MO Universita' degli Studi di Modena

InterData

Programma di ricerca (cofinanziato dal MURST, esercizio 1997)
Metodologie e tecnologie per la gestione di dati e processi su reti Internet e Intranet

Stato dell'arte della ricerca nell'ambito dell'estrazione/integrazione intelligente di informazioni

Sonia Bergamaschi, Domenico Beneventano, Alberto Corni, Maurizio Vincini

T3-R01 **16 luglio 1998**

Sommario

Il presente Rapporto si pone l'obiettivo di fornire una panoramica dello stato dell'arte della ricerca relativamente all'integrazione intelligente di informazioni da fonti eterogenee. Sono statiesaminati tre progetti, TSIMMIS, GARLIC e SIMS, tra i più importanti presenti nell'ambito della ricerca. La scelta di questi sistemi è motivata anche dal loro diverso approccio all'integrazione, così da mostrare la pluralità delle tendenze di ricerca.

Tema	Tema 3: Integrazione di informazioni estratte da siti WWW
Codice	T3-R01
Data	16 luglio 1998
Tipo di prodotto	Rapporto tecnico
Unità responsabile	MO
Unità coinvolte	MO
Autori	Sonia Bergamaschi, Domenico Beneventano, Alberto Corni, Maurizio Vincini
Autore da contattare	Sonia Bergamaschi Dipartimento di Scienze dell'Ingegneria Università degli Studi di Modena Via Campi 213/B, 41100 Modena, Italia sonia@dsi.unimo.it

Stato dell'arte della ricerca nell'ambito dell'estrazione/integrazione intelligente di informazioni

Sonia Bergamaschi Domenico Beneventano Maurizio Vincini

Sommario

Il presente Rapporto si pone l'obiettivo di fornire una panoramica dello stato dell'arte della ricerca relativamente all'integrazione intelligente di informazioni da fonti eterogenee. Sono statiesaminati tre progetti, TSIMMIS, GARLIC e SIMS, tra i più importanti presenti nell'ambito della ricerca. La scelta di questi sistemi è motivata anche dal loro diverso approccio all'integrazione, cosı̀ da mostrare la pluralità delle tendenze di ricerca.

Nella sezione 1 viene presentato il progetto di ricerca TSIMMIS, sviluppato presso l'Università di Stanford. Le sottosezioni 1.1 e1.2 descrivono, rispettivamente, il modello dei dati OEM e il linguaggio MSL utilizzati nel progetto. Le successe sottosezioni descrivono l'architettura e i moduli del sistema di integrazione proposto; infine la sottosezione 1.6 descrive brevemente pregi e difetti dell'approccio.
La sezione 2 presenta il progetto GARLIC, sviluppato presso il centro di ricerca IBM di Almaden, descrivendo il linguaggio GDL (in 2.1), il query planning (in 2.2) e pregi e difetti dell'approccio (in 2.3).
La sezione 3 presenta il progetto SIMS, sviluppato presso l'Università della Southern California, descrivendo il processo di integrazione delle sorgenti (in 3.1), il query processing (in 3.2) e pregi e difetti dell'approccio (in 3.3).

L'Integrazione Intelligente di Informazioni è un campo di ricerca relativamente nuovo, esistono tuttavia in letteratura, diversi sistemi che cercano di realizzare, in modo più o meno efficiente e flessibile, un modulo integratore (nei casi più riusciti) o un semplice modulo di ricerca di informazioni.
Si è quindi ritenuto utile esaminare tre tra i progetti più importanti presenti nell'ambito della ricerca, scelti per differente approccio all'integrazione, cosı̀ da mostrare le principali linee di ricerca.
Il presente documento non vuole comunque essere esautivo, sia per i progetti presentati, ai quali si rimanda per un'analisi più approfondita, sia per altri approcci presenti in letteratura, tra i quali citiamo [PS97, MR96, SAB$^+$, LFK$^+$].

1 TSIMMIS

TSIMMIS (**T**he **S**tanford- **I**BM **M**anager of **M**ultiple **I**nformation **S**ources) [GMea95, HMJ95, YAGM96] è sicuramente uno dei progetti più interessanti in questo campo: sviluppato presso l'Università di Stanford in collaborazione con il Centro di ricerca IBM di Almaden, si pone come obiettivo lo sviluppo di strumenti che facilitino la rapida integrazione di sorgenti testuali eterogenee, includendo sia sorgenti di dati strutturati che **semistrutturati**. Questo obiettivo è raggiunto attraverso un'architettura comune a molti altri sistemi: i *wrapper* convertono i dati in un modello comune mentre i *mediator* combinano ed integrano i dati ricevuti dai wrapper. I wrapper inoltre forniscono un linguaggio di interrogazione comune per l'estrazione delle informazioni, mostrando un'interfaccia verso l'esterno uguale a quella dei mediatori: in questo

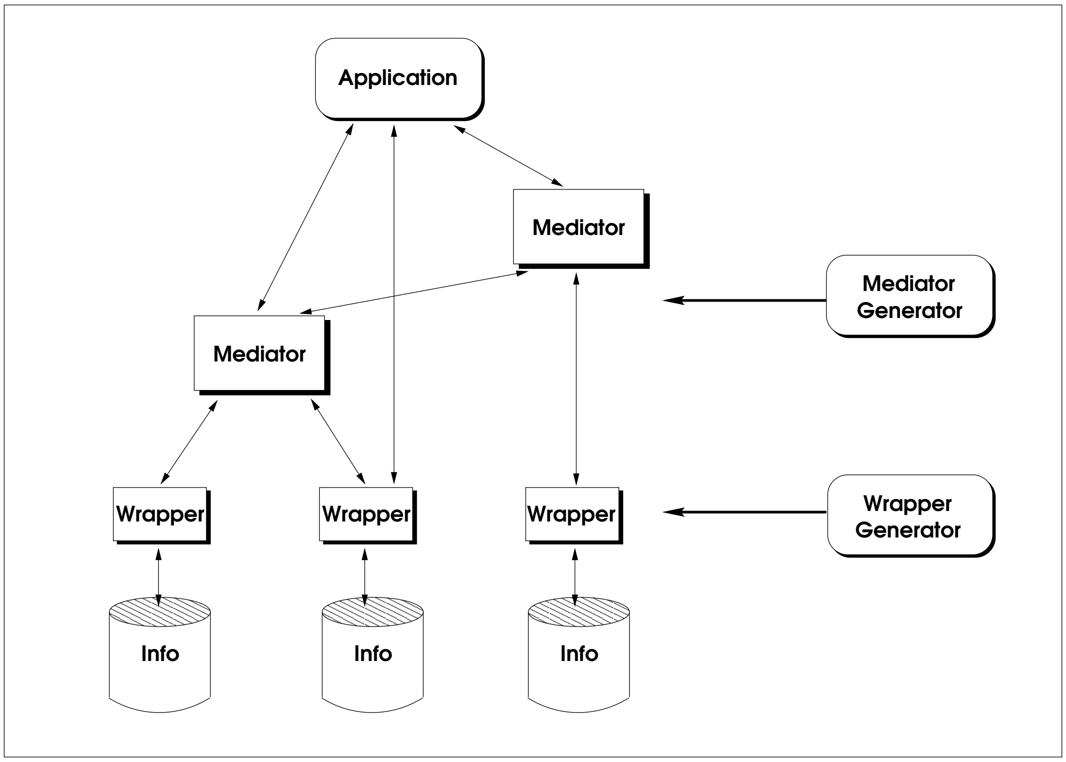

Figura 1: Architettura TSIMMIS

modo, l'utente può porre le interrogazioni attraverso un unico linguaggio sia ai mediatori (ricevendo dati integrati da più sorgenti), sia direttamente ai wrapper (interrogando in questo modo un'unica fonte).

