InterData

Programma di ricerca (cofinanziato dal MURST, esercizio 1997)

Metodologie e tecnologie per la gestione di dati e processi su reti Internet e Intranet

La gestione delle interrogazioni nel sistema MOMIS

Sonia Bergamaschi, Domenico Beneventano, Maurizio Vincini

T3-RXX **20 settembre 1999**

Sommario

Il sistema MOMIS (Mediator envirOnment for Multiple Information Sources) [4, 5, 6, 7], è basato sulla proposta I^3 (Intelligent Information of Integration) [14] ed è rivolto all'integrazione di sorgenti di dati testuali strutturati e semistrutturati. L'architettura complessiva del sistema MOMIS è stata descritta nel Rapporto Tecnico Interdata **T3-R08** [9]; nel presente Rapporto viene descritta in maniera dettagliata la gestione delle interrogazioni.

Tema	Tema 3: Integrazione di informazioni estratte da siti WWW
Codice	T3-RXX
Data	20 settembre 1999
Tipo di prodotto	Rapporto tecnico
Unità responsabile	MO
Unità coinvolte	MO
Autori	Sonia Bergamaschi, Domenico Beneventano, 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

La gestione delle interrogazioni nel sistema MOMIS

Sonia Bergamaschi   Domenico Beneventano   Maurizio Vincini

Sommario

Il sistema MOMIS (Mediator envirOnment for Multiple Information Sources) [4, 5, 6, 7], è basato sulla proposta I^3 (Intelligent Information of Integration) [14] ed è rivolto all'integrazione di sorgenti di dati testuali strutturati e semistrutturati. L'architettura complessiva del sistema MOMIS è stata descritta nel Rapporto Tecnico Interdata **T3-R08** [9]; nel presente Rapporto viene descritta in maniera dettagliata la gestione delle interrogazioni.

A tale scopo verranno descritti due moduli del mediatore del sistema MOMIS. Il primo, Extensional Hierarchy Builder (EHB), serve per individuare e elaborare le relazioni estensionali tra classi locali. I risultati prodotti consentono un notevole miglioramento nell'efficienza e nella qualità delle risposte prodotte dal sistema. Il secondo, Query Manager (QM), è il modulo di gestione delle interrogazioni. In particolare, genera le query in linguaggio OQL_{I^3} da inviare ai wrapper partendo dalla singola query formulata dall'utente sullo schema globale. Servendosi delle tecniche di Logica Descrittiva, il QM genera automaticamente la traduzione della query sottomessa nelle corrispondenti sub-query delle singole sorgenti.

1 Introduzione

L'architettura complessiva del sistema MOMIS è stata descritta nel Rapporto Tecnico Interdata **T3-R08** [9]; nel seguito essa viene brevemente ripresa allo scopo di inquadrare il ruolo del gestore delle interrogazioni.

1.1 L'architettura di MOMIS

Come mostrato in Fig. 1, l'architettura di MOMIS, basata sullo standard I^3 [14], comprende quattro componenti principali:

1. *Wrapper*: posti al di sopra di ciascuna sorgente, sono i moduli che rappresentano l'interfaccia tra il mediatore vero e proprio e le sorgenti locali di dati. La loro funzione è duplice: **(1)** in fase di integrazione, forniscono la descrizione in linguaggio ODL_{I^3} delle informazioni in essa contenute; **(2)** in fase di query processing, traducono la query ricevuta dal mediatore (espressa quindi nel linguaggio comune di interrogazione OQL_{I^3}, che è definito in analogia al linguaggio OQL) in una interrogazione comprensibile (e realizzabile) dalla sorgente stessa. Devono inoltre esportare i dati ricevuti in risposta all'interrogazione, presentandoli al mediatore attraverso il modello comune di dati utilizzato dal sistema;

2. *mediatore*: è il cuore del sistema, in grado di fornire una rappresentazione omogenea delle informazioni ed un accesso integrato alle sorgenti. Il Mediatore è composto da tre moduli distinti:

   - Global Schema Builder (GSB): è il modulo di integrazione degli schemi locali, che, partendo dalle descrizioni delle sorgenti espresse attraverso il linguaggio ODL_{I^3}, genera un unico schema globale da presentare all'utente.

