)%',SECTION.Title)
+ 2, PATINDEX('%.%',SECTION.Title) + 1)
```

Query eseguita:

```
select cod, nomeCorso, professore from
dbIntegrazione.dbo.TableQ10
where nomeCorso like '%computer%'
```

Query_11:

Sorgenti: UCSD, CMU

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
CourseTitle (cmu) CourseName (ucsd)	nomeCorso [tipo dati: `string`]
Code + Sec (cmu) Code (ucsd)	cod [tipo dati: `string`]
Lecturer (cmu) **F7** (ucsd)	prof [tipo dati: `string`]

Nella sorgente UCSD le informazioni sui professori sono memorizzate in tre campi distinti (Winter2004, Fall2003, Spring2004) al contrario della sorgente CMU, dove l'informazione è contenuta solo nel campo "Lecturer". Con la trasformazione **F7** si concatenano le informazioni contenute nei tre campi mappandole nel solo attributo globale "prof":

```
COURSE.Fall2003 + ' and ' + COURSE.Winter2004 + ' and ' +
COURSE.Spring2004
```

Query eseguita:

```
select nomeCorso,prof from dbIntegrazione.dbo.TableQ11 where
nomeCorso like '%Database%'
```

Query_12:

Sorgenti: BROWN, CMU

Tabella dei mapping:

ATTRIBUTI DELLE	ATTRIBUTI TABELLA TARGET
CourseTitle (cmu) **F8** (brown)	nomeCorso [tipo dati: `string`]
Day (cmu) **F9** (brown)	giornoLez [tipo dati: `string`]
Time (cmu) **F10** (brown)	orario [tipo dati: `string`]

Nella sorgente BROWN le informazioni relative al nome del corso, giorno di lezione ed ora sono contenute nel solo campo "Title". Con le trasformazioni **F8**, **F9**, **F10** si estraggono le informazioni mappandole nei rispettivi attributi globali (nome Corso, giornoLez, orario) della tabella di integrazione:

F8:

```
SUBSTRING(COURSE.Title, CHARINDEX('/\"', COURSE.Title) + 3,
CHARINDEX('hr.', SUBSTRING(COURSE.Title, CHARINDEX('/\"',
COURSE.Title) + 3, 1000)) - 1)
```

F9:

```
SUBSTRING(COURSE.Title, CHARINDEX('hr.', COURSE.Title) + 4,
CHARINDEX(' ', SUBSTRING(COURSE.Title, CHARINDEX('hr.',
COURSE.Title) + 4, 10)))
```

F10:

```
SUBSTRING(COURSE.Title, CHARINDEX(' ', SUBSTRING(COURSE.Title,
CHARINDEX('hr.', COURSE.Title) + 4, 10)) + CHARINDEX('hr.',
COURSE.Title) + 4, 15)
```

Query eseguita:

```
select nomeCorso, giornoLez, orario
from dbIntegrazione.dbo.TableQ12
where nomeCorso like '%Computer%Network%'
```

2.7 Architettura Microsoft SQL Server 2005 Integration Services

Microsoft SQL Server 2005 Integration Services (SSIS) è una nuova piattaforma estremamente scalabile per la creazione di soluzioni di integrazione dati a prestazioni elevate, compresi pacchetti di estrazione, trasformazione e caricamento (ETL) per il data warehousing. Integration Services fa parte dell'ambiente di progettazione Business Intelligence Development Studio, che fornisce tutte le potenti funzionalità dell'ambiente di sviluppo Microsoft Visual Studio agli sviluppatori che utilizzano il pacchetto Integration Services;

Nell'SSIS possiamo distinguere quattro componenti principali [19] (vedi Figura 19):

- ✓ *il servizio Integration Services*, responsabile del monitoraggio dell'esecuzione dei pacchetti di Integration Services e gestisce l'archiviazione dei pacchetti;
- ✓ *il modello di oggetti di Integration Services*, che include API native e gestite per l'accesso agli strumenti, alle utilità della riga di comando e alle applicazioni personalizzate di Integration Services ;
- ✓ *il run-time*, che salva il layout dei pacchetti, esegue i pacchetti e offre il supporto per la registrazione, i punti di interruzione, la configurazione, le connessioni e le transazioni; invece gli eseguibili di run-time sono costituiti dai pacchetti, dai contenitori, dalle attività e dai gestori di eventi di Integration Services, nonché dalle attività personalizzate.;
- ✓ *l'attività Flusso di dati*, che incapsula il motore flusso di dati, che fornisce i buffer in memoria utilizzati per lo spostamento dei dati dalle origini alle destinazioni e chiama le origini che estraggono i dati dai file e dai database relazionali. Il motore flusso di dati gestisce inoltre le trasformazioni che modificano i dati e le destinazioni che li caricano o li rendono disponibili ad altri processi. I componenti dei flussi di dati sono costituiti dalle origini, dalle trasformazioni e dalle destinazioni. Il motore del flusso di dati di Integration Services supporta oltre 25 trasformazioni e oltre 10 origini e destinazioni utilizzabili nei flussi di dati.

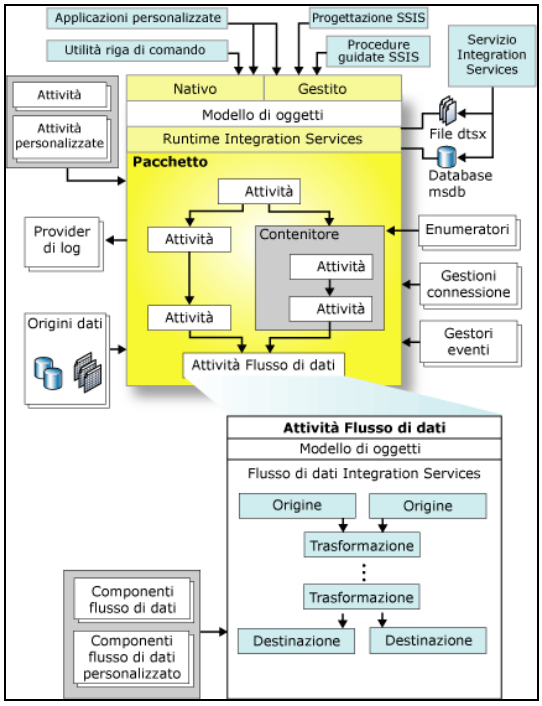


Figura 19. Architettura SSIS

Il pacchetto costituisce l'oggetto di Integration Services più importante , ovvero l'unità di lavoro che viene recuperata, eseguita e salvata. Esso è un insieme organizzato di:

- ✓ **elementi del flusso di controllo**, ovvero attività e contenitori, per la creazione del flusso di controllo di un pacchetto. Gli elementi del flusso di controllo consentono di preparare o copiare dati, interagire con altri processi o implementare un flusso di lavoro ripetuto. I vincoli di precedenza dispongono gli elementi in sequenza, in modo da creare un flusso di controllo ordinato, e specificano le condizioni per l'esecuzione di attività e contenitori. I contenitori dei flussi di lavoro forniscono la necessaria struttura ai pacchetti e i necessari servizi alle attività. Supportano la ripetizione dei flussi di controllo nei pacchetti e consentono di raggruppare attività e contenitori in unità di lavoro significative;

- ✓ **componenti del flusso di dati**, ovvero origini, trasformazioni e destinazioni, per la creazione di flussi di dati in un pacchetto che estrae, trasforma e carica dati. I percorsi collegano tali componenti in modo da formare un flusso di dati ordinato;

- ✓ **gestioni connessioni**, che consentono di connettersi a diversi tipi di origini dei dati per estrarre e caricare dati;

- ✓ **variabili**, che possono essere utilizzate nelle espressioni per aggiornare dinamicamente valori di colonne ed espressioni di proprietà, controllare l'esecuzione di flussi di controllo ripetuti e definire le condizioni applicate dai vincoli di precedenza;

- ✓ **gestori di eventi**, che vengono eseguiti in risposta agli eventi generati in fase di esecuzione da pacchetti, attività e contenitori;

- ✓ **provider di log**, che supportano la registrazione delle informazioni relative alla fase di esecuzione dei pacchetti, ad esempio l'ora di inizio e di fine dell'esecuzione di un pacchetto e dei suoi contenitori e attività.

Tutti questi elementi possono essere assemblati a livello utilizzando gli strumenti di progettazione grafica disponibili tramite l'ambiente grafico dell'SSIS. Al momento della creazione un pacchetto è un oggetto vuoto, privo di funzionalità. Per aggiungere funzionalità a un pacchetto è necessario aggiungervi un flusso di controllo ed uno o più flussi di dati. Nella Figura 20 viene illustrato un semplice pacchetto che contiene un flusso di controllo con un'attività Flusso di dati, che a sua volta contiene un flusso di dati.

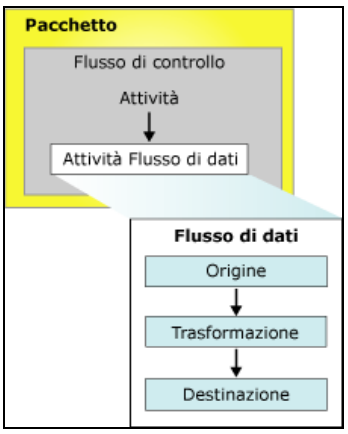


Figura 20. Attività pacchetto SSIS

Dopo aver creato il pacchetto base è possibile estenderne le capacità aggiungendovi ad esempio variabili e funzionalità di registrazione. Il pacchetto completo può essere quindi configurato impostando proprietà a livello di pacchetto che implementano la protezione, consentono il riavvio del pacchetto da un checkpoint o l'incorporamento di transazioni nel flusso di lavoro. Un pacchetto include in genere almeno una gestione connessione, ossia un collegamento tra un pacchetto e un'origine dei dati, che definisce

la stringa di connessione per l'accesso ai dati utilizzati dalle attività, dalle trasformazioni e dai gestori di eventi del pacchetto. Integration Services include tipi di connessione per origini dei dati quali file XML e di testo, database relazionali e database e progetti di Analysis Services.

In Integration Services sono disponibili strumenti grafici e procedure guidate per la creazione e il debug di pacchetti, attività per l'esecuzione di funzioni di flusso di lavoro quali operazioni FTP, messaggi di posta elettronica ed esecuzione di istruzioni SQL, origini dei dati e destinazioni per l'estrazione e il caricamento dei dati, trasformazioni per la pulitura, l'aggregazione, l'unione e la copia dei dati, il servizio di gestione Integration Services, per l'amministrazione dei pacchetti di Integration Services, nonché alcune API (Application Programming Interface) per la programmazione del modello di oggetti di Integration Services.

L'interfaccia utente (Figura 21) di progettazione consente di creare e configurare pacchetti mediante operazioni di trascinamento della selezione (come ad esempio origini e destinazioni dati, funzioni di trasformazione) e gestendo le opzioni nelle finestre di dialogo per ogni oggetto del pacchetto.

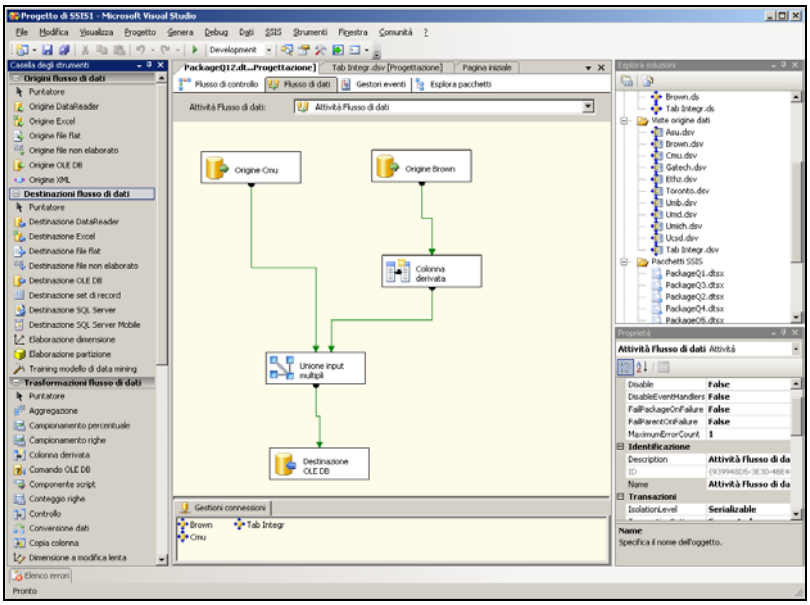


Figura 21. Interfaccia di progettazione SSIS

Come detto in precedenza nell'SSIS sono disponibili tre diversi tipi di componenti dei flussi di dati: origini, trasformazioni e destinazioni. Le origini estraggono dati da archivi dati quali tabelle e viste di database relazionali, file e database di Analysis Services.

Tramite le trasformazioni è possibile modificare, riepilogare e pulire i dati. Le destinazioni consentono di caricare dati in archivi dati o di creare set di dati in memoria. Questi componenti vengono poi collegati tra di loro, in fase di progettazione, tramite percorsi che connettono l'output di un componente all'input di un altro componente. I percorsi definiscono la sequenza dei componenti e consentono di aggiungere annotazioni al flusso di dati o visualizzare l'origine di una colonna. Nella Figura 22 viene illustrato un flusso di dati che include un'origine, una trasformazione con un input e un output e una destinazione. Oltre alle colonne di input, di output ed esterne, la figura include anche gli input, gli output e gli output degli errori.

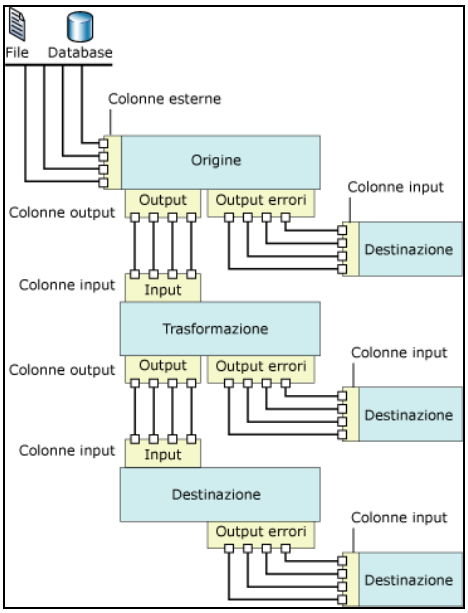


Figura 22. Flusso dati nel processo di integrazione

2.7.1 Origini e viste di dati

Un'origine è il componente di un flusso di dati che rende disponibili i dati di sorgenti esterne ad altri componenti del flusso di dati nel progetto di integrazione. L'origine di un flusso di dati include in genere un output regolare, le cui colonne sono costituite dalle colonne aggiunte dall'origine al flusso di dati. I metadati delle colonne esterne includono informazioni quali nome, tipo di dati e lunghezza della colonna di origine. L'output degli errori di un'origine contiene le stesse colonne dell'output regolare e due colonne aggiuntive che forniscono informazioni sugli errori. Il modello di oggetti di Integration Services non prevede alcun limite al numero degli output regolari e degli

errori di un'origine. La maggior parte delle origini disponibili in Integration Services, ad eccezione del componente script, include un output regolare e molte origini includono anche un output degli errori. Tutte le colonne di output sono disponibili come colonne di input per il componente successivo del flusso di dati. Le origini disponibili sono elencate nella tabella seguente:

Origine	**Descrizione**
Origine DataReader	Consente di utilizzare dati da un provider di dati .NET Framework.
Origine Excel	Estrae dati da un file di Excel.
Origine file flat	Estrae dati da un file flat.
Origine OLE DB	Consente di utilizzare dati da un provider OLE DB.
Origine file non elaborato	Estrae dati non elaborati da un file.
Componente script	Utilizza script per estrarre, trasformare o caricare dati.
Origine XML	Estrae dati da un file XML.

Poiché le origini dei dati non vengono create nel contesto di un pacchetto, una stessa origine dei dati può essere utilizzata da più pacchetti. Per sincronizzare le gestioni connessioni che fanno riferimento all'origine dei dati è pertanto sufficiente aggiornare l'origine dei dati. In pratica un'origine dati rappresenta una semplice connessione a un archivio dati e include tutte le tabelle e le viste presenti in quest'ultimo. Per funzionalità più avanzate, come la selezione di oggetti di database specifici quali tabelle e viste o l'aggiunta di nuove relazioni tra gli oggetti, è necessario utilizzare una vista origine dati anziché un'origine dei dati. Una vista origine dati è un documento che descrive lo schema di un'origine dei dati sottostante, specificando una selezione di oggetti di database denominata, esplorabile e persistente, che può essere utilizzata per definire origini, destinazioni e tabelle di ricerca per attività, trasformazioni, origini dei dati e destinazioni. In Integration Services una vista origine dati è un oggetto della modalità progettazione che semplifica l'implementazione di una stessa origine dei dati in più pacchetti; essendo basata su un'origine dei dati, pertanto, per utilizzare una vista origine dati in un pacchetto è necessario aggiungere l'origine dei dati al pacchetto. È possibile estendere una vista origine dati aggiungendo colonne calcolate popolate tramite espressioni personalizzate, aggiungendo nuove relazioni tra le tabelle, sostituendo le tabelle nella vista origine dati tramite query e aggiungendo tabelle correlate. È inoltre

possibile applicare un filtro a una vista origine dati per specificare un subset dei dati selezionati. Quando un componente di Integration Services, ad esempio un'origine o una trasformazione, utilizza una vista origine dati, la definizione della vista viene convertita in un'istruzione SQL e archiviata in una proprietà del componente. Poiché una vista non viene creata nel contesto di un pacchetto specifico, può essere utilizzata dalle attività e dai componenti del flusso di dati in più pacchetti.

2.7.2 Trasformazioni

Le trasformazioni possono avere funzionalità molto diverse. Possono eseguire attività quali l'aggiornamento, il riepilogo, la pulitura, 'unione e la distribuzione dei dati. Gli input e gli output di una trasformazione definiscono le colonne dei dati in entrata e in uscita. Grazie alle trasformazioni è possibile creare pacchetti con flussi di dati complessi senza dover scrivere codice. Alcune trasformazioni includono un singolo input e più output, mentre altre includono più input e un singolo output, a seconda delle operazioni eseguite sui dati. Le trasformazioni possono includere anche output degli errori che forniscono informazioni sugli errori generati, oltre ai dati che li hanno provocati, ad esempio dati stringa che non è stato possibile convertire in un tipo di dati `Integer`. Il modello di oggetti di Integration Services non prevede alcun limite al numero degli input, degli output regolari e degli output degli errori di una trasformazione. È possibile creare trasformazioni personalizzate che implementano qualsiasi combinazione di input, output regolari e output degli errori. L'input di una trasformazione è definito come una o più colonne di input. Alcune trasformazioni possono inoltre fare riferimento a colonne esterne come input. L'input della trasformazione “Comando OLE DB”, ad esempio, include colonne esterne. Una colonna di output è una colonna che la trasformazione aggiunge al flusso di dati. Sia gli output regolari che gli output degli errori possono contenere colonne di output. Le colonne di output possono essere a loro volta utilizzate come colonne di input per il componente successivo nel flusso di dati, che può essere costituito da un'altra trasformazione o da una destinazione.

Le trasformazioni dell'SSIS si suddividono in 4 categorie principali:

- ✓ **trasformazioni di Business Intelligence**, che eseguono operazioni quali la pulitura dei dati, il text mining e l'esecuzione di query di stima basate su un modello di data mining:

Trasformazione	Descrizione
Trasformazione Raggruppamento fuzzy	Trasformazione che standardizza i valori nei dati delle colonne.
Trasformazione Ricerca fuzzy	Trasformazione che ricerca valori in una tabella di riferimento utilizzando una corrispondenza fuzzy.
Trasformazione Estrazione termini	Trasformazione che estrae termini da un testo.
Trasformazione Ricerca termini	Trasformazione che ricerca termini in una tabella di riferimento e conta i termini estratti dal testo.
Trasformazione Query di data mining	Trasformazione che esegue query di stima basate su un modello di data mining.

- ✓ **trasformazioni a livello di riga**, che aggiornano i valori delle colonne e creano nuove colonne. La trasformazione viene applicata a tutte le righe dell'input;

Trasformazione	Descrizione
Trasformazione Mappa caratteri	Trasformazione che applica funzioni per i valori stringa a dati di tipo carattere.
Trasformazione Copia colonna	Trasformazione che aggiunge copie delle colonne di input al proprio output.
Trasformazione Conversione dati	Trasformazione che converte il tipo di dati di una colonna in un tipo di dati diverso.
Trasformazione Colonna derivata	Trasformazione che popola colonne con risultati di espressioni.
Componente script	Trasformazione che utilizza script per estrarre, trasformare o caricare dati.
Trasformazione Comando OLE DB	Trasformazione che esegue comandi SQL per ogni riga di un flusso di dati.

- ✓ **trasformazioni a livello di set di righe**, che creano nuovi set di righe. Il set di righe può includere valori aggregati e ordinati, set di righe campione o set di righe trasformati tramite Pivot o UnPivot:

Trasformazione	Descrizione
Trasformazione Aggregazione	Trasformazione che esegue aggregazioni quali AVERAGE, SUM e COUNT.
Trasformazione Ordinamento	Trasformazione che ordina i dati.
Trasformazione Campionamento percentuale	Trasformazione che consente di creare un set di dati campione utilizzando una percentuale per specificare le dimensioni del campione.
Trasformazione Campionamento righe	Trasformazione che consente di creare un set di dati campione specificando il numero di righe del campione.
Trasformazione Pivot	Trasformazione che crea una versione meno normalizzata di una tabella normalizzata.
Trasformazione UnPivot	Trasformazione che crea una versione più normalizzata di una tabella non normalizzata.

- ✓ **trasformazioni di divisione e di unione,** che distribuiscono le righe tra output diversi, creano copie degli input della trasformazione, uniscono più input in un singolo output ed eseguono operazioni di ricerca:

Trasformazione	Descrizione
Trasformazione Suddivisione condizionale	Trasformazione che indirizza righe di dati verso output diversi.
Trasformazione Multicast	Trasformazione che distribuisce set di dati tra più output.
Trasformazione Unione input multipli	Trasformazione che unisce più set di dati.
Trasformazione Unione	Trasformazione che unisce due set di dati ordinati.
Trasformazione Merge join	Trasformazione che unisce due set di dati utilizzando un join di tipo FULL, LEFT o INNER.
Trasformazione Ricerca	Trasformazione che ricerca valori in una tabella di riferimento utilizzando una corrispondenza esatta.

- ✓ **altre trasformazioni**, che consentono di esportare e importare dati, aggiungere informazioni di controllo, contare righe e utilizzare dimensioni a modifica lenta.

Trasformazione	Descrizione
Trasformazione Esporta colonna	Trasformazione che inserisce in un file dati esportati da un flusso di dati.
Trasformazione Importa colonna	Trasformazione che legge dati da un file e li aggiunge a un flusso di dati.
Trasformazione Controllo	Trasformazione che rende le informazioni sull'ambiente disponibili a un flusso di dati di un pacchetto.
Trasformazione Conteggio righe	Trasformazione che conta le righe che la attraversano e archivia il totale in una variabile.
Trasformazione Dimensione a modifica lenta	Trasformazione che configura l'aggiornamento di una dimensione a modifica lenta.

2.7.3 Destinazioni

Una destinazione è il componente di un flusso di dati che scrive dati dal flusso di dati a uno specifico archivio dati oppure crea un set di dati in memoria. Le destinazioni devono includere almeno un input. L'input contiene colonne di input, che provengono da un altro componente del flusso di dati e sono mappate alle colonne della destinazione. Molte destinazioni includono anche un output degli errori. L'output degli errori di una destinazione include colonne di output, che in genere contengono informazioni sugli errori generati durante la scrittura dei dati nell'archivio dati di destinazione. Gli errori possono verificarsi per motivi diversi. Ad esempio, una colonna contiene un valore *Null* mentre la colonna di destinazione corrispondente non può essere impostata su *Null*. Il modello di oggetti di Integration Services non prevede alcun limite al numero degli input regolari e degli output degli errori di una destinazione ed è possibile creare destinazioni personalizzate che implementano più input e output degli errori.

Nella tabella seguente vengono elencate le destinazioni disponibili:

Destinazione	Descrizione
Destinazione Training modello di data mining	Consente di eseguire il training di modelli di data mining.
Destinazione DataReader	Espone i dati in un flusso di dati tramite l'interfaccia ADO.NET DataReader.

Destinazione elaborazione dimensione	Carica ed elabora una dimensione di SQL Server 2005 Analysis Services (SSAS).
Destinazione Excel	Scrive i dati in una cartella di lavoro di Excel.
Destinazione file flat	Scrive dati in un file flat.
Destinazione OLE DB	Carica dati tramite un provider OLE DB.
Destinazione elaborazione partizione	Carica ed elabora una partizione di Analysis Services.
Destinazione file non elaborato	Scrive dati non elaborati in un file.
Destinazione set di record	Crea un set di record ADO.
Componente script	Utilizza script per estrarre, trasformare o caricare dati.
Destinazione SQL Server Mobile	Inserisce righe in un database di SQL Server Mobile.
Destinazione SQL Server	Inserisce i dati in massa in una tabella o vista di SQL Server 2005.

2.7.4 Elementi del "flusso dati"

Dopo aver collegato le sorgenti dati esterne, le tabelle di integrazione e testato le connessioni, si procede alla creazione del pacchetto, definendo all'interno un flusso dati che ha lo scopo di estrarre i dati dalle sorgenti, applicare le trasformazioni necessarie e caricare i dati nelle tabelle di integrazione, sulle quali eseguire poi le query del benchmark THALIA.

In particolare, per ogni query, come sorgente dati si fa uso di due moduli "origini dati OLE DB" che utilizza appunto le connessioni create in precedenza; mentre per mezzo del modulo "destinazione dati OLE DB", si caricano i dati trasformati nella tabella di integrazione. In Figura 23 è illustrato il flusso dati creato per popolare la tabella di integrazione utile per eseguire la prima query del benchmark, con le due origini dati GATECH e CMU, ossia le due connessioni ai database dai quali si estraggono i dati e la tabella di destinazione TQ1.

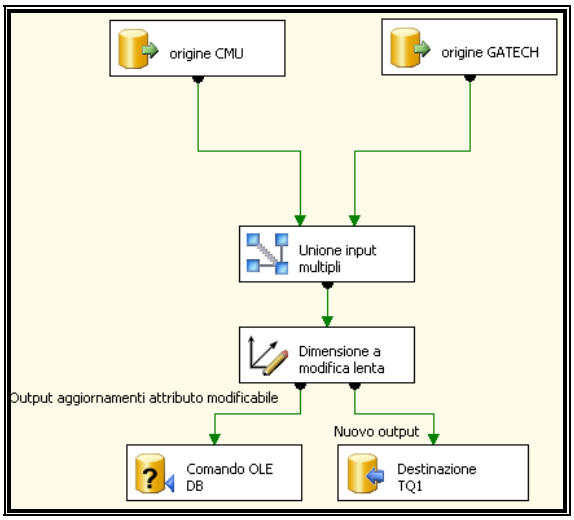


Figura 23. Flusso dati nel processo di integrazione della query 1

Nella creazione del flusso di dati, per ogni query, si è fatto uso di tre funzioni di trasformazione: "*Colonna derivata*", "*Unione input multipli*" e "*Dimensione a modifica lenta*".

La trasformazione "**Colonna derivata**" consente di creare nuovi valori di colonna tramite l'applicazione di espressioni alle colonne di input della trasformazione. Un'espressione può contenere qualsiasi combinazione di colonne dell'input della trasformazione, variabili, funzioni e operatori. Il risultato può essere aggiunto come nuova colonna o inserito in una colonna esistente come valore di sostituzione. Si possono definire più colonne derivate e qualsiasi variabile o colonna di input può comparire in più espressioni. È possibile utilizzare questa trasformazione per concatenare i dati di colonne diverse in una nuova colonna; estrarre caratteri da dati stringa, tramite funzioni quali SUBSTRING, e quindi archiviare il risultato in una colonna derivata; applicare funzioni matematiche a dati numerici e archiviare i risultati in una colonna derivata;creare espressioni che confrontano colonne di input e variabili. Per configurare la trasformazione *Colonna derivata*, si specifica un'espressione per ogni colonna di input o nuova colonna da modificare, si modifica il tipo di dati, si imposta la lunghezza di colonna dei dati stringa, nonché la scala e la precisione dei dati numerici. Questa trasformazione include un input, un output regolare e un output degli errori. Per creare le espressioni si utilizzano gli operatori e le funzioni messe a disposizione dall'SSIS.

La trasformazione "**Unione input multipli**" consente di combinare più input in un unico output. È ad esempio possibile utilizzare gli output di diverse origini dati come input per la trasformazione e combinarli in un singolo output. Gli input della

trasformazione vengono aggiunti all'output della trasformazione uno dopo l'altro, senza riordinare le righe. Se il pacchetto richiede un output ordinato, sarà necessario utilizzare la trasformazione *Unione* anziché la trasformazione *Unione input multipli*. L'output della trasformazione *Unione input multipli* viene creato a partire dal primo input connesso alla trasformazione. Le colonne negli input connessi successivamente alla trasformazione vengono mappate a quelle dell'output della trasformazione. Per unire gli input è necessario mappare le relative colonne alle colonne nell'output. È necessario mappare almeno una colonna di input a ogni colonna di output. Il mapping tra due colonne può essere eseguito solo se i relativi metadati corrispondono. Le colonne mappate devono ad esempio avere lo stesso tipo di dati. Se le colonne mappate contengono dati stringa e la lunghezza della colonna di output è inferiore a quella della colonna di input, la lunghezza della colonna di output verrà aumentata automaticamente in modo che possa contenere la colonna di input. Le colonne di input non mappate a colonne di output vengono impostate su valori *Null* nelle colonne di output.

La trasformazione *Dimensione a modifica lenta* coordina l'aggiornamento e l'inserimento dei record nelle tabelle utilizzate per l'integrazione dei dati, sulle quali poi eseguire le query del benchmark. Tramite questa trasformazione è possibile eseguire un confronto tra le righe in ingresso e le righe presenti nella tabella "destinazione" per distinguere tra righe nuove e righe esistenti; identificare le righe in ingresso contenenti modifiche non consentite; identificare i record che richiedono un aggiornamento; identificare le righe in ingresso contenenti modifiche cronologiche che richiedono l'inserimento di nuovi record e l'aggiornamento dei record scaduti; rilevare le righe in ingresso contenenti modifiche che richiedono l'aggiornamento dei record esistenti, compresi quelli scaduti. La trasformazione *Dimensione a modifica lenta* supporta quattro tipi di modifiche:

- ✓ Le modifiche di tipo "Attributo modificabile" sovrascrivono i record esistenti. Questo tipo di modifica è equivalente a una modifica di tipo 1. La trasformazione Dimensione a modifica lenta invia queste righe a un output denominato **Output aggiornamenti attributi modificabili**.
- ✓ Le modifiche di tipo "Attributo cronologico" creano nuovi record anziché aggiornare quelli esistenti. L'unica modifica consentita in un record esistente è l'aggiornamento di una colonna che indica se il record è aggiornato o scaduto. Questo tipo di modifica è equivalente a una modifica di tipo 2. La

trasformazione *Dimensione a modifica lenta* invia queste righe a due output: **Output inserimenti attributo cronologico** e **Nuovo output**.

- ✓ Le modifiche di tipo "Attributo fisso" indicano che il valore della colonna non deve essere modificato. La trasformazione Dimensione a modifica lenta rileva le modifiche e può inviare le righe modificate a un output denominato **Output attributo fisso**.

- ✓ "Membro derivato" indica che la riga è un record membro derivato nella tabella delle dimensioni. Un record membro derivato è un membro di dimensione sconosciuto. In attesa di dati di dimensione rilevanti, che verranno forniti in un caricamento successivo, viene creato un record membro derivato minimo. La trasformazione Dimensione a modifica lenta invia queste righe a un output denominato **Output aggiornamenti membro derivato**.

In fase di esecuzione la trasformazione *Dimensione a modifica lenta* cerca innanzitutto una corrispondenza tra la riga in ingresso e un record nella tabella di "destinazione". Se non viene trovata alcuna corrispondenza, significa che la riga in ingresso è un nuovo record. In questo caso la trasformazione non esegue altre operazioni e invia la riga a **Nuovo output**. Quando viene trovata una corrispondenza, la trasformazione *Dimensione a modifica lenta* verifica invece se sono presenti eventuali modifiche. Se la riga contiene modifiche, verrà identificato il tipo di aggiornamento eseguito sulle varie colonne e la riga verrà inviata a **Output aggiornamenti attributi modificabili**, **Output attributo fisso**, **Output inserimenti attributo cronologico** o **Output aggiornamenti membro derivato**. Se la riga non contiene modifiche, verrà inviata a **Output non modificato**. La trasformazione *Dimensione a modifica lenta* utilizza un solo input e un massimo di sei output. Un output indirizza una riga al subset del flusso di dati che corrisponde ai requisiti di aggiornamento e inserimento della riga. Inoltre questa trasformazione non supporta un output degli errori.

Di seguito verranno illustrati e descritti i flussi dati implementati per estrarre, trasformare e caricare i dati nelle 12 tabelle di integrazione, e come le query sono state eseguite su tali tabelle.

2.8 Configurazione e connessione alle origini dati

Il primo passo per l'implementazione delle query del benchmark nell'SSIS è stato quello di collegare le sorgenti dati esterne, dove sono memorizzati i dati necessari per l'esecuzione delle query stesse, all'interno del pacchetto. Tramite l'apposita interfaccia, vedi Figura 24, è stata creata una connessione OLE DB, che consente ad un pacchetto di connettersi a un'origine dei dati tramite un provider OLE DB. Inizialmente i dati necessari per il progetto sono memorizzati all'interno di database di SQL Server 2005, perciò, per effettuare la connessione ai database è stato utilizzato il provider Microsoft OLE DB per SQL Server.

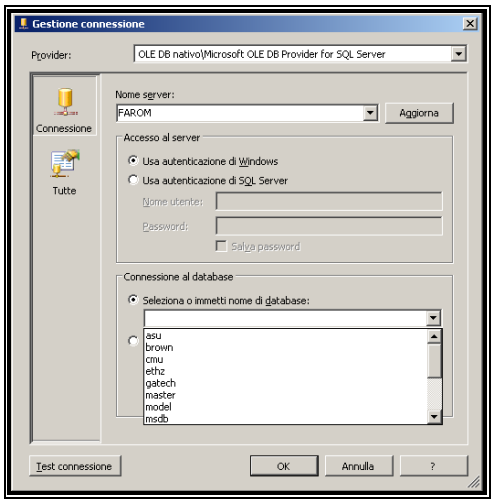


Figura 24. Interfaccia connessione sorgenti

Per configurare la connessione OLE DB, è necessario specificare il nome del server di MSQL Server e il nome del database al quale connettersi. Dopo aver definito tutte le connessioni ai database interessati dal progetto, per ogni origine dati è stata definita una vista di dati, necessaria per poter visualizzare le strutture dati presenti all'interno dei vari database (vedi Figura 25).

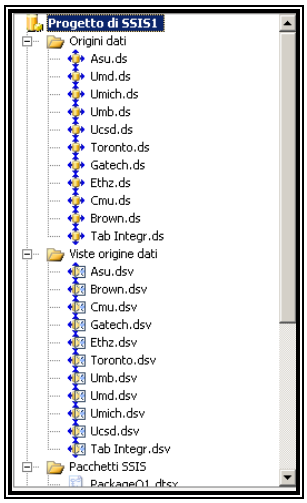


Figura 25. Vista origini dati nel progetto di integrazione

La gestione connessione OLE DB viene utilizzata da diversi componenti di flusso di dati e attività. L'origine e la destinazione OLE DB, ad esempio, utilizzano questa gestione connessione per estrarre e caricare i dati, mentre l'attività Esegui SQL può utilizzarla per connettersi a un database di SQL Server per l'esecuzione delle query.

Una volta definite tutte le connessioni, per implementare le 12 query del benchmark è necessario creare un pacchetto di lavoro per ognuna di esse, ed inoltre creare un database (collegandolo al progetto tramite connessione OLE DB) per memorizzare le relative 12 tabelle nelle quali eseguire l'integrazione dati.

2.9 Implementazione delle query del benchmark THALIA in SSIS

In questo pragrafo verrà descritto il modo in cui sono state implementate le query del benchmark THALIA nel'SSIS, con particolare riferimento ai moduli e funzioni utilizzati per ognuna delle query. Verranno illustrati i moduli utilizzati per i processi di integrazione con relative funzioni di trasformazione.

Query 1

Nel flusso di dati per la prima query (vedi Figura 23) non è necessaria la funzione di trasformazione "Colonna derivata", è necessario utilizzare la funzione "Unione input multipli". Nella seguente tabella è illustrato il mapping tra le colonne di input, delle due sorgenti, e le colonne di output:

Colonne di input (sorgente CMU)	Colonne di input (sorgente GATECH)	Colonne di output (su elemento succ.)
Code	**CRN**	**Code**
CourseTitle	**Title**	**CourseTitle**
Lecturer	**Intructor**	**Lecturer**

Mapping effettuato dala funzione "Unione input multipli"

In questo modo tutti i record della prima sorgente verrano uniti ai record della seconda sorgente. Per default il nome delle colonne di output hanno lo stesso nome delle colonne della prima sorgente che viene collegata alla funzione di trasformazione. Le colonne di output a loro volta, diventano le colonne di input per la funzione di trasformazione successiva, "Dimensione a modifica lenta", che imposta il vincolo di modifica attributo solo sul campo "prof". Inoltre le colonne ricevute in input vengono mappate in quelle di output, ossia le colonne della tabella di integrazione "TQ1", con le seguenti associazioni:

Colonne di input (out funz. "Unione input multipli")	Colonne di output (TQ1)
Code	**cod**
CourseTitle	**nomeCorso**
Lecturer	**prof**

Mapping effettuato dalla funzione "Dimensione a modifica lenta"

Con questo mapping si popola la prima tabella di integrazione (TQ1), sulla quale è possibile eseguire la seguente query:

```
select cod,nomeCorso,prof from TQ1 where prof like
'%Clark%'
```

Query 2

Per questa query è necessaria anche la funzione di trasformazione "Colonna derivata. Si può notare, osservando il flusso dati in Figura 26, che sono state implementate due funzioni di trasformazioni "Colonna derivata" sul percorso dati della sorgente UMB. Per risolvere l'eterogeneità dei dati del campo "Times" (formato 0-24) rispetto al campo "Time" (formato 0-12) della sorgente CMU, nella prima funzione "Colonna derivata" vengono estratti dal campo "Times" gli intervalli orari (es. 13:30-14:30) e trasformati nel formato compatibile con la sorgente CMU (es. 1:30-2:30), suddividendo la stringa in quattro sotto-intervalli (ore-minuti-ore-minuti) e applicando le espressioni di trasformazione per ogni sottointervallo; mentre nella seconda funzione derivata viene fatto il *merge* dei sottointervalli, ricomponendo l'intervallo orario originale.

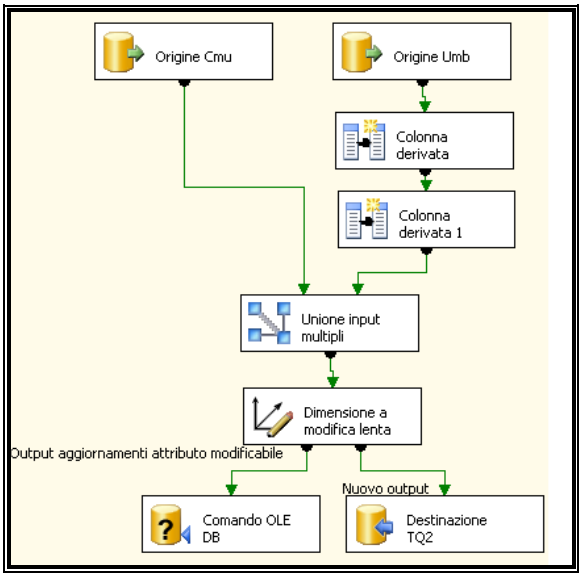


Figura 26. Flusso dati query 2

Nella seguente tabella sono illustrate le colonne in ingresso e in uscita della prima funzione "Colonna derivata". Tra le parentesi è indicato un esempio di scomposizione della stringa con relativa trasformazione:

Colonne di input (sorgente UMB)	Colonne di output (su elemento succ.)
TitleCredits	**TitleCredits**
Times (Es. "13:30-14:45")	**Ora1 ("1")** **Ora2 (":30-")** **Ora3 ("2")** **Ora4 (":45")**

Le espressioni di trasformazione associate ai campi Ora1, Ora2, Ora3, Ora4 sono le seguenti:

```
//Funzione sul campo Ora1
((LEN(Times) == 9) || (LEN(Times) == 7)) ? Times :
(((DT_UI1)SUBSTRING(Times,1,2) > 12) ?
(DT_WSTR,2)(((DT_UI1)SUBSTRING(Times,1,2)) - 12) :
SUBSTRING(Times,1,2))
//Funzione sul campo Ora2
((LEN(Times) == 9) || (LEN(Times) == 7)) ? " " :
SUBSTRING(Times,3,4)
//Funzione sul campo Ora3
((LEN(Times) == 9) || (LEN(Times) == 7)) ? " " :
(((DT_UI1)SUBSTRING(Times,7,2) > 12) ?
(DT_WSTR,3)(((DT_UI1)SUBSTRING(Times,7,2)) - 12) :
SUBSTRING(Times,7,2))
//Funzione sul campo Ora4
((LEN(Times) == 9) || (LEN(Times) == 7)) ? " " :
SUBSTRING(Times,9,3)
```

Nella seguente tabella sono illustrate invece le colonne di input e di output della seconda funzione “Colonna derivata”, che non fa altro che concatenare il contenuto dei campi Ora1, Ora2, Ora3, Ora4 :

Colonne di input	Colonne di output
TitleCredits	**TitleCredits**
Ora1	
Ora2	**Orario**
Ora3	
Ora4	

Successivamente, le colonne di output della seconda funzione “Colonna derivata” saranno uno dei due input della funzione “Unione input multipli”, l’altro sarà dato dalle colonne originale della sorgente CMU:

Colonne di input (sorgente CMU)	Colonne di input (sorgente UMB)	Colonne di output (su elemento succ.)
CourseTitle	**Title**	**CourseTitle**
Time	**Orario**	**Time**

Mapping effettuato dala funzione "Unione input multipli"

In questo modo tutti i record della prima sorgente verranno uniti ai record della seconda sorgente. Le colonne di output a loro volta, diventano le colonne di input per la funzione di trasformazione successiva, “Dimensione a modifica lenta”, che imposta il vincolo di modifica attributo solo sul campo “orario”. Inoltre le colonne ricevute in input vengono mappate in quelle di output, ossia le colonne della tabella di integrazione “TQ1”, con le seguenti associazioni:

Colonne di input (out funz. “Unione input multipli”)	Colonne di output (TQ2)
CourseTitle	**nomeCorso**
Time	**orario**

Mapping effettuato dalla funzione "Dimensione a modifica lenta"

Con questo mapping si popola la seconda tabella di integrazione (TQ2), sulla quale è possibile eseguire la seguente query:

```
SELECT nomeCorso,orario FROM TQ2 WHERE  (orario LIKE
'1:30%')
```

Query 3

Per questa query non è necessaria la funzione di trasformazione “Colonna derivata”, è necessario utilizzare solo la funzione “Unione input multipli”. Nella seguente tabella è illustrato il mapping tra le colonne di input, delle due sorgenti, e le colonne di output:

Colonne di input (sorgente UMD)	Colonne di input (sorgente BROWN)	Colonne di output (su elemento succ.)
Code	**Code**	**Code**
CourseName	**Title**	**Title**

Mapping effettuato dalla funzione "Unione input multipli"

Le colonne di output saranno l’input per la funzione di trasformazione successiva, “Dimensione a modifica lenta”, che carica i dati nella tabella “TQ3”, con le seguenti associazioni:

Colonne di input (out funz. “Unione input multipli”)	Colonne di output (TQ3)
Code	**cod**
Title	**nomeCorso**

Con questo mapping si popola la terza tabella di integrazione (TQ3), sulla quale è possibile eseguire la seguente query:

```
select cod,nomeCorso from TQ3 where nomeCorso like
'%Data Structures%'
```

Query 4

Per implementare il flusso dati per la query 4 sono necessarie due trasformazioni "Colonna derivata", in particolare una agisce sul campo "Umfang" della sorgente ETHZ per estrarre il numero dei crediti (espressi in forma letteraria nel suddetto campo, di tipo stringa) con la formula "#u+#v+1", traducendoli in un numero intero (quindi trasformazione anche del tipo di dato); la seconda invece agisce sul campo "Units" di CMU per tradurre il tipo di dato da stringa ad intero, per rispettare la compatibilità di tipi di dato nel mappagio delle colonne nella tabella di integrazione finale. Le espressioni delle rispettive funzioni di trasformazione sono:

```
//Funzione sul campo "Umfang"
LEN(Umfang) == 2 ? 0 :
(DT_UI1)(SUBSTRING(Umfang,FINDSTRING(Umfang,"V",1) - 1,1)) +
(DT_UI1)(SUBSTRING(Umfang,FINDSTRING(Umfang,"U",1) - 1,1)) +
1)
//Funzione sul campo "Units"
```

Una volta applicate le trasformazioni, derivando le nuove colonne, si può utilizzare la funzione "Unione input multipli" per unire i dati trasformati delle due sorgenti e popolare la quarta tabella di integrazione (TQ4), sulla quale è possibile eseguire la seguente query:

```
SELECT     nomeCorso, crediti
FROM       TQ4
WHERE      (nomeCorso LIKE '%Database%')
AND (crediti >= 5)
```

Query 5

Per la query 5 è necessaria una trasformazione "Colonna derivata" che agisce sul campo "Titel" della sorgente ETHZ per risolvere il problema dell'eterogeneità di lingua, in quanto l'informazione è espressa in lingua tedesca a differenza della sorgente UMD (lingua inglese). L'espressione di trasformazione applicata è la seguente:

```
(FINDSTRING(Titel,"Datei",1) > 0) ? "Database" :
((FINDSTRING(Titel,"Datenbasis",1) > 0) ? "Database" :
Titel)
```

Una volta derivata la nuova colonna per la sorgente ETHZ, si può utilizzare la funzione "Unione input multipli" per unire i dati della sorgente UMD e ETHZ e popolare la quinta tabella di integrazione (TQ5).

Colonne di input (sorgente UMD)	Colonne di input (sorgente BROWN)	Colonne di output (TQ5)
Code	**Typ**	**codice**
CourseName	**Title (*trasformato*)**	**nomeCorso**

Query eseguita:

```
SELECT     codice, nomeCorso
FROM       TQ5
WHERE     (nomeCorso LIKE '%Database%')
```

Query 6

Per la query 6 è necessario unire i dati della sorgente TORONTO, che contiene sia il campo "title" che il campo "text", con i dati della sorgente CMU che non contiene però l'informazione corrispondente al campo "text". Per ovviare a questo problema, tramite la funzione "Unione input multipli" è possibile mappare il campo "title" della sorgente TORONTO con il campo "CourseTitle" di CMU e ignorare il mapping del campo "text"; infatti le colonne di input, della sorgente CMU, non mappate a colonne di output vengono impostate su valori *Null* nelle colonne di output.

Colonne di input (sorgente CMU)	Colonne di input (sorgente TORONTO)	Colonne di output (TQ6)
<ignore>	**text**	**testoRif**
CourseTitle	**title**	**nomeCorso**

Query eseguita:

```
SELECT     nomeCorso, testoRif
FROM       TQ6
WHERE     (nomeCorso LIKE '%Verification%')
```

Query 7

Per implementare questa query vi è ancora un problema di valori nulli da risolvere in fase di integrazione. Nella sorgente ASU non vi è espressa in maniera esplicita l'informazione relativa sui prerequisiti dei corsi, informazione invece presente

esplicitamente nel campo “prerequisite” della sorgente UMICH. L’informazione è ricavata dal campo “Description” della sorgente ASU tramite la funzione di trasformazione “Colonna derivata”, grazie alla quale è possibile definire una nuova colonna “prereq” associando ad essa l’espressione di trasformazione:

```
FINDSTRING(Description,"Prerequisite",1) > 0 ?
RIGHT(Description,(LEN(Description) -
FINDSTRING(Description,"Prerequisite",1) + 13)) : "none"
```

Inoltre sono mantenute le colonne originali della sorgente ASU insieme alla nuova colonna. Quindi, è possibile unire i dati ottenuti con quelli della sorgente UMICH e popolare la tabella di integrazione relativa alla settima query del benchmark.

Colonne di input (sorgente ASU)	Colonne di input (sorgente UMICH)	Colonne di output (TQ7)
Title	**name**	**nomeCorso**
Description	**description**	**descr**
prereq (*nuova colonna*)	**prerequisite**	**prereq**

Query eseguita:

```
SELECT     nomeCorso, descr, prereq
FROM         TQ7
WHERE     (prereq LIKE '%none%')
```

Query 8

Nel processo di integrazione per risolvere questa query vi è un problema analogo a quello riscontrato nell’implementazione della query 6. Infatti nella tabella della sorgente ETHZ manca l’informazione sull’accessibilità dei corsi, presente invece nella sorgente GATECH nell’attributo “Description”. Il problema è stato risolta anche in questo caso con la trasformazione “Unione”, mappando solo l’attributo della sorgente GATECH in uscita.

Colonne di input (sorgente ETHZ)	Colonne di input (sorgente GATECH)	Colonne di output (TQ6)
<ignore>	**Description**	**spec**
Titel	**Title**	**nomeCorso**

Query eseguita:

```
SELECT     nomeCorso, spec
FROM       TQ8
WHERE     (spec LIKE '%JR%')
```

Query 9

La particolarità di questa query consiste nel fatto che le informazioni da estrarre dalla sorgenti BROWN e UMD risiedono in più tabelle, non in una solo tabella come nelle query precedenti. In particolare, nella sorgente UMD, per estrarre le informazioni relative al nome del corso e quelle delle aule del corso è necessario fare il join tra due tabelle presenti su questa sorgente dati: la tabella "Course" e la tabella "Section". La condizione di join è stata specificata grazie al modulo "Origini dati" della sorgente UMD, che tra le varie opzioni di configurazione c'è la possibilità di specificare un comando SQL (vedi Figura 27) da eseguire sulle strutture dati della sorgente stessa, per ottenere così una tabella risultante contenente i dati di interesse per la particolare estrazione.

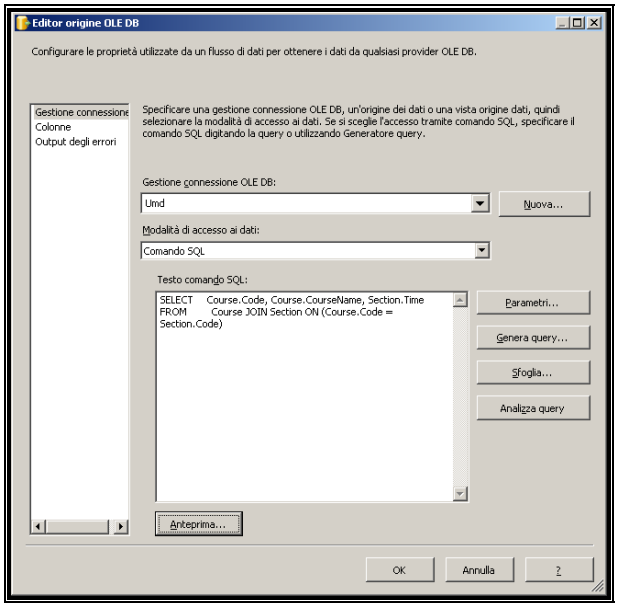


Figura 27. Interfaccia per inserimento comando SQL

Il comando SQL eseguito è il seguente:

```
SELECT     Course.Code, Course.CourseName, Section.Time
FROM         Course JOIN Section ON (Course.Code =
Section.Code)
```

Inoltre è necessario applicare anche la trasformazione "Colonna derivata" per estrarre le informazione relative all'aula dal campo "Time" della sorgente UMD, in quanto in questo campo vi sono anche informazioni sul giorno e ora della lezione. Per questo motivo, con la trasformazione si crea una nuova colonna, "room", contenente solo

l'informazione sulla aula. L'espressione di trasformazione associata al campo "Time" è la seguente:

```
FINDSTRING(Time,"(",1)>0 ?
SUBSTRING(Time,FINDSTRING(Time,"(",1),30) : Time
```

A questo punto si può procedere con l'unione dei dati trasformati della sorgente UMD con i dati di BROWN e caricarli nella tabella di integrazione, eseguendo il seguente mapping fra gli schemi delle due sorgenti:

Colonne di input (sorgente UMD)	Colonne di input (sorgente BROWN)	Colonne di output (TQ9)
Code	**Code**	**cod**
CourseName	**Title**	**nomeCorso**
room (*nuova colonna*)	**Room**	**aula**

Query eseguita:

```
SELECT     cod, nomeCorso, aula
FROM       TQ9
WHERE     (nomeCorso LIKE '%Software Engineering%')
```

Query 10

Anche per questa query è necessario fare il join tra la tabella "Course" e la tabella "Section" presenti sulla stessa sorgente (UMD). La condizione di join aspressa è:

```
SELECT     Course.Code, Course.CourseName, Section.Title
FROM         Course JOIN Section ON (Course.Code =
Section.Code)
```

Infatti l'informazione sui professori è contenuta nel campo "Title", dal quale deve essere estratta con un'appropriata espressione di trasformazione, in quanto vi sono altre informazioni memorizzate in questo campo, oltre a quelle relative ai professori. Con la trasformazione si crea una nuova colonna, "profExtract", contenente solo l'informazione sui professori che tengono un particolare corso. La trasformazione associata al campo "Title" è la seguente:

```
FINDSTRING(Title,"(",2) > 0 ?
SUBSTRING(Title,FINDSTRING(Title,")",1) +
1,FINDSTRING(Title,"(",2) - (FINDSTRING(Title,")",1) + 1)) :
"none"
```

A questo punto si può procedere con l'unione dei dati trasformati della sorgente UMD con i dati della sorgente CMU e caricarli nella tabella di integrazione, eseguendo il seguente mapping fra gli schemi delle due sorgenti:

Colonne di input (sorgente UMD)	Colonne di input (sorgente CMU)	Colonne di output (TQ10)
Code	**Code**	**cod**
CourseName	**CourseTitle**	**nomeCorso**
profExtract (*nuova colonna*)	**Lecturer**	**professore**

Query 11

In questa query è necessario eseguire un concatenamento di dati di colonne diverse in una nuova colonna. Nella sorgente UCSD infatti le informazioni sui professori, che tengono determinati corsi, sono memorizzate in 3 campi diversi; quindi risulta utile creare una nuova colonna "prof", contenente i dati sui professori concatenando le tre colonne (Fall2003, Winter2004, Spring2004).

Nella seguente tabella sono illustrate le colonne di input e di output della funzione "Colonna derivata", che non fa altro che concatenare il contenuto dei campi:

Colonne di input	Colonne di output
Title **Fall2003** **Winter2004** **Spring2004**	**Title** **prof (*nuova colonna*)**

Espressione di trasformazione:

```
Fall2003 + " " + Winter2004 + " " + Spring2004
```

Mapping degli attributi delle due sorgenti (UCSD e CMU) sulla tabella di integrazione:

Colonne di input (sorgente UCSD)	Colonne di input (sorgente CMU)	Colonne di output (TQ11)
Title	**CourseTitle**	**nomeCorso**
prof (*nuova colonna*)	**Lecturer**	**prof**

Query eseguita sulla tabella di integrazione TQ11:

```
SELECT     nomeCorso, prof
FROM       TQ11
WHERE     (nomeCorso LIKE '%Database%')
```

Query 12

Per implementare l'ultima query si deve applicare una funzione di trasformazione sui dati della sorgente BROWN e precisamente sul campo "Title" che contiene informazioni sul nome, giorno e ora dei corsi. L'obiettivo consiste nell'estrarre in

maniera separata queste informazioni, dallo stesso campo, e suddividerle in 3 diverse nuove colonne per rendere compatibile la struttura dei dati con quella della sorgente CMU. Per far questo, si applicano le seguenti espressioni di trasformazione, con il modulo "Colonna Derivata":

```
//per estrarre il nome del corso
FINDSTRING(Title,"hr",1) > 0 ?
SUBSTRING(Title,FINDSTRING(Title,"/",5) +
2,(FINDSTRING(Title,"hr",1) - FINDSTRING(Title,"/",5))) :
Title

//per il giorno del corso
SUBSTRING(Title,FINDSTRING(Title,"hr.",1) +
4,FINDSTRING(SUBSTRING(Title,FINDSTRING(Title,"hr.",1) +
4,10)," ",1))

//per l'ora del corso
SUBSTRING(Title,FINDSTRING(SUBSTRING(Title,FINDSTRING(Title,"h
r.",1) + 4,10)," ",1) + FINDSTRING(Title,"hr.",1) + 4,15)
```

Tabella di trasformazione:

Colonne di input (sorgente BROWN)	Colonne di output (funzione Colonna derivata)
Title	**nomeC** **gior** **ora**

Ora è possibile unire la struttura dati in uscita dalla funzione di trasformazione con i dati della sorgente CMU e popolare la tabella di integrazione relativa alla dodicesima query del benchmark.

Colonne di input (sorgente CMU)	Colonne di input (sorgente BROWN)	Colonne di output (TQ7)
CourseTitle	**nomeC (*nuova colonna*)**	**nomeCorso**
Day	**gior (*nuova colonna*)**	**giornoLez**
Time	**ora (*nuova colonna*)**	**orario**

Query eseguita:

```
SELECT     nomeCorso, giornoLez, orario
FROM       TQ12
WHERE     (nomeCorso LIKE '%Computer%Network%')
```

Capitolo 3

Il progetto MOMIS

3.1 Architettura MOMIS

MOMIS (*Mediator EnvirOnment for Multiple Information Sources*) è un sistema intelligente di integrazione di informazioni da sorgenti di dati strutturati e semistrutturati, la cui realizzazione e` iniziata nel 1998 all'interno del progetto *MURST INTERDATA* attraverso la collaborazione tra i gruppi operativi dell'Università di Modena e Reggio Emilia e di quella di Milano. Come descritto all'interno dei lavori *SIGMOD Record* nella pubblicazione "Semantic Integration of Semistructured and Structured Data Sources", di S.Bergamaschi, S.Castano e M.Vincini [20], il contributo innovativo di questo progetto rispetto ad altri simili risiede nella fase di analisi ed integrazione degli schemi sorgenti, realizzata in maniera semiautomatica; inoltre un lavoro molto approfondito è stato svolto nella fase di query processing, responsabile della traduzione della query generale nelle sottoquery da inviare alle diverse sorgenti di dati.

Il continuo aumento delle possibili fonti di informazione accessibili (sia attraverso internet che mediante reti locali come quelle aziendali) ha portato all'esigenza di creare sistemi che, oltre a facilitare il reperimento delle informazioni cercate, siano in grado di proporre all'utente finale una visione integrata di queste, eliminando incongruenze e ridondanze inevitabilmente introdotte a causa della eterogeneità delle sorgenti. A questo proposito è stato introdotto il concetto di *mediatore*, cioè un modulo in grado di reperire e integrare informazioni ottenute da una molteplicità di sorgenti eterogenee. Ovviamente, le inevitabili differenze strutturali esistenti tra le diverse fonti portano alla nascita di varie tipologie di conflitti, quali per esempio differenze nell'utilizzo dei nomi, o incongruenze strutturali, generati dall'utilizzo di modelli differenti per rappresentare le stesse informazioni.

I sistemi che nel tempo sono stati presentati per risolvere questo genere di problemi si possono suddividere a seconda dell'approccio utilizzato, dividendoli quindi in *semantici* e *sintattici*. I sistemi *sintattici* sono caratterizzati dai seguenti punti:

- ✓ viene utilizzato un modello *self-describing* per trattare tutti gli elementi presenti nel sistema, ignorando gli schemi concettuali delle diverse sorgenti;
- ✓ le informazioni semantiche sono inserite manualmente tramite l'utilizzo di regole;
- ✓ l'integrazione di dati semi-strutturati è facilitata dall'utilizzo di un linguaggio *selfdescribing*;
- ✓ è impossibile ottimizzare semanticamente le interrogazioni (soprattutto nei database di grande dimensione) a causa dell'assenza di schemi concettuali;
- ✓ sono eseguibili solamente query predefinite.

Invece, i sistemi *semantici*, tra i quali si colloca anche MOMIS, hanno le seguenti caratteristiche:

- ✓ ogni sorgente viene codificata nello schema concettuale attraverso l'uso di metadati;
- ✓ le informazioni semantiche sono presenti nello schema concettuale, vengono tradotte in una logica descrittiva, e risultano utili sia nella fase di integrazione delle sorgenti, sia nell'ottimizzazione delle interrogazioni;
- ✓ è necessario che nel sistema sia presente un modello di dati comune, col quale descrivere in maniera uniforme le informazioni condivise;
- ✓ viene realizzata in modo automatico o manuale l'unificazione (parziale o totale) degli schemi concettuali, per giungere alla definizione di uno schema globale.

Come detto, MOMIS adotta un approccio di integrazione delle sorgenti *semantico* e *virtuale*: quest'ultimo termine indica che la vista integrata delle sorgenti (lo schema globale, o GVV) non viene di fatto materializzata, ma creata virtualmente attraverso la decomposizione della query e l'individuazione delle sorgenti da interrogare; inoltre, lo schema globale deve disporre di tutte le informazioni necessarie alla fusione dei risultati ottenuti sulle singole sorgenti, al fine di ottenere una risposta significativa. Le motivazioni che hanno portato all'adozione di un approccio di questo tipo sono le seguenti:

- ✓ avendo a disposizione uno schema globale, all'utente è permesso formulare qualsiasi interrogazione che risulti essere consistente con lo stesso, senza

conoscere gli schemi delle sorgenti dati; inoltre le informazioni semantiche in esso incluse possono permettere di effettuare una eventuale ottimizzazione di queste interrogazioni;

- ✓ l'utilizzo di una semantica *type as a set* per gli schemi consente di controllarne la consistenza, facendo riferimento alle loro descrizioni;
- ✓ la vista virtuale rende il sistema estremamente flessibile riguardo eventuali cambiamenti sia nel numero che nel tipo delle sorgenti.

MOMIS [7] 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 [3]. Allo scopo di presentare ed integrare le informazioni estratte ed integrate in un linguaggio comune viene introdotto un linguaggio object-oriented, chiamato 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:

- ✓ 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. Quindi ODL*I3*, in accordo con gli standard ODMG e I3, riprende le specifiche del linguaggio ODL aggiungendo estensioni utili per l'integrazione di informazioni e per la modellazione di dati semistrutturati:

- **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.

MOMIS è stato realizzato seguendo l'architettura di riferimento I3 [2], illustrata in Figura 28, che 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***: 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.

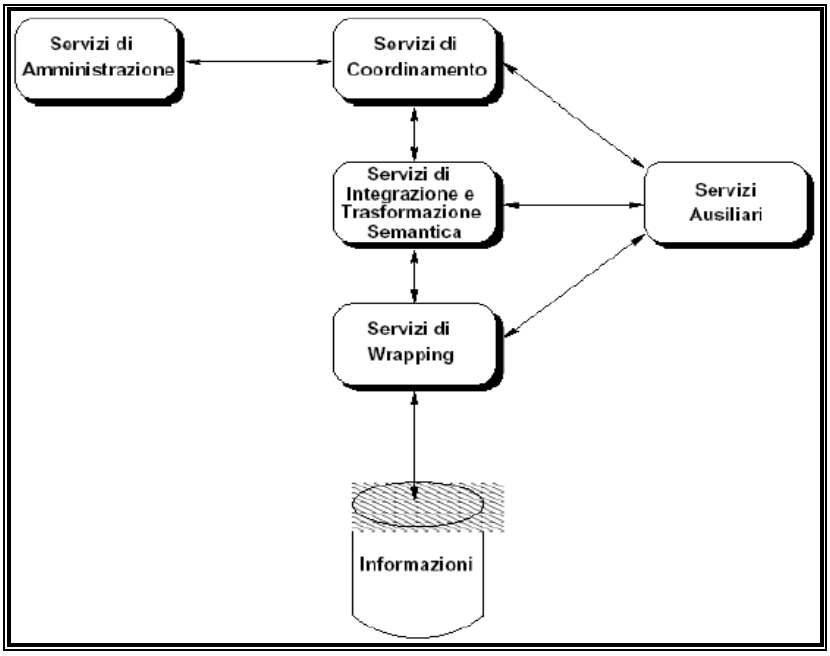


Figura 28. Architettura di riferimento I3

Durante la fase di integrazione si possono distinguere quattro livelli (Figura 29):

1. ***Sorgenti***: sono al livello più basso e rappresentano le fonti di informazioni che devono essere integrate. I tipi di sorgenti alle quali MOMIS 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. In MOMIS, i wrapper sono oggetti CORBA ed hanno duplice funzione:
    - 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*I3*) 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;

Attualmente i wrapper implementati in MOMIS sono:

- ✓ Wrapper JDBC generico;
- ✓ Wrapper per SQLServer;
- ✓ Wrapper XML;
- ✓ Wrapper JDBC-ODBC-MS_Access;
- ✓ Wrapper per Microsoft SQL Server;
- ✓ Wrapper XSD;
- ✓ Wrapper MySQL;
- ✓ WrapperOracle;
- ✓ WrapperOWL.

3. ***Mediatore***: può essere definito il cuore del sistema poichè combina, integra e ridefinisce gli schemi ricevuti dai wrapper tramite un'interfaccia ODL*I3*. È costituito da diversi sottomoduli:
   - ***Global Schema Builder***: è il modulo di integrazione degli schemi locali. Partendo dalle descrizioni delle sorgenti espresse in ODL*I3*, 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 ODB-Tools [21], 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 ODB-Tools;
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.

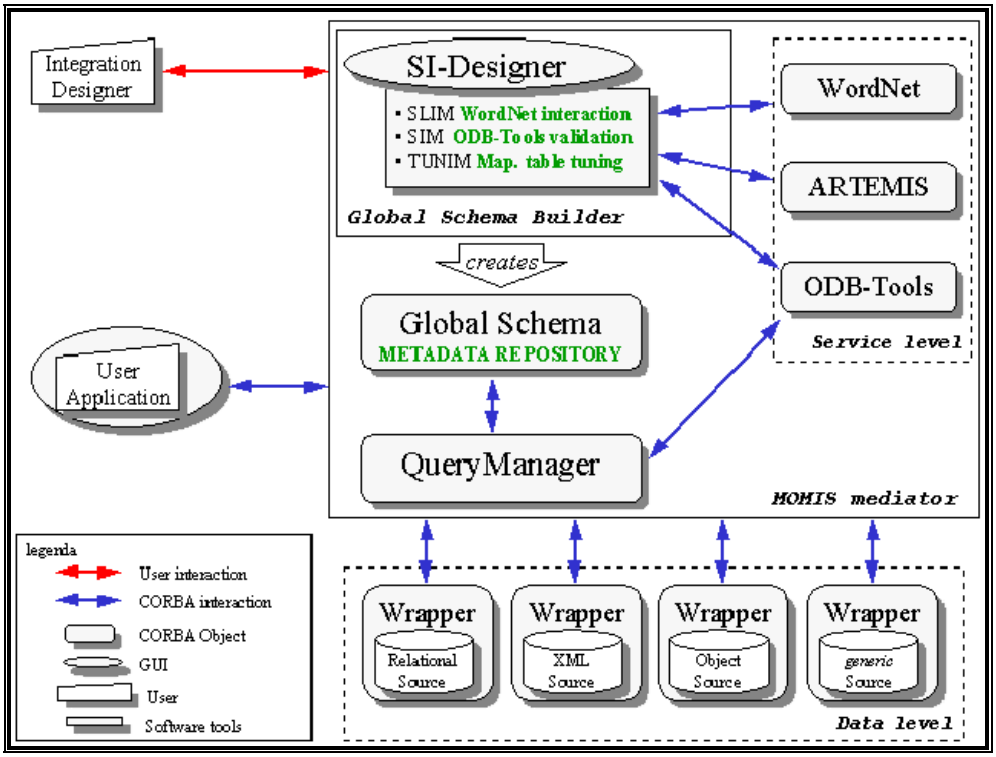


Figura 29. Architettura MOMIS

In Figura 29 si possono notare anche il tool SI-Designer (nel seguito sarà data una descrizione approfondita) e altri tools software:

- ✓ ***ODB-Tools*** 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 e' stato sviluppato presso l'Università di Modena e Reggio Emilia, sotto la direzione della Professoressa Sonia Bergamaschi;
- ✓ ***ARTEMIS*** [22] è un tool basato sulle tecniche di affinità e di clustering che esegue l'analisi semantica ed il clustering delle classi ODL*I3*. 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 è stato sviluppato presso l'Università di Milano e Brescia;
- ✓ ***WordNet*** [23] un database lessicale della lingua inglese, capace di individuare relazioni lessicali e semantiche tra termini.
- ✓ ***CORBA*** [24] non è un vero e proprio tool, ma il protocollo per lo scambio di oggetti tra i moduli adottato da MOMIS. Con CORBA (Common Object

Request Broker Architecture) 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) [25], un consorzio di oltre 800 aziende.

SI-Designer [26] 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. Inoltre effettua la "validazione semantica" delle relazioni e ne inferisce delle nuove sfruttando sempre ODB-Tools.
- ✓ **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 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. 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. In Figura 30 è illustrata la GUI di SI-Designer

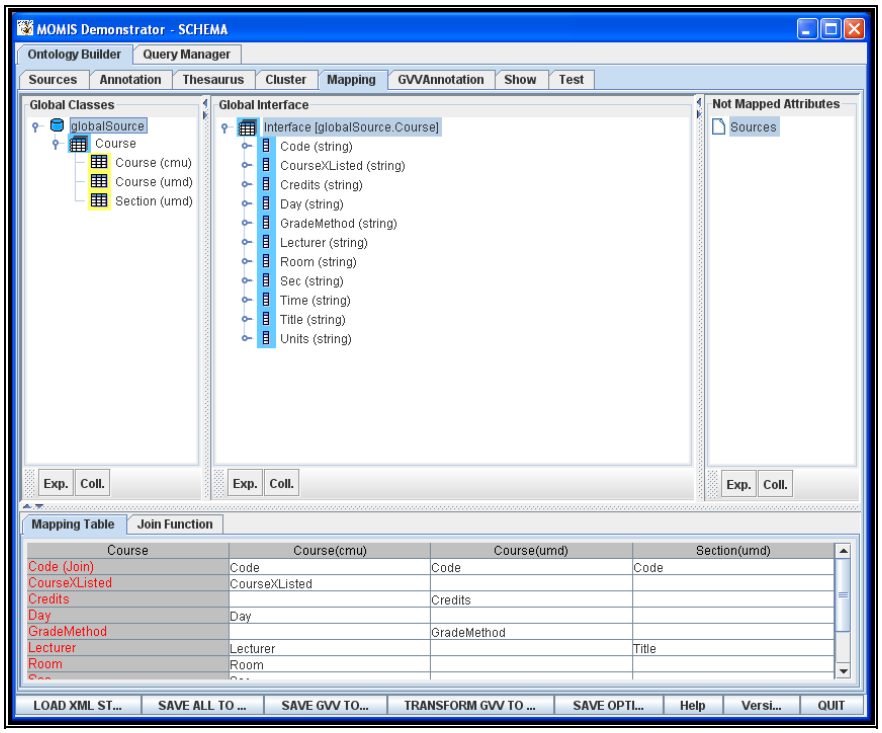


Figura 30. Interfaccia utente SI-Designer

3.1.1 Generazione dell'ontologia con il sistema MOMIS

Il processo di generazione dell'ontologia [3] può essere suddiviso in 4 fasi principali:

1. *Estrazione degli schemi locali*: I wrappers acquisiscono gli schemi locali dalle sorgenti e li traducono in schemi ODL*I3*. Le descrizioni degli schemi di dati strutturati (come ad esempio database relazionali, oggetti database) vengono direttamente tradotti, mentre l'estrazione di schemi da sorgenti dati semistrutturati richiede l'ausilio di ulteriori tecniche di elaborazione. Per eseguire l'estrazione di schemi da file XML, è stato sviluppato uno specifico wrapper che traduce automaticamente gli schemi XSD in strutture relazionali ed importa i dati nel DBMS. In questo lavoro di tesi, tutti gli schemi ed i dati forniti da THALIA sottoforma di file XML (10 schemi e 701 record) sono stati tradotti tramite questo wrapper.

2. *Annotazione manuale delle sorgenti*: in questa fase ogni elemento presente negli schemi locali (nome di tabella, nome di attributo, etc.) viene *annotato* rispetto all'ontologia lessicale *Wordnet*: ciò consiste nell'associare un significato, presente in Wordnet, al nome di ciascun elemento dello schema locale, permettendo di condividere lo stesso significato tra più elementi con nome diverso. Questo permette di sfruttare la conoscenza semantica per ottenere un'integrazione fra gli schemi il piu` possibile automatica. Per passare a questa fase basta selezionare "*Annotation*" nella parte alta dell'interfaccia SI-DESIGNER (vedi Figura 31). La fase di annotazione consiste sostanzialmente in due operazioni che devono essere ripetute per tutte le classi locali e per tutti gli attributi locali:
    - *Scelta della forma base*, il termine di lingua inglese che meglio descrive il concetto rappresentato dalla classe o dall'attributo in esame;
    - *Scelta del senso*, dopo aver selezionato la forma base occorre selezionare il senso relativo al contesto di riferimento.

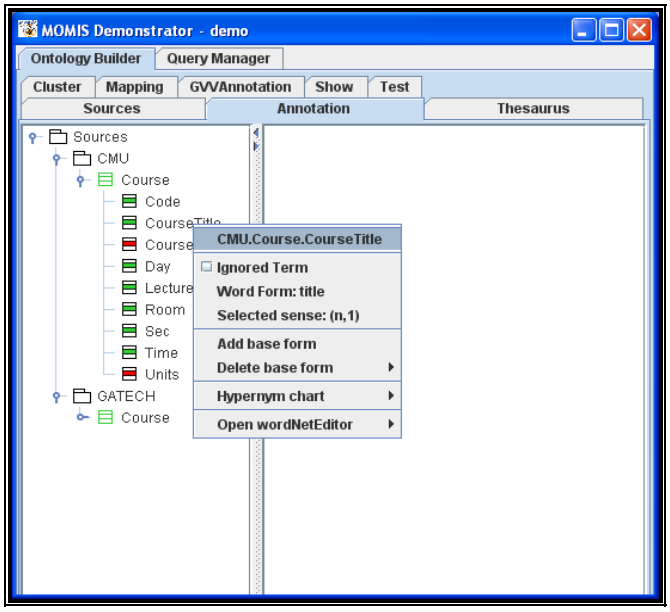


Figura 31. Interfaccia di annotazione

L'interfaccia all'inizio presenta l'albero delle sorgenti sulla sinistra. L'utente deve navigare l'albero e operare sui termini per eseguire l'annotazione; non è necessario annotare tutti i termini, ma aumentando il numero di termini annotati aumenta la probabilità di estrarre relazioni lessicali. MOMIS cerca in modo automatico di selezionare una forma base per ogni termine da annotare; questa

operazione avrà maggior successo se i termini delle sorgenti sono espressi in lingua inglese e senza caratteri particolari, nomi composti, etc. L'algoritmo di annotazione automatica implementato seleziona per un lemma il primo senso indicato da WordNet, il più utilizzato secondo il DB lessicale. La frequenza di utilizzo di un senso viene attribuita in base all'utilizzo del senso nei vari testi di concordanza semantica usati dal gruppo di ricerca di Princeton per la costruzione del DB lessicale. Durante la fase di annotazione può essere utile inserire nuovi concetti all'interno del database lessicale di riferimento WordNet. L'applicazione WordNetEditor, che è possibile eseguire tramite l'SI-Designer, fornisce un'interfaccia grafica che rende possibile l'estensione del database lessicale tramite l'aggiunta di nuovi concetti e nuovi significati per essi.

3. *Generazione del Common Thesaurus*: un insieme di relazioni terminologiche che descrivono la conoscenza riguardante le relazioni tra gli attributi e le classi delle sorgenti; utilizzando le tecniche di inferenza di OLCD vengono estratte relazioni intensionali intra-schema ed inserite nel Common Thesaurus. Successivamente si aggiungono le relazioni inter-schema ottenute:
   - Utilizzando il sistema lessicale WordNet, che identifica le affinità inter-schema sulla base di lessico/significato delle loro denominazioni;
   - utilizzando il sistema ARTEMIS che calcola le affinità strutturali di concetti inter-schema.

   La conoscenza viene espressa tramite relazioni terminologiche di tre tipi:
   - ✓ SYN (relazioni di *sinonimia*), definita tra 2 termini che possono essere scambiati senza modificare il concetto rappresentato:
   - ✓ BT (relazioni di specializzazione): definite tra due termini, di cui uno può avere un significato più generale dell'altro;
   - ✓ RT(relazioni di aggregazione): definita tra due termini tra i quali esiste un legame generico.

   Tramite l'interfaccia grafica "Thesaurus" del SI-Designer è possibile impostare e/o modificare queste relazioni.

4. *Generazione dello Schema Globale o Vista Virtuale Globale* (*GVV*): una volta costruito il Common Thesaurus, SI-Designer può generare le classi globali. Tale operazione viene attuata dal tool in questo modo:

- *Calcolo affinità e generazione dei cluster*: in questa fase SI-Designer funge da interfaccia tra il modulo ARTEMIS ed il progettista e quest'ultimo può interagire con il programma, fin quando non ottiene il numero di cluster desiderato. Per la generalizzazione dei cluster, ARTEMIS utilizza tecniche di clustering attraverso le quali le classi sono automaticamente classificate con una struttura ad albero dove: le foglie rappresentano tutte le classi locali (foglie contigue sono classi caratterizzate da alta affinità, mentre foglie tra loro molto lontane rappresenteranno invece classi a bassa affinità) e ogni nodo rapresenta un livello di clusterizzazione ed ha associato il coefficiente di affinità tra i due sottoalberi (cluster) che unisce. Con SI-Designer, si può immettere ad ogni iterazione un valore di solgia: ogni cluster sarà costituito da tutte le classi appartenenti ad un sottoalbero che al nodo radice ha un coefficiente maggiore del valore di soglia.
- *Generazione degli attributi globali e della mapping-table*: in questa fase, per ogni cluster, SI-Designer crea un insieme di attributi globali e, per ognuno di esso, determina la corrispondenza con gli attributi locali (quelli delle classi appartenenti al clastur cui corrisponde la classe globale). In alcuni casi, la corrispondenza è unica mentre in altri ci sono diversi tipi di corrispondenza che il tool individua ma di cui non può risolvere l'ambiguità: in questo caso il tool chiede al progettista di inserire quella giusta. Quindi ad ogni classe globale costruita viene associato un insieme di attributi globali e la relativa mapping table, mettendo a disposizione del progettista, tramite l'interfaccia grafica "Mapping" del SI-Designer, varie opzioni per revisionare gli insiemi di attributi globali e la mapping table proposti dal tool. Inoltre è possibile modificare il nome degli attributi globali e delle classi globali. La mapping table è una tabella *MT[AG][CL]* dove *CL* è l'insieme delle classi locali che appartengono al cluster cui la mapping table si riferisce, ed *AG* è l'insieme degli attributi globali creato da SI-Designer. In Figura 32 è rappresentata la Mapping Table generata per la risoluzione della query 1 del benchmark, in particolare tra le sorgenti CMU e GATECH.

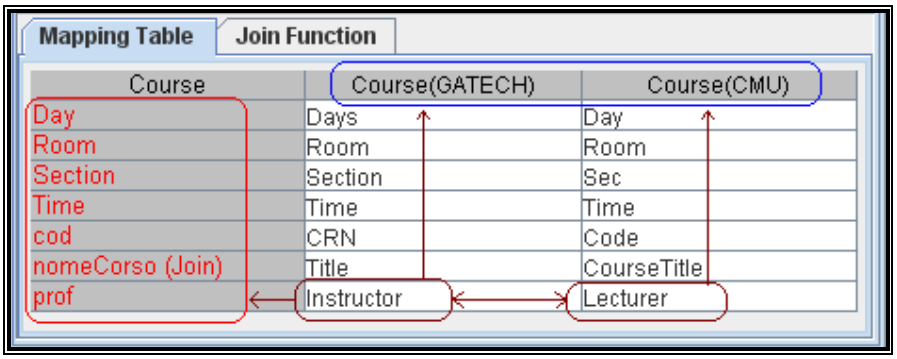


Figura 32. Mapping Table

Una volta creato una schema globale, e di conseguenza la relativa mapping table, l'SI-Designer offre la possibilità di intervenire su questa producendo un ulteriore raffinamento della mapping table creata. In particolare vi è la possibilità di utilizzare:

- *Funzioni di trasformazione dati*: per ogni elemento della mapping table *MT[AG][L]* si può definire una funzione di trasformazione, denotata con *MTF[AG][L]*, che rappresenta il mapping dell' attributo locale di *L* nell'attributo globale *AG*. Naturalmente *MTF[AG][L]* è una funzione che deve essere eseguita, e quindi supportata, direttamente sulla sorgente ospitante i dati di origine. Il sistema MOMIS accetta l'inserimento di funzioni *like SQL-92*:
  - `CHAR_LENGHT`, ritorna la lunghezza di una stringa;
  - `POSITION`, cerca la posizione di determinati caratteri in una stringa;
  - `SUBSTRING`, ritorna una parte di stringa;
  - `CASE WHEN` *`<condizione>`* `THEN` *`<espressione>`*, applica trasformazioni ad un record sulla base di determinati valori espressi nelle condizioni;
  - `RIGHT` and `LEFT`, ritornano rispettivamente i primi (o gli ultimi) N caratteri di una stringa.

  Queste funzioni sono tradotte, a livello wrapper, nel particolare "dialetto SQL" del DBMS che ospita i dati sorgente. MOMIS pemette l'inserimento di queste funzioni tramite l'interfaccia grafica "Mapping" del SI-Designer, tramite la mapping table. E proprio su questa che è possibile specificare le funzioni di trasformazione direttamente sull'attributo locale di una particolare sorgente o sull'attributo globale della mapping table. Ad esempio, per la query 4 del becnhmark è stato necessario creare una complessa funzione di trasformazione per convertire i crediti espressi in formato stringa con un particolare formato

(crediti=#U+#V+1), per i corsi della sorgente ETHZ, in formato di numero intero; quindi sull'attributo locale "Umfang" della mapping table generata per la risoluzione della query, è stata definita la seguente funzione di trasformazione:

```
MTF[Unit][ethz.Unterricht] =

CAST(SUBSTRING(Umfang, POSITION('V' IN Umfang) - 1, 1)
AS int)
+ CAST(SUBSTRING(Umfang, POSITION('U' IN Umfang) - 1,
1) AS int) + 1
```

- *Condizioni di JOIN*: la fusione di dati di diverse sorgenti richiede un istanziamento differente di stessi oggetti reali, ossia l'identificazione degli stessi. Per questo il SI-Designer offre la possibilità di specificare condizioni di JOIN su coppie di classi locali derivanti dalla stessa classe globale, proprio per permettere l'identificazione di istanze diverse di stessi oggetti e la loro conseguente fusione. Date due classi locali, L1 e L2, derivanti dalla classe globale *C*, una condizione di JOIN tra L1 e L2, denotata con JC(L1,L2), è un'espressione booleana di vincoli atomici (L1.Ai Op L2.Aj) dove Ai (Aj) sono attributi globali che non hanno un mapping nulllo in L1 (L2) ed Op è un operatore relazionale. Per esempio, sempre per la query 4, è stata definita la seguente condizione di JOIN:

  ```
  JC(L1, L2) : L1.CourseName = L2. Titel
  where L1= cmu.Course and
  L2= ethz.Unterricht
  ```

- *Funzioni di Risoluzione*: la fusione di dati provenienti da diverse sorgenti implica anche un problema di inconsistenza delle informazioni, derivanti dalle sorgenti dati stesse. In MOMIS, per risolvere i conflitti tra dati, l'approccio proposto è quello di utilizzare funzioni di risoluzione definite per ogni attributo globale che possiede un mapping con attributi locali provenienti da più sorgenti locali. Mentre un attributo che non presenta un conflitto dati (ad esempio le istanze dello stesso oggetto reale di differenti classi locali, che possiedono lo stesso valore per un determinato attributo) è chiamato attributo omogeneo; per questo tipo di attributi non c'è bisogno di una funzione di risoluzione.

- *Mapping query*: MOMIS segue un approccio GAV, quindi per ogni classe globale *C* deve essere definita una mapping query QC sugli schemi di un

insieme delle classi locali *L(C)*. In pratica, il sistema sfrutta la Mapping Table per generare automaticamente la QC associata alla classe globale *C*, estendo l'operatore di disgiunzione totale (FD) agli schemi delle classi locali *LC*. Ad esempio, data una classe globale *C* mappata con *L1*, *L2*, *L3* classi locali, si considera:

FD(T(L1),T (L2),...,T (Ln)), calcolato sulla base della condizione di JOIN.
Prendendo in esame due sole classi locali, FD corrisponde al FULL (OUTER) JOIN:
FD(T (L1),T (L2)) = T (L1) full join T (L2) on (JC(L1,L2)).

Infine, QC è ottenuta tramite l'applicazione delle funzioni di risoluzione all'insieme degli attributi risultanti dopo aver applicato la FD.

3.1.2 Esecuzioni delle query in MOMIS

Le query globali sono espresse sullo schema globale, costruito con il processo descritto in precedenza, ed è possibile formularle direttamente tramite il modulo messo a disposizione da MOMIS, il Query Manager (vedi Figura 33): tramite quest'interfaccia è possibile definire la query ed eseguirla sullo schema globale necessariamente creato in precedenza; inoltre i risultati dell'interrogazione vengono presenti sempre nella stessa finestra, sottoforma di tabella, derivante dall'integrazione degli schemi delle sorgenti integrate.

Naturalmente le query globali devono essere riscritte in un equivalente set di query locali espresse, appunto, ognuna su uno schema locale; la traduzione della query globale nelle specifiche query locali è eseguita considerando il mapping tra lo schema globale (GVV) e gli schemi locali. Con un approccio GAV la traduzione della query è eseguita tramite un processo detto query *unfolding*, espandendo la query globale sulla classe globale appartenente al GVV ed eseguendo il mapping query, come descritto in precedenza.

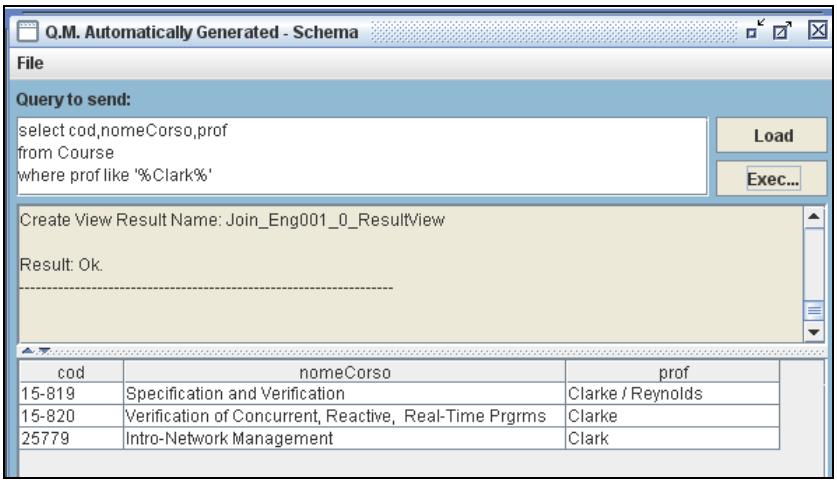


Figura 33. Query Manager

Definendo una query globale *Q* come:

```
Q = SELECT <Q_SELECT-list> from C where <Q_condition>
```

Il processo di *unfolding* è composto dai seguenti tre step:

1. Generazione query locale *LQ:*

   ```
   LQ = SELECT <SELECT-list>
   FROM L
   WHERE <condition>
   ```

   dove *L* è una classe locale relativa a *C.*

   Mentre `<SELECT-list>` è calcolato considerando l'unione di:

   - gli attributi globali in `<Q_SELECT-list>` con un mapping non nullo in *L*;
   - gli attributi globali usati per esprimere la condizione di JOIN per *L*;
   - gli attributi globali in `<Q_condition>` con un mapping non nullo in *L*.

   Il set degli attributi globali è trasformato nel corrispondente set di attributi locali tramite la Mapping Table. Invece, la clausola `<condition>` è calcolata eseguendo un mapping sui vinvoli atomici: ogni vincolo della `<condition>` viene riscritto in maniera tale da essere compatibile nella sorgente locale. Il mapping dei vincoli atomici viene eseguito sulla base di funzioni di conversione dati e funzioni di risoluzione, definite nella Mapping Table.
2. Il secondo step coincide con la generazione dell FD(*LQ1*, *LQ2*, … , *LQn*) calcolando la disgiunzione delle query locali (associate alla query globale).

3. Generazione della query finale (applicando le funzioni di risoluzione):
    - per gli attributi omogenei verrà preso solo un valore di questi;
    - per gli attributi non omogenei sono applicate le funzioni di risoluzione.

In uno scenario di integrazione delle informazioni spesso i dati sono espressi in diversi linguaggi; per questo, MOMIS prevede la possibilità di proporre una *<Q_condition>* in uno specifico linguaggio, per esempio in inglese, ed il processo di riscrittura della query cercherà di tradurre ogni query locale nel linguaggio specificato nella condizione di query. Questo operazione di traduzione viene eseguita da una funzione TRANSLATION, che traduce le parole da un linguaggio in un altro sfruttando il dizionario *open_source* GUTENBERG [27] determinando, per ogni parola, la traduzione nel linguaggio della sorgente dati specifica. Più precisamente, dato un attributo globale *GA*, un vincolo di linguaggio viene espresso nel modo seguente:

GA op TRANSLATE(*term*, *Language*)

Per esempio, nella query 5, del benchmark THALIA, è stato necessario utilizzare questa funzione di traduzione, in quanto i dati della sorgente ETHZ sono espressi in lingua tedesca, mentre quelli della sorgente UMD sono espressi in lingua inglese. La query globale applicata è la seguente:

```
SELECT Name FROM Course
WHERE Name LIKE TRANSLATE('%Database%', 'en')
```

Per lo schema della sorgente UMD non è necessaria nessuna traduzione, quindi la scrittura della query locale è la seguente:

```
SELECT CourseName FROM Course
WHERE CourseName LIKE '%Database%'
```

Mentre per la sorgente dati ETHZ, la riscrittura della query diventa:

```
SELECT Titel FROM Unterricht
WHERE Titel LIKE '%Datenbank%'
OR Titel like '%Datei%'
OR Titel like '%Datenbasis%'
```

3.2 Configurazione e connessione alle sorgenti dati

La prima fase dell'integrazione delle sorgenti è l'acquisizione delle stesse tramite la configurazione del wrapper utilizzato per connettersi ad una particolare tipologia di sorgente dati. In questo lavoro di tesi, i dati delle sogenti sono memorizzati all'interno del DBMS Microsoft SQL Server 2000, quindi come wrapper, di conseguenza, è stato utilizzato *SQLServer_Wrapper* sviluppato ad-hoc er questa particolare tecnologia. Il SI-Designer mette a disposizione l'interfaccia "Sources" (vedi Figura 34) per la gestione della connessione alle sorgenti; dopo aver scelto la tipologia di wrapper e configurato in maniera corretta i parametri di connessione, il sistema genera in maniera automatizzata la stringa JDBC per la connessione al database. I parametri da inserire sono i seguenti:

- `<domani:port>`, il dominio e la porta del server ospitante il DBMS;
- `<username>`, l'username per autenticarsi sul database al quale ci si deve connettere;
- `<password>`, password relativa all'username inserito in precedenza.

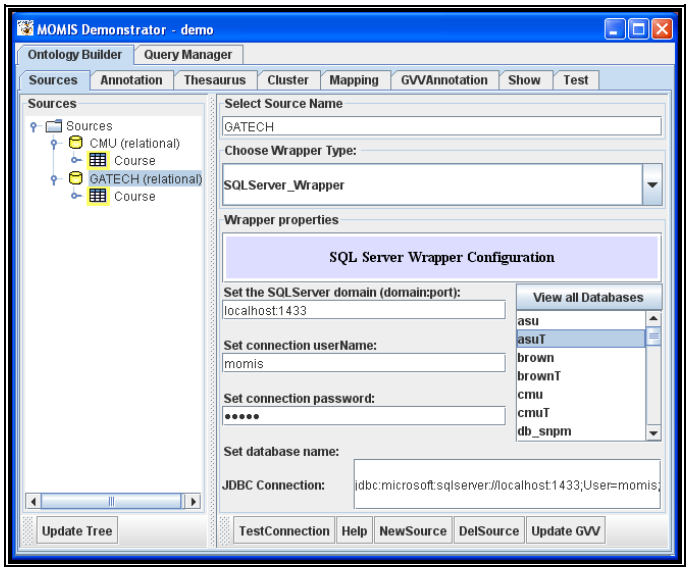


Figura 34. Interfaccia per connessione alle sorgenti

Ad esempio per la connessione al database della sorgente GATECH, si è generata la seguente stringa di connessione:

```
jdbc:microsoft:sqlserver://localhost:1433;User=XXX;Passw
ord=XXX;DatabaseName=gatech
```

Per ogni sorgente dati, utilizzata nel progetto d'integrazione, è necessario effettuare la connessione tramite il wrapper.

3.3 Implementazione delle query del benchmark THALIA in MOMIS

Per implementare le 12 query del benchmark è stato necessario creare, per ognuna di esse, uno schema globale, partendo dagli schemi locali delle due sorgenti coinvolte per ogni query. Durante la fase di raffinamento dei mapping, per ogni query, sono state sviluppate funzioni di trasformazione per risolvere le varie eterogeneità presenti nei dati da integrare. Di seguito per ogni query verranno indicati i mapping tra gli attributi locali di ogni Mapping Table costruita per l'implementazione di ogni query.

Query 1

- ✓ Mapping tra l'attributo "Instructor" della sorgente GATECH e l'attributo "Lecturer" di CMU.
- ✓ Nessun ulteriore raffinamento.
- ✓ Query eseguita:

```
Select Code, Title, Lecturer
from Course
where Lecturer like '%Clark%'
```

Query 2

- ✓ Mapping tra l'attributo "Time" della sorgente CMU e l'attributo "Times" di UMB.
- ✓ Raffinamento sull'attributo "Time" di UMB:

  MTF[Time][umb.Course] = `TIME12-24(Times, 1, 12) + SUBSTRING(Times, 6, 1) + TIME12-24(Times, 7, 12)`

  La funzione TIME12-24 è stata definita come:

```
CASE WHEN ISNUMERIC(SUBSTRING(Times, 1, 2)) = 1 THEN
   CASE WHEN CAST(SUBSTRING(Times, 1, 2) AS int) > 12
       THEN CAST(CAST(SUBSTRING(Times, 1, 2) AS integer)-
       12              AS nvarchar(2))
       ELSE SUBSTRING(Times, 1, 2)
   END + SUBSTRING(Times, 3, 4) +
     CASE WHEN CAST(SUBSTRING(Times, 7, 2) AS int) > 12
THEN        CAST(CAST(SUBSTRING(Times, 7, 2) AS integer)-
                          12
     AS nvarchar(3)) ELSE  SUBSTRING(Times, 7, 2)
   END + SUBSTRING(Times, 9, 3)
END
```

- ✓ Query eseguita:

```
Select Title, Time
from Course
where Time LIKE '1:30%'
```

Query 3

- ✓ Mapping tra l'attributo "CourseName" della sorgente UMD e l'attributo "Title" di BROWN.
- ✓ Raffinamento su "Title" di BROWN:

  MTF[Title][brown.Course]=SUBSTRING(Title,CHARINDEX('/\"', Title)+3,CHARINDEX('hr.',SUBSTRING(Title,CHARINDEX('/\"', Title) + 3, 1000)) - 1)
- ✓ Query eseguita:

```
Select Code, CourseName
from Course
where CourseName LIKE '%Data Structures%'
```

Query 4

- ✓ Mapping tra l'attributo "Units" della sorgente CMU e l'attributo "Umfang" di ETHZ.
- ✓ Raffinamento su "Umfang" di ETHZ, ed utilizzo della funzione **TRANSLATE** per tradurre il termine *database* in lingua tedesca, in fase di query sulla sorgente ETHZ:

  MTF[Unit][ethz.Unterricht] = CAST(SUBSTRING(Umfang, CHARINDEX('V', Umfang) - 1, 1) AS int) + CAST(SUBSTRING(Umfang, CHARINDEX('U', Umfang) - 1, 1) AS int) + 1
- ✓ Query eseguita:

```
Select Title, Units
from Course
where Title LIKE TRANSLATE('%Database%', 'en')
```

Query 5

- ✓ Mapping tra l'attributo "CourseName" della sorgente UMD e l'attributo "Title" di ETHZ.
- ✓ Utilizzo della funzione TRANSLATE in fase di query.
- ✓ Query eseguita:

```
Select CourseName
from Course
where Title LIKE TRANSLATE('%Database%', 'en')
```

Query 6

- ✓ Mapping tra l'attributo "text" della sorgente TORONTO e nessun attributo di CMU.
- ✓ Query eseguita:

```
Select Title, Text
from Course
where Title LIKE '%verification%'
```

Query 7

- ✓ Mapping tra l'attributo "prerequisite" della sorgente UMICH e l'attributo "description" di ASU.
- ✓ Raffinamento su "prerequisite" di ASU:

  MTF[prerequisite][asu.Course] = CASE PATINDEX('%Prerequisite%', Description) WHEN 0 THEN 'None' ELSE RIGHT(Description, LEN(Description) - PATINDEX('%Prerequisite%', Description) + 1) END
- ✓ Query eseguita:

```
Select name, description, prerequisite
from Course
where prerequisite like '%none%'
```

Query 8

- ✓ Mapping tra l'attributo "Description" della sorgente GATECH e nessun attributo di ETHZ.

- ✓ Nessun ulteriore raffinamento.
- ✓ Query eseguita:

```
Select DISTINCT Title, Description
from Course
where Description LIKE '%JR%'
```

Query 9

- ✓ Mapping tra l'attributo "room" della sorgente BROWN e l'attributo "time" di UMD.
- ✓ Raffinamento su "Time" di UMD:

  MTF[Room][umd.section] = `SUBSTRING([Time], PATINDEX('%(%', [Time]), 30)`
- ✓ Inoltre è stata definita la condizione di JOIN, nella Mapping Table, sul campo "Code" tra le due tabelle, Course e Section, nella sorgente UMD, per recuperare l'informazione del campo "Time" della tabella Section.
- ✓ Query eseguita:

```
select Code, Title, Room
from Course
where Title LIKE '%Software Engineering%'
```

Query 10

- ✓ Mapping tra l'attributo "lecturer" della sorgente CMU e l'attributo "title" di UMD.
- ✓ Raffinamento su "title" di UMD:

  MTF[Title][umd.section] = `SUBSTRING(SUBSTRING(Title, 1, PATINDEX('%.%',Title)), PATINDEX('%)%',Title) + 2, PATINDEX('%.%',Title) + 1)`
- ✓ Inoltre è stata definita la condizione di JOIN, nella Mapping Table, sul campo "Code" tra le due tabelle, Course e Section, nella sorgente UMD, per recuperare l'informazione del campo "title" della tabella Section.

- ✓ Query eseguita:

```
Select Code, Title, Lecturer
from Course
where Title LIKE '%computer%'
```

Query 11

- ✓ Mapping tra l'attributo "lecturer" della sorgente CMU e gli attributi "Fall2003", "Winter2004" e "Spring2004" di UCSD.
- ✓ Raffinamento su "Fall2003", "Winter2004" e "Spring2004" di UCSD:

  MTF[Lecturer][ucsd.Course] = `CASE WHEN (LEN(Fall2003) > LEN(Winter2004) AND LEN(Fall2003) > LEN(Spring2004)) THEN Fall2003 WHEN (LEN(Winter2004) > LEN(Fall2003) AND LEN(Winter2004) > LEN(Spring2004)) THEN Winter2004 WHEN (LEN(Spring2004) > LEN(Fall2003) AND LEN(Spring2004) > LEN(Winter2004)) THEN Spring2004 END`
- ✓ Query eseguita:

```
select Title, Lecturer
from Course
where Title LIKE '%Database%'
```

Query 12

- ✓ Mapping tra gli attributi "CourseTitle", "Day", "Time" della sorgente CMU e l'attributo "Title" di BROWN.
- ✓ Raffinamento sull'attributo "Title" di BROWN:

  MTF[Title][brown.Course] = `SUBSTRING(Title, CHARINDEX('/\"', Title) + 3, CHARINDEX('hr.', SUBSTRING(Title, CHARINDEX('/\"', Title) + 3, 1000)) - 1)`

  MTF[Day][brown.Course] = `SUBSTRING(Title, CHARINDEX('hr.', Title) + 4, CHARINDEX(' ', SUBSTRING(Title, CHARINDEX('hr.', Title) + 4, 10)))`

  MTF[Time][brown.Course] = `SUBSTRING(Title, CHARINDEX(' ', SUBSTRING(Title, CHARINDEX('hr.', Title) + 4, 10)) + CHARINDEX('hr.', Title) + 4, 15)`

- ✓ Query eseguita:

```
Select Title , Day, Time
from Course
where Title LIKE '%Computer%Networks%'
```

Capitolo 4

Analisi comparativa dei sistemi di integrazione

In questo capitolo verrà sviluppato il confronto tra i sistemi di integrazione dati utilizzati in questo lavoro di tesi, analizzandone le caratteristiche principali: costo delle licenze, facilità di installazione, limiti e vantaggi dei diversi sistemi. Inoltre verrà proposta un'analisi comparativa sulla modalità di integrazione dei dati implementata da ciascun sistema di integrazione. Infine, verranno illustrati i risultati ottenuti dall'applicazione del benchmark THALIA ai vari sistemi di integrazione, indicando, per ogni sistema, quante e quali query è stato possibile risolvere e con quale facilità/complessità.

È fondamentale fare una premessa: i sistemi di integrazione testati, possono essere suddivisi in due grandi categorie: MOMIS e l'IICE di IBM hanno un approccio virtuale, cioè non materializzano il database dei dati integrati, ma creano, anche se in maniera differente, delle viste virtuali dei dati di origine; mentre i sistemi Microsoft SSIS e Oracle DI seguono rispettivamente le logiche di integrazione ETL ed E-LT, non creano perciò una vista globale di tutti i dati delle sorgenti d'origine, ma materializzano il database dei dati integrati.
Nella presente discussione tutti i sistemi saranno, comunque, citati come sistemi di integrazione dati.

4.1 Caratteristiche d'installazione

In questo paragrafo verranno illustrate le particolarità riscontrate durante il processo d'installazione dei sistemi di integrazione. In linea generale, non ci sono stati problemi per i sistemi di integrazione Microsoft, Oracle e MOMIS: questi, seppur in maniera differente, hanno dimostrato una certa facilità d'installazione. Invece, per il

sistema di integrazione dell'IBM, sono state riscontrate alcune difficoltà di installazione, dovute principalmente ad un'incompatibilità tra il server delle applicazioni (WAS 6.1) e la versione dell'IICE (release 8.3) utilizzata nella prima parte del lavoro di tesi. Problemi che sono stati risolti, poi, grazie al rilascio della versione 8.4 dell'IICE, da parte di IBM, contemporaneamente allo svolgimento del lavoro di tesi.

Relativamente al software di integrazione dati IBM, in primo luogo è stato installato il pacchetto WebSphere Application Server 6.1 (ultima release), come server delle applicazioni su cui distribuire l'IICE 8.3. Successivamente è stato installato l'IICE, sempre sulla stessa macchina. La procedura è risultata essere abbastanza pesante, sia in termini di tempo che di utilizzo di risorse, ma comunque è andata a buon fine. Purtroppo, dopo aver configurato il primo connettore RDBMS, e settato i parametri per la connessione alla prima sorgente dati, quando si è provato a testarne la connessione si è verificato il primo problema: un errore relativo alla classe Java che cercava di instanziare la connessione, in particolare un "*ClassNotFoundException*". Dopo vari e invani tentativi di provare a risolvere il problema, è stato chiesto supporto ai tecnici IBM, sul forum riguardante il software IICE; il problema è stato risolto aggiungendo le seguenti righe di codice nel file di configurazione "*config.bat*" dell'IICE8.3, vedi discussione su forum [28] di IBM:

```
set
EJB_CLIENT_CLASSPATH=%EJB_CLIENT_CLASSPATH%;%WAS_HOME%\l
ib\wsexception.jar
rem - New for WAS 6.1
set
EJB_CLIENT_CLASSPATH=%EJB_CLIENT_CLASSPATH%;%WAS_HOME%\r
untimes\com.ibm.ws.admin.client_6.1.0.jar
```

È stato quindi possibile istanziare tutti i connettori relativi alle varie sorgenti dati, per testare il software con un progetto di integrazione, ma per eseguire le query federate è risultata necessaria l'installazione di un wrapper, fornito con il software stesso. Provando ad installare l'opportuno wrapper, si è riscontrato un altro problema: durante l'avvio della procedura di installazione, il processo non riusciva a reperire nel sistema determinate librerie. Problema abbastanza anomalo, al quale, dopo svariati tentativi, non è stata trovata soluzione; facendo nuovamente riferimento al forum del supporto IBM, è stato evidenziato da parte dei tecnici, un'incompatibilità tra la versione 8.3 dell'IICE e la versione 6.1 del WAS sul quale era stato distribuito l'IICE, vedi forum [29] di supporto. Contemporaneamente a queste prove del lavoro di tesi, è stata rilasciata sul

mercato, da parte di IBM, una nuova versione dell'IICE, la release 8.4 (18 dicembre 2007), quella che è stata effettivamente utilizzata, poi, in questo lavoro di tesi. Infatti installando nuovamente il WAS 6.1 e distribuendo su questo l'installazione dell'IICE8.4, sono stati risolti tutti i problemi riscontrati in precedenza, e si è potuto procedere con le varie prove.

Come si può notare l'installazione del software IBM non è stata per niente agevole ed è pesata sul lavoro di tesi soprattutto in termini di tempo impiegato per risolvere le incompatibilità riscontrate in fase di installazione e configurazione.

Per quanto riguarda Microsoft SSIS non c'è stato alcun problema d'installazione, infatti lanciando il file eseguibile di Microsoft SQL Server 2005, è stato possibile scegliere, come opzione aggiuntiva, l'installazione del pacchetto di Integration Services. Inoltre non è stata necessaria nessuna configurazione aggiuntiva, in quanto come DBMS d'appoggio il sistema di integrazione utilizza SQL Server 2005. Per il sistema Oracle Data Integrator, discorso analogo al precedente, con la piccola differenza che dopo aver installato il pacchetto software, è necessaria una piccola configurazione: infatti, come descritto nel paragrafo 2.5, bisogna configurare Master e Work repository per impostare i parametri di connessione al database che Data Integrator utilizza per importare i metadati nel processo di integrazione. Configurazione che è risultata essere molto semplice, e che non ha dato nessun tipo di problema; inoltre questa procedura è descritta molto bene nel manuale.

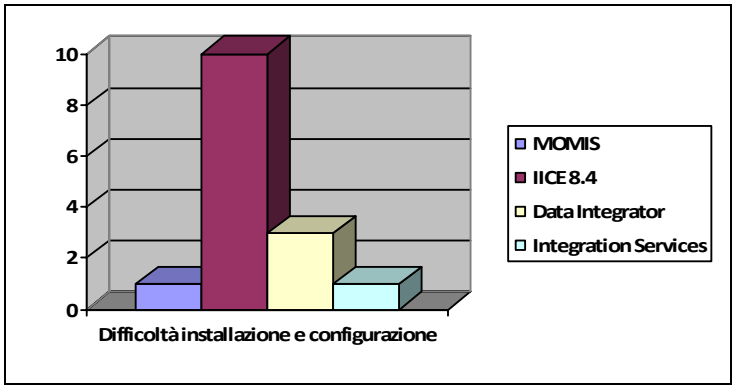
L'installazione di MOMIS è risultata essere altrettanto semplice, infatti attraverso la tecnologia Java Web Start è stato possibile rendere disponibile per tutti gli utenti in modo semplice ed intuitivo il client dell'Ontology Builder, SI-Designer. È possibile scaricare, installare ed eseguire l'applicazione "con un solo click"; non sono necessarie complesse procedure d'installazione o di configurazione: tutto avviene in modo del tutto trasparente all'utente. Anche l'aggiornamento a nuove versioni avviene in modo automatico e trasparente ad ogni esecuzione dell'applicazione. Gli unici requisiti richiesti all'utente sono la presenza di Java Web Start installato ed un collegamento Internet sempre attivo durante l'esecuzione di SI-Designer. Java Web Start è una nuova tecnologia di distribuzione per le applicazioni basate su Java che consente all'utente di lanciare e gestire le applicazioni direttamente dal Web: è possibile attivare le applicazioni in maniera semplice, avendo la certezza di utilizzare sempre la versione più aggiornata ed evitando le complesse procedure di installazione o aggiornamento. La modalità tipica di distribuzione del software attraverso il Web richiede che l'utente trovi

il programma di installazione in rete, lo scarichi, lo posizioni nella cartella desiderata e lo esegua. Una volta eseguita l'installazione, vengono richieste le directory e le opzioni di installazione, come, ad esempio, il tipo di installazione desiderata (completa, tipica o minima). Il risultato è una procedura complessa e che richiede tempo, e che deve essere ripetuta per ogni nuova versione del software. Al contrario, per le applicazioni distribuite sul Web, come i client di posta elettronica e i calendari realizzati in linguaggio HTML, i siti di aste e così via, l'installazione e l'utilizzo sono facili e immediati; il browser Web, infatti, consente di automatizzare l'intera procedura evitando di dover effettuare download, installazioni e configurazioni complesse e garantendo, nel contempo, l'utilizzo della versione più aggiornata. Java Web Start assicura alle applicazioni complete gli stessi vantaggi descritti per le applicazioni realizzate in linguaggio HTML. L'unico svantaggio consiste nella necessità di scaricare l'applicazione la prima volta che la si utilizza. Prima di installare SI-Designer è necessario aver installato una qualunque versione di Java Web Start.

Dall'indirizzo web "*http://www.dbgroup.unimo.it/MOMIS/MOMIS-jws/index_ita.html"* si può installare, se autorizzati (username e password), ed avviare l'SI-Designer di MOMIS. Per i successivi utilizzi dell'applicazione sono possibili diverse modalità:

- ✓ attraverso l'Application Manager di Java Web Start;
- ✓ attraverso l'icona di collegamento presente sul desktop e/o nel menù di avvio di Windows (la creazione dei collegamenti viene notificata da Java Web Start al primo o secondo utilizzo dell'applicazione);
- ✓ utilizzando nuovamente il link indicato in precedenza.

Di seguito è riportato un grafico con una scala simbolica del livello di difficoltà di installazione e configurazione riscontrato nei sistemi trattati.



**I valori delle ordinate sul grafico sono soggettivi, e derivanti dalle prove fatte durante il lavoro di tesi*

4.2 Analisi comparativa tra MOMIS ed i sistemi di integrazione dati analizzati

In questo paragrafo saranno riepilogate le caratteristiche principali dei vari sistemi di integrazione In particolare verrà effettuata un'analisi tecnica della modalità dei processi di integrazione dati, valutando relativi pregi e difetti di ciascun sistema. Inoltre, sarà affrontata un'analisi comparativa dei risultati del benchmark THALIA applicato ai sistemi di integrazione utilizzati in questo lavoro di tesi.

4.2.1 Oracle Data Integrator: caratteristiche, pregi e difetti

Oracle Data Integrator si pone come sistema di integrazione migliorativo rispetto ai sistemi data warehouse. Rispetto alla tradizionale architettura ETL (Extract, Trasfom and Load) dei sistemi di data warehouse, Oracle propone un architettura ELT in cui la trasformazione dei dati avviene sullo stesso server su cui i dati vengono caricati, non richiedendo più un server dedicato e potendo far uso delle operazioni SQL. Le regole per l'integrazione sono semplificate rispetto ai sistemi ETL, in quanto fanno uso di un design dichiarativo in cui è necessario specificare il compito del processo, quali dati deve integrare e quali trasformazioni e aggregazioni attuare per generare automaticamente il data flow risultante.
Il sistema si compone di quattro moduli grafici:

- Designer, il cuore dell'applicazione per gli sviluppatori e gli amministratori di metadati; utilizza i metadati e le regole definite per generare lo scenario di produzione.
- Operator, mostra i log di esecuzione, è pensato per il monitoraggio della produzione.
- Topology Manager, definisce l'architettura fisica e logica.
- Security Manager, gestisce i profili e i privilegi.

Il componente runtime, lo Scheduler Agent, coordina l'esecuzione dello scenario. I repository sono organizzati in:

- Master repository, è uno solo, contiene informazioni topologiche (tecnologie e server) e di sicurezza (privilegi, profili utente)

- Work repository, possono essere più di uno, tra essi, l'Execution Repository e' utilizzato per memorizzare informazioni di esecuzione (con obiettivo di produzione)

Oracle Data Integrator consente l'accesso al repository dei metadati da qualsiasi web browser. Il repository dei metadati può essere installato su diversi RDBMS: Oracle, Microsoft SQL Server, IBM DB2 UDB, IBM DB2/400, Informix, Sybase. Il sistema di Oracle permette di integrare sul database target le modifiche che avvengono sulle sorgenti, senza bisogno di tornare ad eseguire il processo di integrazione: un modulo dedicato segue gli eventi di modifica delle sorgenti e li esegue regolarmente in batch (pull mode) o in tempo reale (event-driven, push mode). L'integrazione è definita manualmente dall'utente attraverso l'uso di un interfaccia grafica, e non è disponibile alcun strumento che permette di definire delle regole e inferire i mapping in modo da assistere il progettista. Non vi è, quindi, automazione nel processo di mapping dei dati. C'è un ampio supporto per l'integrazione di sorgenti RDBMS, mentre altri tipi di dati sono più difficili da trattare (per esempio, l'integrazione di file XML è supportata, ma non ci sono moduli che mantengono l'aggiornamento dei dati in formato XML).
L'installazione del sistema risulta essere abbastanza semplice, è richiesta solo una piccola configurazione iniziale per il Master ed i Work repository.

4.2.2 Information Integrator Content Edition: caratteristiche, pregi e difetti

Il componente IBM Information Integrator consente di realizzare l'integrazione di sorgenti dati eterogenee di tipo strutturato e semistrutturato attraverso interfacce standard (ODBC, JDBC, etc.). Il processo di integrazione produce un database virtuale federato con gli oggetti remoti configurati come se fossero tabelle locali. Il database virtuale federato comunica con le sorgenti di dati attraverso moduli, uno per ogni tipo di sorgenti dati. L'IICE, fornisce inoltre la possibilità di integrare sorgenti dati multimediali come documenti, immagini, audio, video.

L'architettura è di tipo Service Oriented Architecture, che può essere descritta dai seguenti servizi principali:

- *integration services*: forniscono un'interfaccia comune ai repository sottostanti includendo funzionalità per la gestione dei contenuti e dei workflow;
- *federation services*: includono funzionalità di ricerca sul database integrato federato e funzionalità di mapping per la traduzione delle query rispetto agli schemi dei diversi repository;
- *developer and end user services*: includono un web client per il processo di integrazione, componenti web per la costruzione rapida di applicazioni web ed una API;
- *security services:* oltre alla gestione dell'autenticazione e dell'autorizzazione sulle sorgenti esistono un authentication system e un authorization system per operazioni sul sistema federato.

Il sistema consente di interrogare il database virtuale attraverso attraverso query federate, lanciate tramite uno script da riga di comando: tramite queste è possible definire criteri di selezione e criteri di ricerca sugli attributi globali definite nell'associazione dati costruita durante la fase di mapping, e passata anche questa come parametro nella formulazione della query. Le sorgenti di contenuti, identificate da un url, sono interrogabili tramite la funzioni getContentLookup() che restituisce informazioni relative ai documenti e la funzione getContent() che restituisce il contenuto dei documenti. E' possibile definire funzioni personalizzate (FULLTEXT, MAXRESULTS, ITEMCLASS, DATAMAP, CONTAINER) per ricerche full-text o definire constraint su sorgenti di dati o di contenuti. Le funzioni personalizzate possono quindi essere utilizzate all'interno di una query sul database virtuale integrato. Non sono previsti meccanismi per la risoluzione di conflitti sui dati. Inoltre non sono previste funzioni di trasformazioni sui dati; il sistema mette a disposizione una classe Interface Java, che, opportunamente programmata, può essere utilizzata dall'utente per implementare l'*unfolding* della query federata sulle diverse sorgenti dati target, ed inoltre manipolare il result-set finale della query per ulteriori raffinamenti sui dati.

I connettori del sistema WebSphere Information Integrator permettono di integrare un gran numero di sorgenti dati e di sistemi per la gestione dei contenuti. La gestione delle ricerche sul database integrato federato (query-brokering e aggregazione) risulta essere molto efficiente. WebSphere Information Integrator permette di materializzare il database virtuale sul sistema federato e presenta inoltre funzioni di conversione on-

the-fly per i contenuti multimediali in formati leggibili dai browser web. L'integrazione è però definita a mano dall'utente attraverso l'uso di un interfaccia grafica, l'unico supporto fornito dal sistema è la ricerca sul repository dei metadati, per suggerire i possibili elementi che potrebbero essere collegati tramite mapping. Il sistema non fornisce un processo automatico per la scoperta e la creazione dei mapping.
L'installazione del sistema non è semplice, necessita di diversi componenti oltre al modulo per l'information integration, tra cui un modulo Server per le applicazioni. La documentazione per l'installazione non è chiara soprattutto quando dovrebbe indicare i prerequisiti di ogni modulo. Anche la configurazione del sistema non è banale.

4.2.3 Microsoft SQL Server 2005 Integration Services: caratteristiche, pregi e difetti

Microsoft SQL Server 2005 Integration Services (SSIS), è uno strumento di integrazione dati di SQL Server 2005 che consente l'estrazione, la trasformazione e il caricamento di dati (ETL). SSIS consente di estrarre una sorgente dati per caricare i suoi dati in SQL Server in seguito ad eventuali trasformazioni. Per lavorare con SSIS è necessario il framework Microsoft Visual Studio. L'ambiente di lavoro è suddiviso in 3 parti fondamentali:

- Il *Control Flow*: permette di controllare, ordinare e dissociare i task da implementare per la gestione dell'operazione di importazione dei dati.
- Il *Data Flow*: permette di gestire flussi dati da trasformare. In questa sessione, si effettua la selezione, la trasformazione e l'importazione dei dati da elaborare. Esso comprende un "Connection Manager" per la gestione delle connessioni con tutti tipi di sorgente di dati.
- L'*Event Handler*: Gli eventi possono essere associati ai task coinvolti nell'operazione di gestione della fase di importazione, di trasformazione e di selezione dei dati da elaborare.

La trasformazione dei dati si effettua mediante "Script Component". Per ogni colonna della sorgente da trasformare, viene creata una colonna intermedia corrispondente, con lo scopo di dissociarle. Questa nuova colonna avrà lo stesso formato di quella della tabella finale, sulla quale verranno integrati i dati. E' possibile dare alla colonna intermedia un nome e specificare un nuovo tipo di dato. Quindi la tabella intermedia

viene creata mediante gli strumenti visuali dello Script Component, mentre il Design Script è il modulo che permette di scrivere il codice per la trasformazione dei dati della sorgente. Dopo aver stabilito la connessione con la sorgente mediante il Connection Manager, creato la tabella intermedia e infine scritto il codice per la trasformazione dei dati, vengono definiti i mapping tra la tabella intermedia e la tabella finale: per ogni colonna della tabella intermedia si specifica la colonna corrispondente nella tabella di destinazione. Non vi è un processo che automatizzi il mapping dei dati tra le tabelle, la definizione dei mapping è a carico dell'utente.

I moduli utilizzati nel processo di integrazione sono estremamente user-friendly, ed è possibile definire il flusso di integrazione totalmente da interfaccia grafica, trascinando gli oggetti (Script Component) direttamente nell'aerea di lavoro del progetto. Le funzioni messe a disposizione per la trasformazione sono proprie del sistema, e differiscono in parte dalle funzioni SQL standard.

Inoltre sono previsti particolari Script Component per l'individuazione e la gestione delle modifiche al database integrato, e per il controllo dei dati nuovi ed obsoleti; ciò è permesso grazie alla definizione di particolare regole sugli schemi dei dati integrati. Il sistema può interfacciarsi con vari DBMS tramite provider OLE DB.

Il processo di installazione è molto semplice, e non c'è bisogno di ulteriori configurazioni. Infatti il pacchetto Integration Services viene installato contemporaneamente ad SQL Server 2005, utilizzando il proprio DBMS per l'integrazione dei dati in fase di progetto.

4.3 MOMIS a confronto con gli altri sistemi

Un primo confronto, tra MOMIS ed i sistemi di integrazione commerciali analizzati, è quello relativo alla presenza di un dizionario di riferimento per il mapping degli attributi definiti per creare le viste globali sui dati delle varie sorgenti. Riguardo alla logica del processo di integrazione, i sistemi di integrazione testati possono essere suddivisi in due principali categorie: MOMIS e l'IICE di IBM hanno un approccio virtuale, creando, anche se in maniera differente, delle viste virtuali globali dei dati di origine; mentre l'SSIS e l'Oracle DI, che seguono, rispettivamente, le logiche di integrazione ETL ed E-LT (ossia con trasformazione dei dati direttamente sul database,

anziché ETL), permettono di creare un mapping tra gli attributi della tabella sorgente e la tabella target in cui vengono caricati i dati da integrare, non creano perciò una vista globale di tutti i dati delle sorgenti d'origine, ma materializzano nel database integrato i dati integrati.
La definizione degli attributi globali è uno step cruciale nel processo di integrazione dati, ed avere a disposizione un dizionario comune per il mapping aiuta molto l'utente nella definizione degli attributi globali. È da sottolineare il fatto che solo il sistema MOMIS, tra i sistemi considerati, mette a disposizione un insieme di funzioni che permettono di scoprire i mapping tra le sorgenti da integrare e di costruire in modo semi-automatico lo schema globale/integrato. Infatti l'algoritmo di annotazione automatica implementato in MOMIS seleziona per un lemma il primo senso indicato da WordNet, il più utilizzato secondo il DB lessicale utilizzato come riferimento dal sistema di integrazione. Con questo metodo viene parzialmente automatizzato dal sistema il mapping degli attributi, lasciando comunque all'utente la possibilità di raffinare il mapping con aggiunta di termini nuovi al dizionario, o di modificare il senso del lemma associato in maniera automatica dal sistema. Tutto ciò perfeziona comunque il processo di mapping. MOMIS, alla fine del processo di integrazione, crea le classi globali senza memorizzare nel database integrato i dati, che memorizzati nelle sorgenti dati coinvolte nel processo.

Nel sistema IICE di IBM l'utente ha, invece, la possibilità di creare una vista globale degli attributi, creando delle associazioni tra i dati delle varie sorgenti; con questo approccio, però, il lavoro di mapping è completamente delegato all'utente, ed è compiuto manualmente. Non esiste quindi un metodo, nel software IBM, che automatizza, anche parzialmente, la procedura di mapping; è l'utente, in questo caso, che crea la mappa degli attributi, e solo su queste associazioni può esprimere le query federate. Inoltre l'IICE non fa uso di nessun database integrato, infatti la "Mapping Table" viene generata esclusivamente per permettere di esprimere le query federate su attributi globali, e di conseguenza permettere solo ed esclusivamente di personalizzare una query federata per le varie sorgenti che possiedono i propri attributi locali mappati nelle associazioni dati della mapping table specificata in una determinata query federata. Diversamente, nei sistemi di integrazione di Microsoft e Oracle, la definizione dei mapping tra attributi locali e gli attributi degli schemi dati integrati avviene nel momento in cui si implementa il processo di integrazione, definendo la corrispondenza

tra gli attributi della tabella sorgente (che contiene i dati da integrare) e la tabella destinazione (che contiene gli attributi globali, sui quali effettuare l'integrazione). Sia nell'SSIS che nel Data Integrator, il mapping viene implementato manualmente dall'utente, tramite apposite interfacce grafiche che rendono molto agevole il processo, ma non vi è nessun metodo che automatizzi questo procedimento. Se l'utente non definisce come collegare i dati delle sorgenti al database integrato, questi non vengono collegati.