In Figura 1 è mostrata l'architettura: ad ogni sorgente corrisponde un wrapper (o *traduttore*) che converte nel modello comune i dati estratti dalla sorgente; sopra i wrapper stanno i mediatori. I wrapper convertono inoltre le query scritte utilizzando il modello comune in richieste comprensibili dalla particolare sorgente da loro servita.

La peculiarità di questo progetto si manifesta nel modello comune dei dati utilizzato per rappresentare le informazioni. **OEM** (Object Exchange Model) [YHMJ94] è un modello a *etichette* basato sui concetti di identità di oggetto e annidamento particolarmente adatto a descrivere dati la cui struttura non è nota o è variabile nel tempo. In aggiunta a questo, sono disponibili ed utilizzati dal sistema due linguaggi di interrogazione (*OEM-QL*) e *MSL* (Mediator Specification Language), per ottenere i dati dalle fonti ed integrarli opportunamente.

Per facilitare il compito dell'amministratore del sistema, sono stati progettati due moduli (*translator generator* e *mediator generator*) che supportano le fasi di realizzazione rispettivamente dei wrapper e dei mediatori, fornendo un insieme di librerie di funzioni predefinite.

1.1 Il modello OEM

Si presenta brevemente il modello OEM, fondamentale per capire il tipo di approccio (*strutturale*) dell'intero progetto TSIMMIS. OEM fa parte dei cosiddetti *self describing model*, dove ad ogni informazione è associata una etichetta che ne descrive il significato. Esso comprende caratteristiche tipiche dell'approccio ad oggetti ma in modo molto semplificato: non fa uso di un forte sistema dei tipi (in pratica sono ammessi solamente i tipi base), non supporta direttamente né le classi, né i metodi, né l'ereditarietà, bensì solo l'identità e il nesting tra oggetti.

Un esempio di descrizione dell'oggetto **persona** è il seguente:

```
<ob1:  person,  set,         {sub1,sub2,sub3,sub4,sub5}>
       <sub1:   last_name,   str,  'Smith'>
       <sub2:   first_name,  str,  'John'>
       <sub3:   role,        str,  'faculty'>
       <sub4:   department,  str,  'cs'>
       <sub5:   telephone,   str,  '32435465'>
```

L'oggetto `person` considerato viene rappresentato da un'insieme di oggetti annidati: ogni oggetto è rappresentato da stringhe separate da virgole e comprese fra i simboli $<$ e $>$. La prima stringa (ad es. `ob1`) rappresenta l'identificatore, la successiva è un'etichetta che ne qualifica il genere, la terza e la quarta specificano il tipo ed il valore. Fondamentale, per la semantica dell'oggetto stesso, è l'etichetta, che ne descrive il ruolo all'interno del contesto in cui è utilizzato. Nell'esempio riportato, sono descritti in totale sei oggetti: un oggetto che si potrebbe definire di top-level (`person`) e cinque sotto-oggetti, che a `person` sono collegati, come mostra l'ultima stringa dell'oggetto `ob1`. È importante sottolineare che, usando l'OEM, non è necessario che oggetti che si descrivono tramite la stessa etichetta abbiano pure lo stesso schema: per esempio, potrebbe esistere un altro oggetto `person` con un diverso insieme di tipi di sotto-oggetti. In questo modo risulta molto semplificata l'integrazione di oggetti provenienti da schemi diversi e semistrutturati, non dovendo predefinire le strutture che questi oggetti dovranno seguire, ed anzi accettando tutti gli oggetti con una determinata etichetta, qualunque schema seguano.

1.2 Il linguaggio MSL

MSL è il linguaggio utilizzato per definire, in modo dichiarativo, i mediatori: attraverso l'uso di *rule*, si può specificare il punto di vista del mediatore, ed in questo modo definire lo "schema globale" in modo manuale. A run time, ricevuta una particolare richiesta, l'interprete del mediatore (MSI) raccoglie ed integra le informazioni ricevute dalle sorgenti, in accordo con le specifiche definite nella vista. MSL è dunque un linguaggio dichiarativo per la definizione di *viste* in grado di interfacciarsi con OEM e quindi di supportare evoluzioni degli schemi, irregolarità strutturali degli oggetti delle sorgenti, strutture sconosciute senza dover ogni volta ridefinire le viste. Ogni regola è costituita da una *testa* e da una *coda*, separate dal simbolo :-. La *coda* descrive dove andare a recuperare l'oggetto che si vuole ricevere, mentre l'intestazione definisce la struttura che questo oggetto dovrà avere, una volta estratto e ricostruito. Un esempio di regola MSL sarà presentato nella Sezione 1.4

1.3 Il generatore di Wrapper

TSIMMIS include un insieme di strumenti, chiamati *OEM Support Libraries*, per realizzare facilmente i wrapper, i mediatori e le interfacce utenti. Questi strumenti comprendono diverse procedure per lo scambio di oggetti OEM in un architettura Client/Server, dove un Client può essere indifferentemente un mediatore, una applicazione o un'interfaccia utente, mentre il server può essere sia un wrapper, sia un mediatore. In questo modo, si è cercato di semplificare il più possibile la realizzazione dei wrapper, ed in particolare del modulo di Conversione, che ha il compito di convertire una query espressa nel linguaggio MSL in una interrogazione comprensibile dalla sorgente con la quale il wrapper interagisce.

Per facilitare l'intero processo di integrazione, e per poter servire anche sorgenti dalle limitate possibilità di risposta ad interrogazioni, nel progetto TSIMMIS si è ipotizzato di dover esprimere esplicitamente l'intero insieme delle query a cui la sorgente può rispondere (dunque un insieme comunque limitato di query) e di predisporre, attraverso i cosiddetti *query templates*, per ogni query supportabile in MSL, la rispettiva traduzione nel linguaggio di interrogazione proprio della sorgente stessa. Un esempio potrà sicuramente semplificare la descrizione di tutto il processo. Supponiamo di interagire con una base di dati universitaria, *WHOIS*, che mantiene informazioni sugli studenti e sui professori di una data università, e con possibilità molto limitate: le uniche operazioni consentite sono il ritrovamento dei dati di una persona, conoscendone il nome o il cognome (od entrambi). Le interrogazioni supportate dalla sorgente, espresse nel suo linguaggio, potrebbero essere le seguenti:

1. ritrova le persone dato il cognome: `>lookup -ln 'ss'`
2. ritrova le persone dati cognome e nome: `>lookup -ln 'ss' -fn 'ff'`
3. ritrova tutte le informazioni della sorgente: `>lookup`