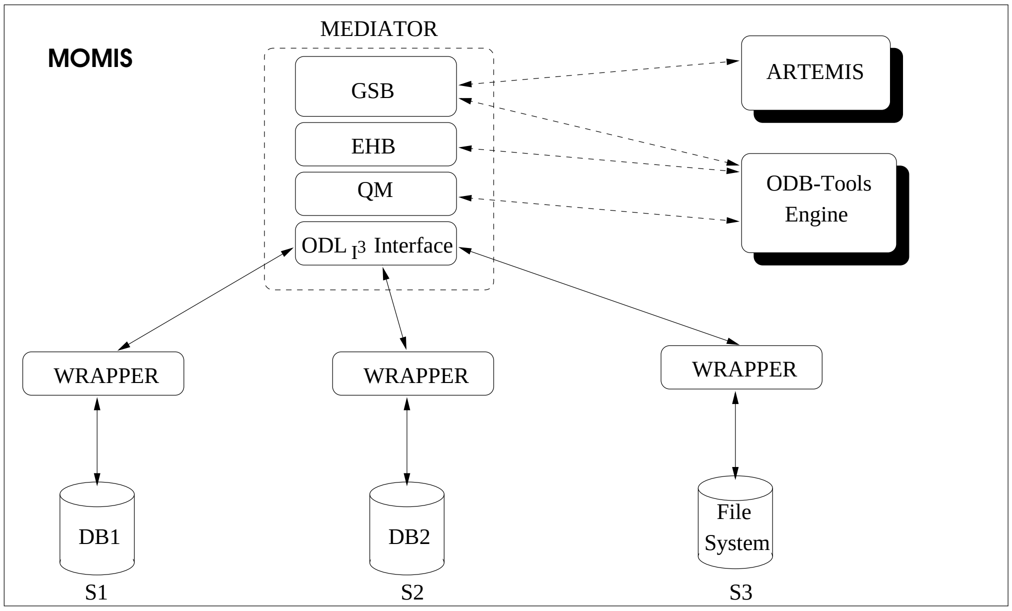

Figura 1: Architettura del sistema MOMIS

- Extensional Hierarchy Builder (EHB): questo modulo è stato progettato per individuare e elaborare le relazioni estensionali tra classi locali. I risultati prodotti consentono un notevole miglioramento nell'efficienza e nella qualità delle risposte prodotte dal Query Manager.
- Query Manager (QM): è il modulo di gestione delle interrogazioni. In particolare, genera le query in linguaggio OQL_{I^3} da inviare ai wrapper partendo dalla singola query formulata dall'utente sullo schema globale. Servendosi delle tecniche di Logica Descrittiva, il QM genera automaticamente la traduzione della query sottomessa nelle corrispondenti sub-query delle singole sorgenti.

3. *ODB-Tools Engine*, un tool basato sulla OLCD Description Logics [1, 8] che compie la validazione di schemi e l'ottimizzazione di query [3, 2].
4. *ARTEMIS-Tool Environment*, un tool che compie analisi e clustering di schemi [11].

Mediante questa struttura e queste funzionalità MOMIS è quindi in grado di fornire un accesso integrato ad informazioni eterogenee memorizzate sia in database di tipo tradizionale (e.g. relazionali, object-oriented) o file system, sia in sorgenti di tipo semistrutturato.

In questo Rapporto Tecnico verranno descritte le componenti EHB e QM del mediatore; l'altra componente, GSB, é già stata descritta nel precedente Rapporto Tecnico **T3-R08** [9] e le sue funzionalità verranno brevemente richiamate nella prossima sezione, tramite l'introduzione di un esempio di riferimento.

1.2 Esempio di riferimento

L'esempio si riferisce ad una realtà universitaria: le sorgenti da integrare sono tre. La prima sorgente, `University` (SU), è un database di tipo relazionale, che contiene informazioni sullo staff e sugli studenti di una determinata università. È composta da cinque tabelle: `Research_Staff`, `School_Member`, `Department`, `Section` e `Room`. Per ogni professore (presente

Sorgente University (SU)

```
Research_Staff(first_name,last_name,relation,email,
               dept_code,section_code)
School_Member(first_name,last_name,faculty,year)
Department(dept_name,dept_code,budget,dept_area)
Section(section_name,section_code,length,room_code)
Room(room_code,seats_number,notes)
```

Sorgente Computer_Science (SCS)

```
CS_Person(name)
Professor:CS_Person(title,belongs_to:Division,rank)
Student:CS_Person(year,takes:set⟨Course⟩,rank)
Division(description,address:Location,fund,sector,employee_nr)
Location(city,street,number,county)
Course(course_name,taught_by:Professor)
```

Sorgente Tax_Position (STP)

```
University_Student(name,student_code,faculty_name,tax_fee)
```