MOMIS risulta essere l'unico sistema di integrazione, tra quelli considerati, che effettua un mapping semi-automatico degli attributi, grazie all'impiego del database lessicale WordNet; mentre negli altri sistemi non vi è un metodo che automatizzi questo processo. Inoltre MOMIS permette di scoprire le relazioni di join tra le sorgenti da integrare, in base ad algoritmi di record linkage, mentre nei sistemi di integrazione Microsoft e Oracle le relazioni di join devono essere definite manualmente dall'utente.

MOMIS permette di inserire delle espressioni di trasformazione sui dati tramite funzioni *like* SQL92, tramite l'ausilio di interfacce grafiche dedicate in maniera molto semplice e veloce. Anche i sistemi Oracle e Microsoft possiedono alcune funzioni di trasformazione, ma sono differenti dallo standard SQL, essendo personalizzate ciascuna per il proprio ambiente di sviluppo. Al contrario l'IICE ha un meccanismo completamente diverso, e molto più complicato per implementare trasformazioni sui dati: infatti in questo sistema è richiesta la conoscenza, da parte dell'utente, di un minimo di programmazione Java, per poter sviluppare le classi che attuano l'unfolding delle query e l'elaborazione sugli stessi risultati tramite funzioni proprie del linguaggio di programmazione.

Dalla differente logica di integrazione, implementata dai sistemi considerati, ne consegue un'altra importante differenza qualitativa per quanto riguarda l'integrazione dei dati effettuata dai sistemi considerati: MOMIS è l'unico sistema che effettua *l'object identification*, implementando tra le classi globali (associate alle sorgenti locali) il "full outer join", a differenza dell'IICE, che permette di integrare le sorgenti dati con una vista globale, tramite le associazioni dati definite nel "Data map", ma che non effettua l'*object identification* tra i dati da integrare. In poche parole il sistema IBM non permette di specificare delle relazioni di join tra tabelle contenute in diverse sorgenti dati. Ciò è una forte limitazione, che incide fortemente sulla qualità del processo di integrazione, e di conseguenza sull'organizzazione stessa dei dati integrati. Questa limitazione è presente anche nei sistemi Microsoft SSIS e Oracle DI: infatti questi

permettono sì di specificare delle condizioni di join sulle tabelle, ma solo fra quelle contenute sulla stessa sorgente dati, non permettendo quindi di avere un *object identification* a livello globale. Da questo punto di vista, MOMIS risulta attuare una migliore ed efficiente integrazione dati, e si pone a livello superiore rispetto agli altri sistemi di integrazione analizzati. Ciò comporta notevoli differenze anche sugli schemi dei dati ottenuti dopo aver eseguito i processi di integrazione, anche se, nell'ambito dell'analisi comparativa fra i sistemi, rispetto al benchmark THALIA, ciò non ha influito in maniera sostanziale sull'implementazione delle varie query, perché queste hanno l'obiettivo di testare il sistema di integrazione dal punto di vista della risoluzione delle eterogeneità sintattiche e semantiche degli attributi. Infatti, è insito nella logica del benchmark che le diverse sorgenti dati non contengono elementi dati comuni: per fare un esempio, non succede mai che, nei dati di origine forniti dal benechmark, in due tabelle di diversi sorgenti ci siano informazioni che si possano riferire ad una stessa entità. Si presuppone cioè, per la risoluzioni delle query, che gli insiemi di dati presenti sulle varie sorgenti, siano disgiunti. Ma questa è solo una caratteristica intrinseca dei dati forniti dal benchmark THALIA, che considera vari database di diverse università mondiali.

Come svantaggio, invece, MOMIS risulta avere una minor compatibilità rispetto alle varie tecnologie DBMS presenti sul mercato: infatti come database integrato supporta solo Microsoft SQL Server 2000; al contrario, in Microsoft SSIS e Oracle DI, il repository dei metadati (schema globale/target) può essere memorizzato su diversi RDBMS. Per di più, nei sistemi di integrazione di IBM, Microsoft e Oracle c'è un ampio supporto per l'integrazione di varie origini dati e tipi di RDBMS, mentre in MOMIS sono stati sviluppati solo alcuni wrapper per il collegamento alle principali tipologie di sorgenti dati (Oracle, SQLServer, MySQL, JDBC generico, JDBC-ODBC-MS_Access, file XML, OWL). Dal punto di vista delle interfacce grafiche per la gestione del processo di integrazione, sicuramente MOMIS si pone ad un livello inferiore rispetto ai sistemi commerciali presi in considerazione, quindi potrebbe essere un ulteriore miglioramento da apportare al sistema.

Inoltre, non esistono attualmente in MOMIS funzioni per la gestione delle modifiche del database integrato materializzato.

Nella seguente tabella è riportato un confronto tra le caratteristiche principali dei diversi sistemi di integrazione.

	Produttore	Tipi sorgenti dati	Approccio	Creazione vista	Query manager
MOMIS	DBGROUP-UNIMO	Semistrutturati e strutturati	Database virtuale (GAV)	Semiautomatica	SI
IICE 8.4	IBM	Strutturati, semistrutturati, dati multimediali	Database virtuale (LAV)	Manuale (interfaccia grafica)	NO*
Data Integrator	Oracle	Semistrutturati e strutturati	DB materializzato (E-LT)	Manuale (interfaccia grafica)	SI
Integration Services	Microsoft	Semistrutturati e strutturati	DB materializzato (ETL)	Manuale (interfaccia grafica)	SI

**Le query federate vengono lanciate da riga di comando tramite uno script*

4.4 Costi delle licenze

Per affrontare un'analisi coerente sui costi delle licenze bisogna, innanzitutto, tener conto che alcuni produttori, a differenza di altri, includono funzionalità complete nei prodotti di base.

Per quanto riguarda i costi delle licenze dei sistemi di integrazione utilizzati in questo progetto di tesi, da questa analisi basata sui viene escluso il sistema MOMIS reso disponibile come progetto *open source* per la comunità scientifica. I sistemi di integrazione dati di Microsoft, Oracle ed IBM, sono invece concessi tramite licenze, con costi diversi tra loro. Inoltre, ogni produttore ha diverse strategie di vendita sul mercato, e spesso, alle licenze delle versioni base dei prodotti, si aggiungono costi di licenze per i componenti aggiuntivi.

Ad esempio, per ottenere le funzionalità dal pacchetto Integration Services, i clienti Microsoft sono costretti ad acquistare la licenza della versione Enterprise di Microsoft SQL Server 2005, che mette a disposizione Integration Services (SSIS) come modulo aggiuntivo in fase di installazione. Invece IBM, nella versione base di WebSphere Information Integrator Content Edition (IICE), include funzionalità limitate, rendendo disponibili ulteriori funzionalità tramite l'acquisto di altre licenze per varie opzioni e componenti aggiuntivi. Queste opzioni possono avere un costo molto

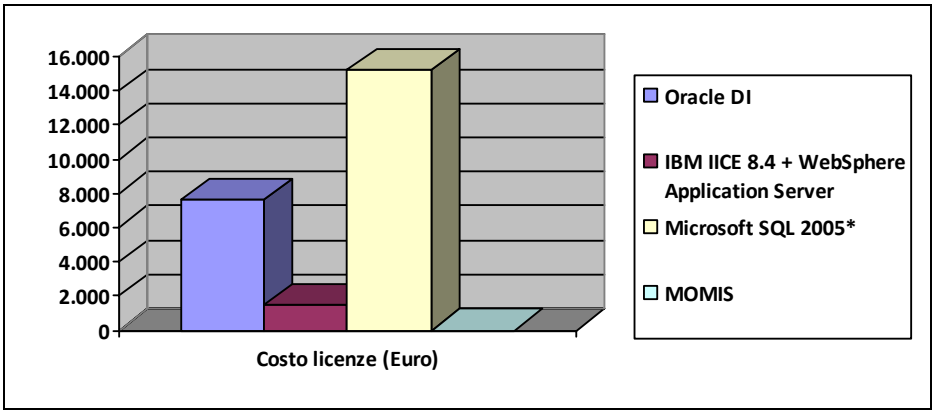
elevato, con un prezzo che talvolta è perfino superiore a quello del prodotto di base. Per questo motivo, in fase di valutazione, è importante che il cliente sappia quali sono le funzionalità incluse nel prodotto di base e quali sono disponibili solo con l'acquisto di licenze aggiuntive. Nel caso di IBM alcuni componenti aggiuntivi, quali i connettori (sviluppati per supportare le varie tecnologie di DBMS, o semplicemente i vari tipi di sorgenti dati tipo file XML, etc) fanno parte di una licenza esterna distinta da quella base, e figurano come componenti *out of box*. Ciò comporta una forte limitazione nell'uso delle funzionalità del prodotto e costringe l'utente ad un'ulteriore spesa per l'acquisto di moduli, in questo caso i connettori, che diversamente sarebbe molto oneroso e difficile sviluppare autonomamente. Inoltre, per il funzionamento dell' IICE è necessario il server delle applicazioni, e la miglior soluzione per la compatibilità dei prodotti è quella di utilizzare IBM WebSphere Application Server, quindi altri costi aggiuntivi per le licenze.

Microsoft ha invece adottato una diversa strategia: fornire funzionalità complete per la gestione e l'analisi dei dati nelle edizioni Enterprise di SQL Server. In questo modo i clienti hanno la certezza di non dover affrontare ulteriori costi incrementali per l'utilizzo dei propri sistemi di gestione dei dati. È giusto sottolineare, però, che il costo di licenza del pacchetto Microsoft è superiore e non di poco ai costi delle licenze IBM. Oracle, invece, si interpone tra i due sistemi, considerando il prezzo della licenza, ed offre nel pacchetto tutte le funzionalità necessarie per eseguire processi di integrazione dati completi. Naturalmente MOMIS, ponendosi come progetto *open source* risulta essere esente da costi. Nella seguente tabella sono illustrati i costi totali delle licenze per i sistemi di integrazione analizzati.

Software	**Costo licenze (Euro)**
Oracle DI	**7.629**
IBM IICE 8.4 + WebSphere Application Server	**1.441**
Microsoft SQL 2005*	**15.206**
MOMIS	***open source***

**Il pacchetto comprende all'interno Integration Services*

Grafico relativo alla tabella precedente:



Comunque, effettuare una comparazione tra i prezzi delle licenze dei sistemi di integrazione analizzati in precedenza, è molto complicato. Il confronto tra i prodotti risulta difficile a causa della complessità dei modelli di licensing. Anche se in un primo momento alcuni software possono apparire economici, spesso i produttori impongono costi nascosti che si aggiungono al prezzo di acquisto iniziale e con il passare del tempo i clienti si trovano costretti ad affrontare spese considerevoli per funzionalità aggiuntive che pensavano fossero già integrate nel prodotto.

4.5 Risultati benchmark THALIA

In questo paragrafo è stata fatta un'analisi comparativa tra i sistemi di integrazione, sulla base dei risultati prodotti dall'implementazione delle query del benchmark THALIA. Come si può notare nella tabella sottostante, sono illustrati i risultati ottenuti applicando il benchmark a ciascun sistema di integrazione dati/ETL; inoltre si noti che i risultati realativi ad Oracle e Microsoft sono relativi ad una soluzione di integrazione non virtuale.

LEGENDA:

Si = query risolta
No = query non risolta
☆ = difficoltà d'implementazione difficile
☆☆ = difficoltà d'implementazione media
☆☆☆ = difficoltà d'implementazione facile

	MOMIS	IBM IICE	Oracle DI	Microsoft IS
	Eterogeneità attributi			
Query 1	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆
Query 2	*Si* ☆☆	*Si* ☆	*Si* ☆☆	*Si* ☆☆☆
Query 3	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆
Query 4	*Si* ☆☆☆	*Si* ☆	*Si* ☆☆☆	*Si* ☆☆☆
Query 5	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆
	Informazione mancante			
Query 6	*Si* ☆☆☆	*Si* ☆☆	*Si* ☆☆☆	*Si* ☆☆☆
Query 7	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆	*Si* ☆☆☆
Query 8	*Si* ☆☆☆	*Si* ☆☆	*Si* ☆☆☆	*Si* ☆☆☆
	Eterogeneità strutturali			
Query 9	*Si* ☆☆☆	*No**	*Si* ☆☆☆	*Si* ☆☆☆
Query 10	*Si* ☆☆☆	*No**	*Si* ☆☆☆	*Si* ☆☆☆
Query 11	*Si* ☆☆☆	*Si* ☆	*Si* ☆☆☆	*Si* ☆☆☆
Query 12	*Si* ☆☆	*Si* ☆	*Si* ☆☆	*Si* ☆☆

**Possono essere risolte creando delle viste sulle sorgenti dati*

In particolare nella tabella precedente, per ogni query è indicata l'informazione relativa alla sua implementazione sul particolare sistema, con un'informazione indicativa della

difficoltà riscontrata nell'implementare il raffinamento sull'integrazione dati, tramite i metodi messi a disposizione dai rispettivi sistemi.
Dai risultati in tabella, si può subito osservare che non è stato possibile implementare le query 9 e 10 nel sistema IICE di IBM, poiché non è stato possibile definire, nella query federata, una relazione di join tra le due tabelle contenute in una delle due sorgenti chiamate in causa dalla query. Questo perché, di per sé, il sistema IBM non permette di definire relazioni di Join tra le diverse strutture dati delle varie sorgenti. Il problema potrebbe comunque essere risolto, creando delle viste locali sui dati della particolare sorgente dati, ed integrare la vista nelle associazioni dati definite nell'IICE. Per il sistema MOMIS non si è verificato nessun problema, in quanto viene eseguito in fase di integrazione un full-outer-join tra le classi globali da integrare. Nei sistemi Microsoft e IBM è stato possibile esprimere le relazioni di join relativamente a tabelle contenute nella stessa sorgente, e per i particolari casi di integrazione, relativi alle query 9 e 10, il problema è stato risolto in questo modo.

Oltre a questa prima valutazione, si può che come la difficoltà di implementazione delle procedure di raffinamento sui dati da integrare, sia stata particolarmente più elevata nel caso del software di integrazione di IBM, dove per ogni raffinamento si è dovuta sviluppare una classe Java dedicata, ed implementare i metodi per eseguire l'unfolding della query federata ed un altro metodo per elaborare i risultati ottenuti, per presentarli in una formattazione compatibile con i requisiti dettati dal benchmark per la particolare query. Mentre nei sistemi Microsoft ed Oracle le trasformazioni sui dati sono state implementate tramite funzioni ad-hoc messe a disposizione dai sistemi stessi, al contrario di MOMIS che supporta funzioni *like* SQL92. La difficoltà principale è stata quella di creare delle espressioni di trasformazione con queste funzioni, per manipolare i dati e strutturali in un modo tale da essere coerenti per le richieste del benchmark.
C'è da sottolineare il fatto che i dati integrati sui quali sono state eseguite le query del benchmark non risultano avere la stessa struttura in ognuno dei processi nei vari sistemi; ciò è diretta conseguenza del modo in cui i diversi sistemi implementano i processi di integrazione: per esempio solo con MOMIS si ottengono degli schemi integrati con la caratteristica dell'object identification, caratteristica invece non presente negli altri sistemi di integrazione dati analizzati. Il risultato delle query è comunque risultato uguale nei vari sistemi, ma questo è dovuto al fatto che i dati del benchmark

THALIA, con i quali sono state effettuate le integrazioni, contenevano insiemi disgiunti di dati, relativamente alle varie sorgenti.

Per quanto riguarda la query 1 e la query 3, queste sono state risolte in tutti i sistemi tramite semplice mapping tra gli attributi delle sorgenti integrate in ogni query. Mentre per risolvere la query 2, è stato necessario implementare una funzione di trasformazione particolare, per risolvere l'eterogeneità degli attributi che contenevano gli orari in diversi formati. Stesso discorso per le query 4 e 5, dove è stato possibile risolvere le eterogeneità tramite le funzioni di trasformazione.

Nelle query 6, 7 ed 8, dove era necessario il trattamento dei valori nulli, nel sistema IICE di IBM, nel result set di ogni query, per gli attributi sui quali non è stato possibile definire un mapping con gli attributi dell'altra sorgente da integrare, è stato necessario ricorrere ad un "trucco", cioè si è impostato un valore nullo di default.

Conclusioni

Dal lavoro compiuto in questa tesi si possono trarre risultati molto interessanti. In primo luogo si può affermare che per quanto riguarda la logica del processo di integrazione, i sistemi di integrazione testati, possono essere suddivisi in due grandi categorie: MOMIS e l'IICE di IBM hanno un approccio virtuale, cioè non materializzano il database dei dati integrati, ma creano, anche se in maniera differente, delle viste virtuali dei dati di origine; i sistemi Microsoft SSIS e Oracle DI, che seguono rispettivamente le logiche di integrazione ETL ed E-LT (ossia con trasformazione dei dati direttamente sul database, anziché ELT), permettono di creare un mapping tra gli attributi della tabella sorgente e la tabella target in cui vengono caricati i dati da integrare, non creano perciò una vista globale di tutti i dati delle sorgenti d'origine, ma materializzano il database dei dati integrati.

Tra i punti di forza di MOMIS rispetto agli altri sistemi, vi è, innanzitutto, la capacità di effettuare un mapping semi-automatico degli attributi, grazie all'impiego del database lessicale WordNet; mentre negli altri sistemi non vi è un metodo che automatizzi questo processo, ed il lavoro è delegato completamente all'utente attraverso una potente interfaccia grafica. Inoltre, MOMIS è l'unico sistema che effettua *l'object identification*, implementando tra le classi globali (associate alle sorgenti locali) il "full outer join"; al contrario degli altri sistemi che non eseguono l'object identification a livello globale. Tra gli svantaggi di MOMIS, si può evidenziare il fatto che supporta un solo DBMS target, mentre gli altri sistemi supportano più tecnologie. Inoltre MOMIS potrebbe essere migliorato dal punto di vista della interfaccia grafica, dove paga, i termini di qualità, rispetto agli altri sistemi commerciali. Queste due note di demerito di MOMIS, possono essere considerate oggetto di uno sviluppo futuro del sistema.

Per quanto riguarda i risultati ottenuti dall'implementazione delle query del benchmark THALIA, questi possono essere considerati, in linea generale, positivi: infatti, ad eccezione dell'IICE (2 query non sviluppate), su tutti gli altri sistemi è stato possibile implementare tutte le query, e risolvere i problemi di eterogeneità anche se in maniera differente. La difficoltà di implementazione è risultata essere abbastanza contenuta nei sistemi Oracle, Microsoft e MOMIS, mentre nel sistema IBM si è riscontrata una

difficoltà implementativa maggiore, a causa dei metodi di trasformazione che sono stati sviluppati interamente con classi Java.

Infine, il benchmark THALIA è risultato essere un buon strumento di valutazione, e tramite la sua applicazione è stato possibile scoprire come i principali sistemi di integrazione commerciali e MOMIS affrontano i problemi attuali legati all'integrazione dell'informazione.

Come ultima osservazione va ricordato che l'approccio virtuale del sistema IBM e di MOMIS consente la piena autonomia delle sorgenti dati da integrare mentre l'approccio Oracle e MICROSOFT obbligano a replicare le sorgenti da integrare su un unico server con tutti i problemi collegati all'allineamento dei dati in caso di modifica dei dati nelle sorgenti.

Ricordo ancora che un'analisi di prodotti di data integration open source e` inclusa nella tesi del collega Francesco Nigro "Data River: un sistema di integrazione dati open source" [30]. In tale tesi viene proposto il progetto di trasformazione del sistema MOMIS in un prodotto open source per la comunità scientica.

Bibliografia

[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, AI*IA Notizie Journal, Year XIII, N. 2

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

[3] D. Beneventano, S.Bergamaschi
"Semantic search engines based on data integration systems". In Semantic Web Services: Theory, Tools and Applications. IGI Global, Information Science Reference, Hershey, New York, 2006.

[4] http://www-06.ibm.com/software/data/integration/db2ii/editions_content.html

[5] http://www.oracle.com/technology/products/oracle-data-integrator

[6] http://www.microsoft.com/italy/server/sql/technologies/integration/default.mspx

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

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

[9] http://www.cise.ufl.edu/research/dbintegrate/thalia/index.htm

[10] http://www.cise.ufl.edu/research/dbintegrate/thalia/index.htm

[11] http://telegraph.cs.berkeley.edu/tess/

[12] http://www-306.ibm.com/software/webservers/appserv/was/

[13] http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=GC18-9619-01

[14] http://www-306.ibm.com/software/data/enterprise-search/omnifind-enterprise/

[15] http://www.oracle.com/technology/products/oracle-data-integrator/pdf/oracledi_architecture.pdf

[16] http://www.oracle.com/technology/products/oracle-data-integrator/pdf/oracledi_tech_whitepaper.pdf

[17] http://www.oracle.com/technology/products/oracle-data-integrator/10.1.3/htdocs/documentation/oracledi_users.pdf

[18] http://www.oracle.com/technology/products/oracle-data-integrator/10.1.3/htdocs/documentation/oracledi_km_reference.pdf

[19] Documentazione Microsoft SQL Server 2005 Integration Services

[20] “Sigmod Record, Vol. 28 No. 1“, http://www.sigmod.org/sigmod/record/issues/9903/

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

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

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

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

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

[26] 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

[27] http://www.gutenberg.org

[28] http://www.ibm.com/developerworks/forums/thread.jspa?threadID=186220&tstart=0

[29] http://www.ibm.com/developerworks/forums/thread.jspa?threadID=187216&tstart=0

[30] Francesco Nigro, “ Data River: un sistema di integrazione dati open source”