Ad ognuna di queste interrogazioni corrisponderà un *query template*: le query in entrata al wrapper saranno scritte in MSL, e dovranno essere tradotte, attraverso questi *query template*, nel linguaggio locale. Ad esempio, alle interrogazioni 1 e 2 sopra descritte corrisponderanno le seguenti query MSL:

```
(QT2.1) Query ::= *O :- <O person {<last_name $LN>}>
(AC2.1)           {printf (lookup_query, 'lookup -ln %s',$LN);}
(QT2.2) Query ::= *O :- <O person {<last_name $LN>
                                   <first_name $FN>}>
(AC2.2)           {printf (lookup_query, 'lookup -ln %s -fn %s ',
                            $LN,$FN);}
```

Ad ogni *query template*, è associata una azione, scritta in questo caso in C, che realizza la traduzione da MSL a linguaggio "nativo". Naturalmente, il sistema sarebbe molto limitato se permettesse di rispondere solamente a questo insieme predefinito di interrogazioni: esiste dunque un modulo che permette di definire, data una query in input (che può essere ricevuta da un mediatore, come pure direttamente da una applicazione o da un'interfaccia utente), se ad essa la sorgente sia in grado di dare risposta. In particolare, oltre alle query predefinite, sono supportate tutte le query q tali che:

- q è equivalente ad una query q' direttamente supportata, ovvero gli insiemi delle risposte delle due query coincidono;
- q è sussunta da q' direttamente supportata, ovvero l'insieme risposta di q è incluso nell'insieme risposta di q'.

1.4 Il generatore di Mediatori

Il mediatore, in TSIMMIS, è il modulo che si preoccupa di dare una visione integrata dei dati, agendo su differenti sorgenti. Supponiamo di dover integrare due sorgenti, di cui la prima è una base di dati relazionale, *Computer_Science*, il cui schema è riportato di seguito:

```
employee(first_name,last_name,title,report_to)
student(first_name,last_name,year)
```

mentre la seconda è una sorgente ad oggetti, *WHOIS*. Ad ogni sorgente corrisponde, come già

```
<&e1,  employee,  set,        {&f1,&l1,&t1, &rep1}>
       <&f1,      first_name, string,   'Joe'>
       <&l1,      last_name,  string,   'Chung'>
       <&t1,      title,      string,   'professor'>
       <&rep1,    reports_to, string,   'John Hennessy'>

<&e2,  employee,  set,        {&f2,&l2,&t2}>
       <&f2,      first_name, string,   'John'>
       <&l2,      last_name,  string,   'Hennessy'>
       <&t2,      title,      string,   'chairman'>
..............etc.

<&s3,  student,   set,        {&f3,&l3,&y3}>
       <&f3,      first_name, string,   'Pierre'>
       <&l3,      last_name,  string,   'Huyn'>
       <&y3,      year,       integer,  3>
```

Figura 2: Oggetti esportati da **CS** in OEM

mostrato nell'architettura, un wrapper: **CS** esporta le informazioni della prima, **WHOIS** quelle della seconda (esempi di oggetti esportati da questi wrapper sono rispettivamente riportati in Figura 2 e in Figura 3).

```
<&p1,  person,    set,       {&n1, &d1, &rel1, &elem1}>
       <&n1,      name,      string,  'Joe Chung'>
       <&d1,      dept,      string,  'cs'>
       <&rel1,    relation,  string,  'employee'>
       <&elem1,   e_mail,    string,  'chung@cs'>
..............etc.
```

Figura 3: Oggetti esportati da **WHOIS** in OEM

Si vuole sviluppare un modulo mediatore, chiamato **MED**, che integri tutte le informazioni inerenti una persona, ricavate dai due wrapper. Per esempio, supponendo che la persona in questione si chiami 'Chung', ed appartenga al dipartimento 'CS' (ovvero Computer Science), la risposta che si vorrebbe ottenere è rappresentata in Figura 4.

Per realizzare questa integrazione, TSIMMIS fa uso di un sistema di regole, MSL (Mediator Specification Language), che permettono di specificare la struttura dell'oggetto integrato, a partire dalle strutture degli oggetti da recuperare nelle sorgenti. Le regole devono essere quindi esplicitamente specificate dall'amministratore del sistema, che è l'unico che deve conoscere lo schema di *persona* delle sorgenti e del mediatore. In particolare, riferendoci all'esempio, la rule

```
<&cp1,  cs_person,  set,         {&mn1, &mrel1, &t1, &rep1, &elm1}>
        <&mn1,      name,        string,  'Joe Chung'>
        <&mrel1,    relation,    string,  'employee'>
        <&t1,       title,       string,  'professor'>
        <&rep1,     reports_to,  string,  'John Hennesy'>
        <&elem1,    e_mail,      string,  'chung@cs'>
```

Figura 4: Oggetti esportati da **MED**

che realizza il processo specificato sarà:

```
(MS1) Rule:
<cs_person {<name N> <rel R> Rest1 Rest2}>
          :- <person {<name N> <dept 'cs'> <relation R> | Rest1}>
             @whois
             AND decomp(N, LN, FN)
             AND <R {<first_name FN> <last_name LN> | Rest2}>@cs
External:
decomp(string,string,string)(bound,free,free) impl by name_to_lnfn
decomp(string,string,string)(free,bound,bound) impl by lnfn_to_name.
```

La *testa* della regola `MS1` specifica la struttura che avrà l'oggetto unificato: sarà esportato da **MED** con etichetta `cs_person` con un nome (`name`), un ruolo (`relation`), ed un insieme non specificato di altri attributi, ovvero con tutte le altre informazioni che sarà possibile recuperare dalle due sorgenti (rispettivamente `Rest1` e `Rest2`). Nella *coda* sono invece definiti i percorsi dove devono essere recuperate le informazioni: dalla sorgente **WHOIS** si ricavano oggetti di tipo `person` con un nome definito (lo stesso espresso nell'intestazione della rule), un ruolo, e l'attributo dipartimento uguale a 'cs'; dalla sorgente **CS** sono invece recuperati degli oggetti di tipo `R` (dove `R` è il ruolo specificato nell'intestazione, e può valere '`employee`' o '`student`), e di nome e cognome specificato. In ausilio alla rule vi è inoltre una funzione esterna, `decomp`, che realizza la trasformazione da `name` a `first_name` e `last_name` e viceversa.
In tutto il processo non permane alcuna ambiguità: per prima cosa, sono recuperati dalle sorgenti locali gli oggetti la cui struttura (e le cui etichette) sono conformi alla struttura specificata nella *coda* della rule, poi questi oggetti sono unificati e presentati in modo integrato come specificato nella *testa*.