Figura 2: Esempio di riferimento

nella tabella `Research_Staff`), sono memorizzate informazioni sul suo dipartimento (attraverso la foreign key `dept_code`), sul suo indirizzo di posta elettronica (`email`), e sul corso da lui tenuto (`section_code`). Per il corso, viene memorizzata pure l'aula (`Room`) dove questo si svolge, mentre del dipartimento sono descritti, oltre al nome (`dept_name`) ed al codice (`dept_code`), il budget (`budget`) che ha a disposizione e l'area (`dept_area`) a cui appartiene, sia essa Scientifica, Economica,... Per gli studenti presenti nella tabella `School_Member` sono invece mantenuti il nome (nella coppia `first_name` e `last_name`), la facoltà di appartenenza (`faculty`) e l'anno di corso (`year`).
La sorgente `Computer_Science` (SCS) contiene invece informazioni sulle persone afferenti a questa facoltà, ed è un database ad oggetti. Sono presenti sei classi: `CS_Person`, `Professor`, `Student`, `Division`, `Location` e `Course`. I dati mantenuti sono comunque abbastanza simili a quelli della sorgente SU: per quanto riguarda i professori, sono memorizzati il titolo (`title`), e la divisione di appartenenza (`belongs_to`), che a sua volta fa parte di un dipartimento (e ne può quindi essere considerata una specializzazione); per gli studenti sono memorizzati i corsi seguiti (`takes`) e l'anno di corso (`year`). Il corso ha poi un attributo complesso che lo lega al professore che ne è titolare (`taught_by`), mentre per la divisione si tiene l'indirizzo (`address`), i fondi (`fund`) e il numero di impiegati (`employee_nr`).
È presente inoltre una terza sorgente, `Tax_Position` (STP), facente capo alla segreteria studenti, che mantiene i dati relativi alle tasse da pagare (`tax_fee`). In questo caso (STP), non si tratta di un database ma di un file system, che contiene quindi semplici tracciati record.

1.3 Integrazione intensionale di schemi

L'integrazione delle sorgenti informative strutturate e semistrutturate viene compiuta in modo semi-automatico, utilizzando le descrizioni degli schemi locali in linguaggio ODL$_{I^3}$ e combinando le tecniche di logica descrittiva e di clustering. Le attivitá compiute sono le seguenti:

1. *Estrazione di relazioni terminologiche*
2. *Affinity-based clustering di classi ODL$_{I^3}$*
3. *Costruzione dello schema globale di mediatore*

1.3.1 Estrazione di relazioni terminologiche

Durante questo passo viene costruito un Thesaurus Comune di *relazioni terminologiche*. Le relazioni terminologiche esprimono la conoscenza di inter-schema su sorgenti diverse e corrispondono alle asserzioni intensionali utilizzate in [12]. Le relazioni terminologiche sono derivate in modo semi-automatico a partire dalle descrizioni degli schemi in ODL_{I^3}, attraverso l'analisi strutturale e di contesto delle classi coinvolte, utilizzando ODB-Tools e le tecniche di logica descrittiva.

Le relazioni che si possono definire all'interno del Thesaurus sono le seguenti: SYN (synonym-of), BT (broader-term), NT (narrower-term) e RT (related-term). L'intero processo che porta, partendo dalle descrizioni degli schemi in ODL_{I^3}, alla definizione del un Thesaurus comune si articola in quattro passi.

1. **Estrazione automatica di relazioni dagli schemi sorgenti.** Sfruttando le informazioni semantiche presenti negli schemi strutturati può essere identificato in modo automatico un insieme di relazioni terminologiche.

   Nel nostro esempio, le relazioni terminologiche automaticamente estratte sono le seguenti:

   ⟨`SCS.Professor` NT `SCS.CS_Person`⟩
   ⟨`SCS.Student` NT `SCS.CS_Person`⟩
   ⟨`SCS.Professor` RT `SCS.Division`⟩
   ⟨`SCS.Student` RT `SCS.Course`⟩
   ⟨`SCS.Division` RT `SCS.Location`⟩
   ⟨`SCS.Course` RT `SCS.Professor`⟩
   ⟨`SU.Research_Staff` RT `SU.Department`⟩
   ⟨`SU.Research_Staff` RT `SU.Section`⟩
   ⟨`SU.Section` RT `SU.Room`⟩

2. **Revisione/Integrazione delle relazioni.** Interagendo con il modulo, il progettista deve inserire tutte le relazioni terminologiche che non sono state estratte nel passo precedente, ma che devono comunque essere presenti per pervenire ad una esatta integrazione delle sorgenti.