1.5 Il Linguaggio LOREL

Evidenziato il punto debole del tipo di integrazione realizzata nella scarsa flessibilità dovuta alle interrogazioni predefinite dai *query template*, si è cercato di porvi rimedio sviluppando un linguaggio di interrogazione ad hoc: LOREL (Ligtweight Object REpository Language) [DAY+96]. Caratteristica saliente di questo linguaggio, la cui sintassi si ispira ad SQL ed OQL, consiste nel fatto che esso è in grado di sfruttare strutture predefinite, se presenti, ma non ne ha necessariamente bisogno per fornire risposte significative ad un'interrogazione. Come OEM è un modello ad "oggetti leggeri", nel senso che presenta un ridotto numero di caratteristiche dei tradizionali modelli ad oggetti, così LOREL è definiti un linguaggio di interrogazione per "oggetti leggeri". In questo paragrafo si illustrano solo le caratteristiche salienti di questo linguaggio, rimandandone l'approfondimento alla letteratura.

LOREL si distingue dagli altri linguaggi di interrogazione perchè:

- è in grado di recuperare informazioni anche quando certi dati non sono presenti, attraverso un assegnamento parziale, e non totale, delle tuple o degli oggetti
- non distingue tra attributi multi_valore e a valore singolo (in SQL attributi multivalore non sono ammessi, in OQL sono distinti da una diversa sintassi)
- opera uniformemente su dati che hanno tipi differenti perchè non esegue controllo dei tipi
- consente di interrogare sorgenti la cui struttura è parzialmente sconosciuta utilizzando wildcards in sostituzione dei nomi degli attributi

Queste caratteristiche sono in varia misura dovute al fatto che, al contrario degli altri linguaggi, LOREL non impone un rigido sistema di tipi, anzi tutti i dati, compresi i valori scalari, sono rappresentati da oggetti (con lo stesso paradigma di OEM, quindi): ogni oggetto ha un identificatore, un'etichetta ed un valore, quest'ultimo può essere tanto uno scalare quanto un set di oggetti innestati nell'oggetto iniziale.

1.6 Pregi e difetti di TSIMMIS

Il punto di forza di questo progetto è sicuramente il modello comune OEM adottato: esso permette l'integrazione tanto di oggetti la cui struttura è fissa e nota o variabile nel tempo, quanto di oggetti di struttura non predefinita (si veda nell'esempio l'uso degli attributi `Rest1` e `Rest2` in Sezione 1.4), rendendo possibile ciò che è negato in qualunque ambiente convenzionale orientato agli oggetti che non aderisca ad una semantica di mondo aperto. Queste possibilità rendono l'intero sistema estremamente flessibile, permettendo l'integrazione di sorgenti il cui schema può essere sia parzialmente sconosciuto, sia variabile nel tempo.

Il meccanismo dei *query template* sostiene il fatto che le sorgenti abbiano diverse capacità di rispondere ad un'interrogazione (è infatti inverosimile pensare che tutte possiedano le funzionalità di un DBMS) e semplifica la fase di interrogazione: a causa della assenza di una struttura predefinita l'interrogazione coi tradizionali statement SQL/OQL diventa aleatoria. A questo processo può però essere mosso un appunto: esprimendo le capacità di risposta di una fonte di informazioni attraverso query predefinite, è improbabile che si riesca a rappresentare completamente il range di informazioni estraibili dalle sorgenti: può accadere il caso in cui la fonte non risponde ad una query q ma è invece in grado di realizzare una query q' più generale di q. Basterebbe allora, per realizzare q, avere un filtro presso il mediatore, ed eseguire una ulteriore selezione sui dati ricevuti in risposta a q'.

Per quanto riguarda la soluzione proposta con l'introduzione di LOREL, a causa della novità della proposta, un indubbio svantaggio risiede nel fatto che non si possono comunque manipolare le query e gli oggetti facendo affidamento sulle specifiche funzionalità di query processing dei DBMS esistenti, mentre occorre sviluppare moduli dedicati per le fasi di parsing, query rewrite, optimization and execution. Il vantaggio principale consiste invece nel riuscire a superare alcune delle limitazioni di SQL ed OQL. Inoltre, rispetto alla soluzione coi query template, lascia all'utente una maggior flessibilità di interrogazione (ma indubbiamente anche una maggior complessità).

Rimane poi un altro problema aperto: il processo di integrazione di informazioni è di per sè un processo ambiguo, in cui non si possono conoscere le sorgenti se non in modo approssimativo; perciò risulta una forzatura eccessiva ipotizzare di lasciare esclusivamente al progettista del sistema il compito di individuare i concetti comuni e di integrarli opportunamente. Questo grosso svantaggio è imputabile anche all'approccio *strutturale* scelto, che non richiede (e quindi non utilizza) gli schemi concettuali delle sorgenti. A questo si contrappone l'approccio *semantico*, nel quale sono sfruttate le informazioni codificate negli schemi al fine di pervenire ad una totale integrazione di tutte le fonti di informazioni.

2 GARLIC

L'obiettivo del progetto **GARLIC** [MHS+94, RS95] (sviluppato presso il centro di ricerca IBM di Almaden) è la costruzione di un Multimedia Information System (MMIS) in grado di integrare dati che risiedono in differenti basi di dati, nonché in un insieme di server di diversa natura, mantenendo la reciproca indipendenza dei server ed evitando la replicazione fisica dei dati. *Multimedia* deve in questo contesto essere interpretato in senso molto ampio, indicando non solo immagini, video, e audio ma anche testi e tipi di dati specifici di alcune applicazioni (disegni CAD, mappe, ...). Poiché molte di queste informazioni sono già modellate tramite oggetti, GARLIC fornisce un modello orientato ad oggetti che permette a tutte le differenti sorgenti di descriversi in modo uniforme. A questo modello si aggiunge inoltre un linguaggio di interrogazione, anch'esso orientato ad oggetti (ottenuto con un'estensione al linguaggio SQL) e utilizzato dal livello intermedio dell'architettura GARLIC, i cui compiti sono:

- presentare lo schema globale alle applicazioni,
- interpretare le interrogazioni,
- creare piani di esecuzione per le interrogazioni e
- riassemblare i risultati in un'unica risposta omogenea.

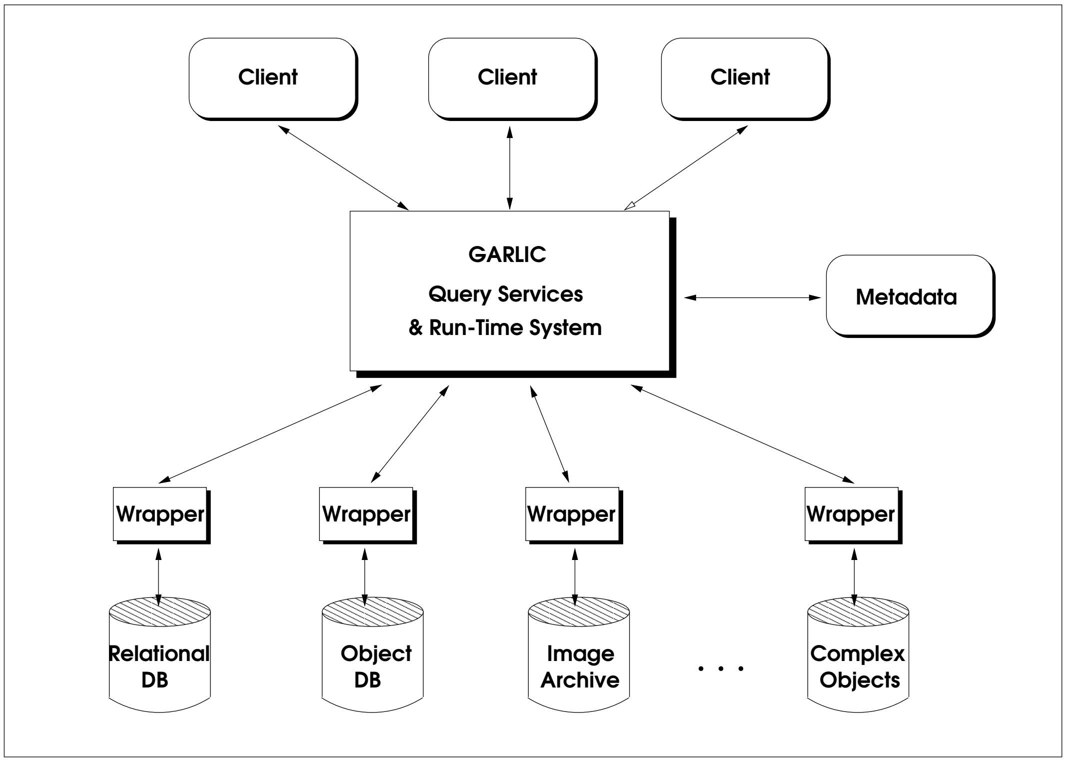

Figura 5: Architettura GARLIC

La Figura 5 rappresenta l'architettura di GARLIC. Alla base della figura vi sono le sorgenti di informazioni che devono essere integrate (tra le quali basi di dati relazionali, ad oggetti, file system, document manager, image manager,...). Sopra ogni sorgente è posizionato un wrapper, che traduce le informazioni sugli schemi, sugli accessi ai dati e sulle interrogazioni, dal protocollo interno di GARLIC ai protocolli nativi delle sorgenti. Le informazioni riguardanti lo schema unificato sono mantenute nel deposito di metadati. L'altra sorgente di informazioni alla base della figura (Complex objects) serve invece per memorizzare gli *oggetti complessi* di GARLIC, utilizzati dalle applicazioni per unire i dati originariamente separati (siano essi appartenenti a schemi distinti, o allo stesso schema). I servizi di query processing sono forniti dal componente *Query Services & Run-Time System*: ad esso spetta il compito di presentare alle applicazioni

Relational Repository Schema

```
interface Country {
   attribute string name;
   attribute string airlines_served;
   attribute boolean visa_required;
   attribute Image scene;
   }
interface City {
   attribute string name;
   attribute long population;
   attribute boolean airport;
   attribute Country country;
   attribute Image scene;
   }
```

Web Repository Schema

```
interface Hotel {
   attribute readonly string name;
   attribute readonly short class;
   attribute readonly doube daily_rate;
   attribute readonly string location;
   attribute readonly string city;
   }
```

Image Server Repository Schema

```
interface Image {
   attribute readonly string file_name;
   double matches (in string file_name);
   void display (in string device_name);
   }
```

Figura 6: GDL schema

una visione unificata, ad oggetti, del contenuto del sistema e di gestirne le richieste (interrogazioni o modifiche). Lo schema globale è presentato all'utente, e alle applicazioni, attraverso il modello di dati di GARLIC (**G**arlic **D**ata **M**odel): è costituito dalla **unione** degli schemi locali. Fra questi è pure disponibile lo schema degli *oggetti complessi*, creati ad hoc dalle applicazioni per avere una visione integrata di oggetti preesistenti. Il fornire una descrizione dei dati attraverso il GDL (**G**arlic **D**ata **L**anguage) permette inoltre di identificare i dati che si vogliono integrare e parallelamente escludere dalla descrizione quelli che non si vogliono rendere accessibili dall'esterno.

In sostanza GARLIC presenta all'utente ed alle loro applicazioni i benefici dei database caratterizzati da un preciso schema (simile a quello offerto dai DBMS ad oggetti o object_relational) ma senza replicazione fisica. Internamente le differenze da un DBMS tradizionale sono invece più evidenti: il sistema riunisce informazioni provenienti da più sorgenti e per fonderle utilizza due strumenti:

- il modello dei dati orientato agli oggetti
- la memorizzazione degli oggetti complessi[1].

2.1 Il linguaggio GDL

Tra i vari compiti dei wrapper vi è quello di fornire una descrizione del contenuto della sorgente da loro servita, utilizzando il **G**arlic **D**ata **L**anguage, o **GDL**. GDL è un variante dell'ODMG[2] **O**bject **D**escription **L**anguage [Cat96]: attraverso le interface, ed un forte sistema dei tipi, si possono descrivere gli oggetti ed il loro comportamento, e memorizzare la loro descrizione in un *repository schema*. I vari *repository* sono quindi registrati come parti di un GARLIC Database, e *fusi* nello schema globale presentato all'utente. Un esempio di GDL è riportato in Figura 6.

L'esempio considera una semplice applicazione per una agenzia di viaggio: l'agenzia gestisce informazioni sugli stati e sulle città per le quali organizza viaggi (in un db relazionale), nonché un sito web per la prenotazione di hotels, ed un image server che raccoglie immagini pubblicitarie. La base di dati relazionali ha due sole tabelle: `Country` e `City`. La tabella `Country` mantiene

[1]Gli Oggetti Complessi servono nell'integrazione dei dati multimediali con quelli tradizionali, per aggiungere metodi che implementino nuovi comportamenti realizzabili grazie alla visione completa delle informazioni.

[2]Le differenze dallo standard ODL sono dovute alla necessità di esprimere situazioni non presenti in ambiente centralizzato.

le informazioni sugli stati, ed ha come chiave l'attributo `name`. La tabella `City` invece possiede sia una chiave, `name`, sia una foreign key, `country`: è interessante vedere come sia compito del wrapper individuare le foreign key nello schema originale (in questo caso `country`), e riportarle in GDL come se fossero attributi con dominio complesso (il dominio di `country` diventa così la tabella `Country`), facendone quindi una traduzione da relazionale a visione orientata agli oggetti (e questo è sicuramente un punto a favore di GARLIC). Più improbabile è invece la soluzione adottata per l'attributo `Scene`, che viene messo in relazione con la classe `Image`: ipotizzando infatti di sapere a priori che questo attributo si riferisce ad una tabella di un altro sistema, non dovrebbe comunque essere tra i compiti del wrapper il provvedere al *linking* di classi appartenenti a schemi di sorgenti diverse (il wrapper dovrebbe infatti occuparsi, ed essere a conoscenza, solo delle informazioni strettamente relative alla sorgente che gestisce, mentre dovrebbe essere un modulo di livello superiore a provvedere all'integrazione). A supporto di questa visione, viene la soluzione adottata nella descrizione della sorgente Web per le prenotazioni di hotels: l'attributo `city`, che andrebbe logicamente collegato alla tabella `City`, in realtà insiste su un dominio di tipo stringa. Questo perché, nell'esempio, si suppone che il sito web sia al di fuori del diretto controllo dell'agenzia, a differerenza della base di dati relazionale e dell'image server. L'eventuale integrazione del sito con le altre sorgenti è quindi (giustamente) demandata alla creazione di un *oggetto complesso*, ad un livello superiore. In particolare, in GARLIC, con *oggetto complesso* si definisce una *vista* il cui obiettivo è arricchire (estendendo, semplificando o deformando) uno o più oggetti appartenenti a schemi locali. Questi oggetti complessi sono definiti in modo dichiarativo (allo stesso modo con cui in SQL è possibile definire una vista basata su altre tabelle) utilizzando il linguaggio di interrogazione interno di GARLIC (si tratta di un'estensione orientata agli oggetti dello **S**tandard **Q**uery **L**anguage) e sono visti dall'utente, e dalle applicazioni, come oggetti veri e propri.
Interessante è anche l'uso della parola chiave `readonly`, che permette di discriminare tra fonti aggiornabili e fonti da cui si può esclusivamente estrarre dati, anche se poi non viene spiegato alcun meccanismo di propagazione degli update.