   Per esempio, supponiamo che una relazione di SYN sia definita tra due classi aventi la stessa struttura (per esempio `SU.Section` e `SCS.Course`). Per rendere compatibile questa relazione terminologica con una relazione di equivalenza per ODB-Tools é necessario uniformare la descrizione di entrambe le classi. Il risultato della modifica per le classi esaminate diventa:

```
interface Section
( ... )
{ attribute string  section_name;
  attribute integer section_code;
  attribute integer length;
  attribute integer room_code;
  attribute string  course_name
  attribute Professor taught_by };

interface Course
( ... )
{ attribute string  course_name
  attribute Professor taught_by;
```

```
  attribute string  section_name;
  attribute integer section_code;
  attribute integer length;
  attribute integer room_code };
```

Lo stesso tipo di problematica si evidenzia per la relazione di BT tra `SU.Department` e `SCS.Division`. Tradurre questa relazione terminologica nella corrispondente relazione di subsumption implica la modifica delle descrizioni nel seguente modo:

```
interface Department
(...)
{ attribute string  dept_name;
  attribute integer budget;
  attribute integer dept_code;
  attribute string  dept_area };

interface Division
( ...  )
{ attribute string   dept_name;
  attribute integer  budget;
  attribute integer  dept_code;
  attribute string   dept_area;
  attribute string   description;
  attribute Location address;
  attribute integer  found;
  attribute integer  employee_nr;
  attribute string   sector };
```

Infine quando viene asserita una nuova relazione di RT verrà inserito, nella classe alla destra della relazione, un nuovo attributo di aggregazione che mappa sulla prima classe.

Queste trasformazioni conducono alla generazione automatica di uno "schema virtuale" contenente, per ciascuno schema locale, le descrizioni modificate delle classi; tale schema viene utilizzato da ODB-Tools per eseguire ulteriori inferenze di relazioni per arricchire il Thesaurus.

Nel nostro esempio, supponiamo che il progettista inserisca le seguenti relazioni terminologiche relative sia a classi che ad attributi:

⟨SU.Research_Staff BT SCS.Professor⟩
⟨SU.School_Member BT SCS.Student⟩
⟨STP.University_Student BT SCS.Student⟩
⟨SU.Department BT SCS.Division⟩
⟨SU.Section SYN SCS.Course⟩
⟨name BT first_name⟩
⟨name BT last_name⟩
⟨dept_code BT belongs_to⟩
⟨dept_name SYN description⟩
⟨section_name SYN course_name⟩
⟨faculty SYN faculty_name⟩
⟨fund SYN budget⟩
⟨dept_area SYN sector⟩

3. **Validazione delle relazioni.** In questa fase, ODB-Tools viene utilizzato per validare le relazioni terminologiche del Thesaurus definite tra due attributi. La validazione è basata sul controllo di compatibilità dei domini associati agli attributi. In questo modo le relazioni terminologiche vengono distinte in *valide* e *invalide*. In particolare, dati i due attributi $a_t = \langle n_t, d_t \rangle$ e $a_q = \langle n_q, d_q \rangle$ coinvolti in una relazione, verranno eseguiti i seguenti controlli:

   - $\langle n_t$ SYN $n_q\rangle$: la relazione è considerata valida se d_t e d_q sono equivalenti, oppure uno è la specializzzazione dell'altro;
   - $\langle n_t$ BT $n_q\rangle$: la relazione è considerata valida se d_t contiene od è equivalente a d_q;
   - $\langle n_t$ NT $n_q\rangle$: la relazione è considerata valida se d_t è contenuto in, od è equivalente a d_q;

   Nel caso in cui il dominio dell'attributo contiene il connettore di unione, una relazione viene considerata valida quando almeno un dominio risulta compatibile. Questa fase di validazione è compiuta utilizzando ODB-Tools per il calcolo della relazione di subsumption e equivalence.

   Con riferimento al Thesaurus definito in precedenza, riportiamo l'output della fase di validazione: per ciascuna relazione il flag di controllo [1] indica una relazione valida, mentre il flag [0] una invalida.

⟨name BT first_name⟩	[1]
⟨name BT last_name⟩	[1]
⟨dept_code BT belongs_to⟩	[0]
⟨dept_name SYN description⟩	[1]
⟨section_name SYN course_name⟩	[1]
⟨faculty SYN faculty_name⟩	[1]
⟨fund SYN budget⟩	[1]
⟨dept_area SYN sector⟩	[1]

4. **Inferenza di nuove relazioni.** In questa fase vengono inferite nuove relazioni terminologiche utilizzando lo "schema virtuale" definito nella fase di revisione/integrazione e calcolando le nuove relazioni di subsumption e aggregazione. Mediante ODB-Toolsqueste nuove relazioni semantiche vengono tradotte nelle corrispondenti relazioni terminologiche arricchendo il Thesaurus che ora contiene sia le relazioni esplicite sia quelle inferite. Il risultato ottenuto viene chiamato Common Thesaurus.