2.2 Query Planning

La fase di query planning porta, da una interrogazione posta sullo schema unificato, alla definizione di un insieme di query che le sorgenti locali devono eseguire. Parte attiva in questo processo la hanno i wrapper: le loro conoscenze sono utilizzate per formulare differenti piani di accesso, e per determinare il più efficiente tra questi. Durante la fase di pianificazione della query l'ottimizzatore di GARLIC identifica il frammento maggiore possibile che coinvolge una particolare sorgente, e lo spedisce al corrispondente wrapper: questo determina zero o più piani di accesso che realizzino, in toto o in parte, la query a lui assegnata. A questo punto l'ottimizzatore memorizza tutti i piani di tutti i wrapper interrogati, ed eventualmente aggiunge le operazioni da effettuare nel caso in cui una parte della query originale non sia eseguibile da alcun wrapper. È infatti da sottolineare come GARLIC sia in grado di gestire pure sorgenti con particolari restrizioni: oltre ad ipotizzare che una sorgente non sia in grado di effettuare, ad esempio, join a più vie, il wrapper può essere a conoscenza di particolarissime limitazioni sulle operazioni realizzabili, come possono essere la lunghezza massima di una stringa, il valore massimo di una costante in una interrogazione,ecc ... Il vantaggio consiste nel fatto di non dover comunicare tutte le restrizioni all'ottimizzatore, bensì di incapsularle a livello di wrapper. A sua volta l'ottimizzatore, in grado di realizzare funzioni tipiche di un DBMS, potrà effettuare quelle operazioni scartate dalle sorgenti. Oltre a questo, la possibilità di limitare le funzioni di risposta di una sorgente agevola notevolmente la fase di sviluppo di un wrapper: in un primo momento si possono realizzare solo le funzioni più semplici (rendendo utilizzabili da subito le sue informazioni), rimandando ad un secondo momento lo sviluppo di funzioni di ricerca più avanzate.

2.3 Pregi e difetti di GARLIC

Nonostante GARLIC sia da considerare la versione commerciale di TSIMMIS (sono entrambi stati sviluppati in ambienti IBM), l'intera impostazione del progetto è stata modificata. La differenza maggiore è senza dubbio l'abbandono del modello OEM (vedi Sezione 1.1), a cui è stato preferito l'utilizzo degli schemi locali (descritti attraverso il modello GDL): a fronte di una perdita di flessibilità dell'intero sistema (è ora praticamente impossibile gestire anche sorgenti semi-strutturate), l'intera architettura risulta semplificata, così come pure è semplificata la fase di integrazione degli schemi. Purtroppo però non sono stati fatti passi avanti nella automazione della fase di integrazione: l'utente, o l'applicazione, deve definirsi una visione ad-hoc di tutti gli schemi, o deve considerarne semplicemente l'unione (con tutte le duplicazioni di informazioni che ne conseguono). Molto approfondita è invece la fase di query planning, in senso di tradizionali operazioni di ottimizzazione dei costi di accesso ai DB, che non si è potuto descrivere meglio in questi paragrafi per motivi di spazio, ma a cui si rimanda negli articoli in bibliografia [MHS$^+$94, RS95].
Sostanzialmente diverse da progetti analoghi sono comunque le funzioni del wrapper: da semplice traduttore di linguaggi e protocolli, in GARLIC il wrapper include molte delle funzioni demandate in altri sistemi al mediatore vero e proprio. Se dal punto di vista architetturale questo potrebbe anche essere considerato un errore, risulta sorprendente (e forse poco credibile) la sostanziale differenza dei tempi di sviluppo dichiarata da TSIMMIS e da GARLIC: mentre in TSIMMIS si considera ragionevole impiegare circa 6 mesi per realizzare un semplice wrapper, nel progetto GARLIC può essere sviluppato in poche settimane...

3 SIMS

SIMS [ACHK93, AKH96], sviluppato presso l'Università della Southern California, è un mediatore di informazioni che si occupa di fornire accesso e integrazione ad una molteplicità di sorgenti eterogenee. Il cuore del progetto, ed il suo punto di forza, è la sua dichiarata abilità nel ritrovare e trattare le informazioni in modo *intelligente*, ovvero utilizzando tecniche di intelligenza artificiale. SIMS si distingue infatti dai progetti precedentemente esposti, e ne migliora l'approccio all'interrogazione, per la sua abilità nell'ottimizzare la fase di query processing, essendo in grado di manipolare, attraverso tecniche di intelligenza artificiale, le descrizioni semantiche delle sorgenti. Rimane invece manuale la fase di integrazione delle informazioni (ovvero la costruzione dello schema globale a partire da quelli locali), nonostante sia stata automatizzata, attraverso il modulo LIM, la traduzione di tutti gli schemi locali dal modello originale al modello di conoscenza di SIMS, che si basa sulla logica descrittiva LOOM.

Il progetto si basa sulle seguenti idee di fondo:

- *Rappresentazione e Modellazione della conoscenza*: è usata per descrivere il dominio che accomuna le informazioni, come pure le strutture ed il contenuto delle sorgenti di informazioni stesse. Il modello di ogni sorgente deve indicare il modello dei dati da questa utilizzato, il linguaggio di interrogazione, la dimensione, e deve descrivere il contenuto di tutti i suoi campi usando la terminologia di un predefinito modello comune del dominio (denominato *domain model*, e che costituisce in pratica lo schema globale);

- *Pianificazione e Ricerca*: è utilizzata per costruire una sequenza di query dirette alle singole sorgenti, a partire dalla interrogazione dell'utente;
- *Riformulazione*: dopo aver determinato un insieme di piani di accesso alle sorgenti, SIMS identifica, applicando un modello dei costi ed un algoritmo di ottimizzazione semantica, il più efficiente tra questi. Questa ricerca del piano migliore è aiutata anche dal fatto di avere a disposizione gli schemi descrittivi, semanticamente ricchi, sia delle singole sorgenti, sia del modello globale.

3.0.1 Componenti di SIMS

Per adempiere a tutte le sue funzioni, e per utilizzare tecniche *intelligenti*, SIMS fa uso di una serie di strumenti, caratteristici dei sistemi KBMS:

1. **LOOM**: è il sistema di rappresentazione della conoscenza utilizzato per descrivere sia le fonti locali di informazioni, sia lo schema globale, sia la fonte di informazione rappresentata dalle risposte alle query già ottenute, e memorizzate, da una determinata applicazione. Si tratta di una logica descrittiva derivato dal KL-ONE (modello sviluppato da Brachman nel 1976 in [BS85]).
2. **LIM**: è un'interfaccia (**L**OOM **I**nterface **M**odule) per mediare tra LOOM e le basi di dati. In particolare provvede a tradurre le descrizioni degli schemi delle sorgenti in linguaggio LOOM, nonché a tradurre query dirette alle sorgenti da LOOM al linguaggio di interrogazione proprio della singola sorgente. Deve essere sviluppata ad-hoc per ogni interfaccia che andrà a servire, ma automatizza la fase di wrapping degli schemi locali.
3. **Prodigy**: È il modulo che risolve i problemi di selezione delle sorgenti e pianificazione delle interrogazioni: partendo da una query sullo schema globale, utilizzando una serie di operatori da applicare alla query ed alla conoscenza memorizzata, ottiene uno stato finale caratterizzato dalle risposte che soddisfano la query.

3.1 Integrazione delle sorgenti

Poichè SIMS fa uso di un approccio "semantico", è assolutamente necessario che possieda le descrizioni dettagliate di tutte le fonti informative, creandone un *modello*. Per ogni singola sorgente, il *modello* deve contenere le seguenti informazioni:

- descrivere il *contenuto* informativo della sorgente;
- specificare se si tratta di una sorgente "classica"(nel qual caso si dovrà utilizzare l'interfaccia LIM) o di una sorgente di conoscenza LOOM;
- descrivere le dimensioni della base di dati e delle sue tabelle, nonché la loro locazione, per poter stimare il costo di un determinato piano di accesso;

```
(retrieve (?depth)
          (:and (port ?port)
                (port.name ?port ''SAN-DIEGO'')
                (port.depth ?port ?depth)
```

Figura 7: Esempio di query SIMS

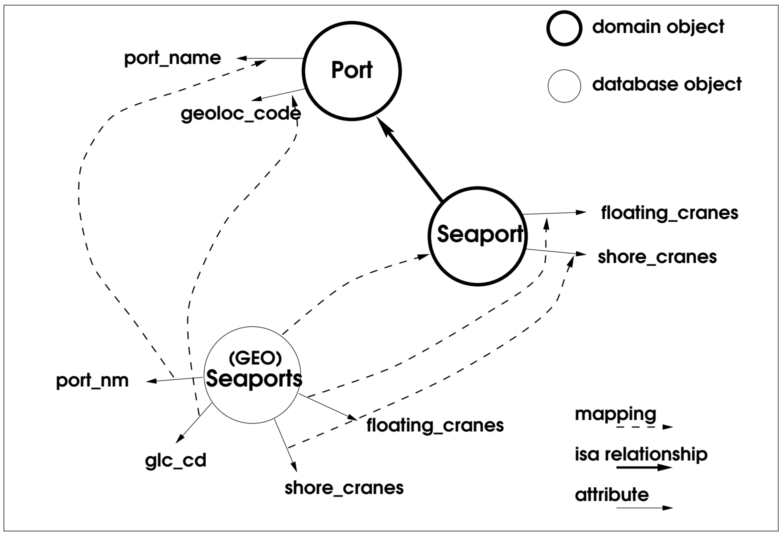

Figura 8: Mapping tra *domain model* e modello locale

- definire le chiavi delle tabelle, se esistono.

In aggiunta ai modelli delle sorgenti, viene definito un modello del dominio applicativo, definito *domain model*, che costituirà lo schema globale, l'unico col quale l'utente dovrà interagire. Questo è costituito da una base di conoscenza terminologica organizzata in modo gerarchico (attraverso il linguaggio LOOM), dove i nodi rappresentano tutti gli oggetti, le azioni, gli stati, possibili all'interno del dominio.

Le entità del dominio non devono però necessariamente corrispondere a classi appartenenti ad una determinata sorgente: il *domain model* deve essere inteso come la descrizione del dominio applicativo dal punto di vista dell'utente, e solo con esso l'utente avrà a che fare. In particolare, per porre una query, l'utente compone uno statement LOOM (un esempio di interrogazione è riportato in Figura 7, che richiede la profondità del porto di SAN-DIEGO) usando solo la terminologia del *domain model*, essendo in questo modo esonerato dal dover conoscere tutte le sorgenti integrate nel sistema (benché possa pure interagire direttamente con alcune di esse, se ha particolare familiarità con i loro termini).

La parte critica dell'intero processo è invece la fase di integrazione delle sorgenti, ovvero il collegare lo schema globale alle varie sorgenti locali. Questo consiste nel descrivere tutti i concetti e le relazioni di una sorgente attraverso i termini del *domain model*. Questo mapping deve essere fatto per tutte le sorgenti, ed in modo particolarmente accurato: un anello di mapping tra un concetto globale ed uno locale significa che i due concetti rappresentano la stessa classe di informazioni (ed analogamente per gli attributi). Un esempio qualitativo di questo mapping è riportato in Figura 8: se l'utente richiede tutti gli elementi della classe globale `Seaport`, andrà interrogata la classe `Seaports` della sorgente `GEO`.

3.2 Query Processing

L'intero processo che porta dalla formulazione di una query da parte dell'utente, posta sullo schema globale, alla presentazione dei risultati, è rappresentato in Figura 9.

Selezione delle fonti informative Il primo passaggio da realizzare, una volta in possesso dell'interrogazione dell'utente formulata usando la terminologia dello schema globale, è identifi-

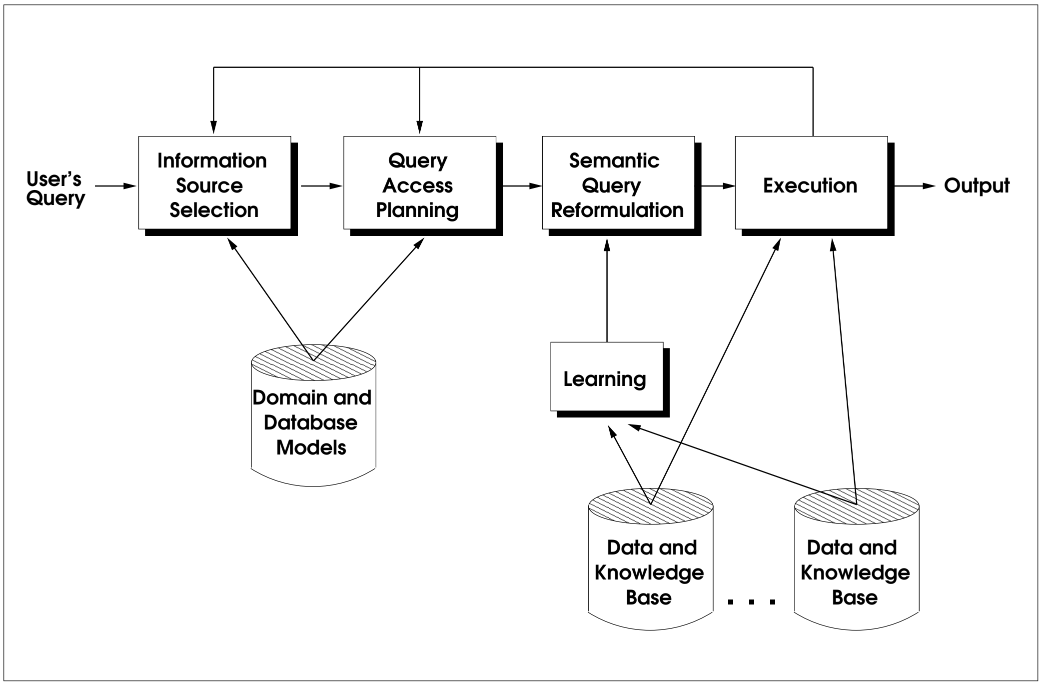

Figura 9: Query Processing

care le appropriate sorgenti che saranno interessate dalla query. Per esempio, se l'utente richiede informazioni sui `porti` ed esiste in una base di dati un concetto che contiene i porti, il mapping è facilissimo, come pure la riformulazione della query. In realtà, molto spesso non esisterà un mapping diretto e sarà necessario riformulare la query utilizzando termini propri delle sorgenti locali. A questo scopo, vengono utilizzati una serie di operatori di riformulazione, diretti a definire una query equivalente alla originale. Tra di essi, operatori di generalizzazione (che fanno uso delle relazioni tra classe e super-classe e spostano ad un livello superiore dell'albero gerarchico la query, magari aggiungendovi delle limitazioni ai domini degli attributi in modo da ottenere ancora una query equivalente), operatori di specializzazione (che attraverso tecniche di intelligenza artificiale cercano di riclassificare la query ad un livello gerarchico inferiore, in modo da determinare l'insieme di classi da interrogare), operatori di partizione.

Piano di accesso Il piano di accesso determina un ordine per le query trovate nella fase precedente, cercando di massimizzare il parallelismo delle operazioni. Per fare questo sono analizzati i costi di accesso alle sorgenti ed i costi di eventuali passaggi aggiuntivi che il sistema dovrà realizzare per produrre i risultati finali. L'ordine di esecuzione è determinato esaminando quali passi del piano di accesso si basano su risultati raccolti in altre sorgenti, facendo uso del modulo Prodigy, che per questa attività presenta funzioni analoghe a quelle di un ottimizzatore DBMS.

Query Plan Reformulation Oltre ad una ottimizzazione basata sul modello dei costi di accesso, SIMS è pure in grado di realizzare una query reformulation di tipo semantico. L'idea di base è trasformare la query risultante dal piano di accesso in una query semanticamente equivalente che può essere eseguita in maniera più efficiente. Questa ottimizzazione è realizzata sia a livello di singola sorgente, sia a livello globale. Nella singola sorgente, il problema della riformulazione è analogo al problema dell'ottimizzazione semantica di una query all'interno di una base di dati. La riformulazione si basa quindi su un processo di inferenza che utilizza delle informazioni estratte dal database e codificate attraverso l'uso di *rule* e di limitazioni sui range dei domini. Analogamente viene effettuata una ottimizzazione della query globale, in modo da realizzare efficientemente la fase di processing dei risultati parziali ottenuti dalle singole sorgenti.

3.3 Pregi e difetti di SIMS

Non sono pochi gli aspetti interessanti ed innovativi che caratterizzano questo progetto, dovuti fondamentalmente al massiccio uso di tecniche di Intelligenza Artificiale, permettendo di sfruttare tutti i pregi di un approccio semantico. Queste tecniche sono utilizzate sia in fase di identificazione delle sorgenti (in assenza di mapping diretti si determinano a run time le sorgenti che sono interessate dalla query), sia in fase di ottimizzazione della query stessa.
Altro aspetto da sottolineare, assente in progetti analoghi, è l'utilizzo di una base di dati (o meglio, di conoscenza) interna al mediatore stesso sia per riprocessare le risposte delle sorgenti (ma questo è un aspetto già incontrato), sia per memorizzare le risposte già ottenute, utilizzandole successivamente, in parte o in toto, come sorgente aggiuntiva (evitando in questo modo di andare a recuperare dati che sono presenti nella memoria del sistema).

Rimane intentata, come negli altri progetti, l'automazione della fase di integrazione degli schemi locali, mentre risulta possibile usufruire della conoscenza semantica degli schemi sorgenti, cosa che negli altri approcci è stata trascurata.

Riferimenti bibliografici

[ACHK93] Y. Arens, C.Y. Chee, C. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[AKH96] Y. Arens, C. A. Knoblock, and C. Hsu. Query processing in the sims information mediator. *Advanced Planning Technology*, 1996.

[BS85] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Cat96] R. Cattel, editor. *The Object Database Standard: ODMG-93.* Morgan Kaufmann, 1996.

[DAY$^+$96] D.Quass, A.Rajaraman, Y.Sagiv, J.Ullman, and J.Widom. Querying semistructured heterogeneous informations. Technical report, Stanford University, 1996.

[GMea95] H. Garcia-Molina and et al. The tsimmis approach to mediation: Data models and languages. In *NGITS workshop*, 1995. Available ftp://db.stanford.edu/pub/garcia/1995/tsimmis-models-languages.ps.

[HMJ95] Y.Papakonstantinou H.Garcia-Molina and J.Ullman. Medmaker: a mediation system based on declarative specification. Technical report, Stanford University, 1995. ftp://db.stanford.edu/pub/papakonstantinou/1995/medmaker.ps.

[LFK$^+$] Alon Levy, Dana Florescu, Jaewoo Kang, Anand Rajaraman, and Joanne J. Ordille. The information manifold project. Available at http://www.research.att.com/ levy/imhome.html.

[MHS+94] M.J.Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Object exchange across heterogeneous information sources. Technical report, Stanford University, 1994.

[MR96] Oliver M.Duschka and Micheal R.Genesereth. Infomaster - an information integration toolkit. Technical report, Department of Computer Science. Stanford University, 1996.

[PS97] Daniel P.Miranker and Vasilis Samoladas. Alamo: an architecture for integrating heterogenous data sources. In *Proceedings of the 4th KRDB Workshop*, Athens, Greece, August 1997.

[RS95] M.T. Roth and P. Scharz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proc. of the 23rd Int. Conf. on Very Large Databases*, Athens, Greece, March 1995.

[SAB+] V.S. Subrahmanian, Sibel Adali, Anne Brink, J.ames, J. Lu, Adil Rajput, Timothy J. Rogers, Robert Ross, and Charles Ward. Hermes: A heterogeneous reasoning and mediator system. Available at http://www.cs.umd.edu/projects/hermes/overview/paper/index.html.

[YAGM96] Y.Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB Int. Conf.*, Bombay, India, September 1996.

[YHMJ94] Y.Papakonstantinou, H.Garcia-Molina, and J.Widom. Object exchange across heterogeneous information sources. Technical report, Stanford University, 1994.