   Rispetto al nostro esempio, le relazioni terminologiche inferite sono:

⟨SCS.CS_Person BT SU.Research_Staff⟩
⟨SCS.CS_Person BT SU.School_Member⟩
⟨SU.Section RT SCS.Professor⟩
⟨SCS.Professor RT SU.Department⟩
⟨SCS.Professor RT SU.Section⟩
⟨SCS.Course RT SU.Room⟩
⟨SCS.Student RT SU.Section⟩
⟨SCS.CS_Person BT STP.University_Student⟩

In Figura 3 sono invece riportate tutte le classi di tutte le sorgenti, così come risultano riorganizzate alla fine della fase di generazione del Thesaurus comune, facendo quindi riferimento alle classi modificate. È quindi una rappresentazione grafica delle relazioni che sussistono tra queste: le linee tratteggiate mettono in evidenza le relazioni inferite, le linee unite rappresentano invece quelle esplicite; le linee dotate di frecce rappresentano inoltre relazioni di generalizzazione, mentre quelle in cui le frecce sono assenti denotano le relazioni di aggregazione.

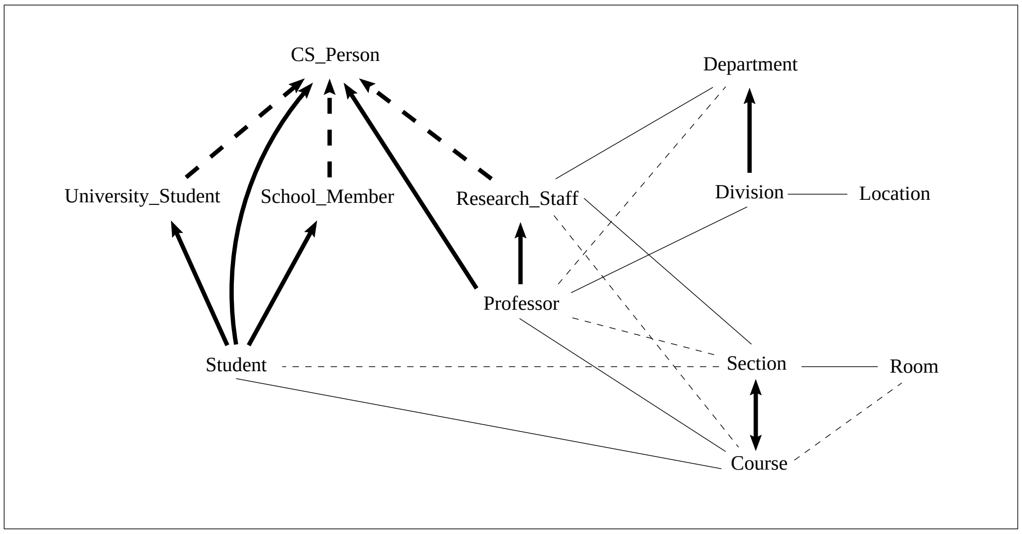

Figura 3: Thesaurus comune per le sorgenti SU, SCS, e STP.

1.3.2 Affinity-based clustering di classi $\mathbf{ODL}_{I^3}$

Con il supporto dell'ambiente ARTEMIS-Tool, le relazioni terminologiche contenute nel Thesaurus vengono utilizzate per valutare il livello di affinitá tra le classi ODL_{I^3} in modo di identificare le informazioni da essere integrate a livello globale. A tal fine, ARTEMIS calcola i coefficienti che misurano il livello di affinitá delle classi ODL_{I^3} basandosi sia sui nomi che sugli attributi. Le classi ODL_{I^3} con maggiore affinitá vengono raggruppate utilizzando le tecniche di clustering [13].

Per l'identificazione degli insiemi di classi ODL_{I^3} aventi elevata affinità negli schemi considerati sono utilizzate tecniche di clustering, attraverso le quali le classi sono automaticamente classificate in gruppi caratterizzati da differenti livelli di affinità, formando un albero [13].
La procedura di clustering adottata opera in modo iterativo andando a creare insiemi di classi, *cluster* appunto, di dimensioni via via crescenti. Ad ogni passo viene quindi costruito un nuovo cluster unendo quelli ottenuti al passo precedente ed aventi il valore massimo di affinità. Questo processo porta alla creazione di un *albero di affinità*, caratterizzato dall'avere come foglie le classi

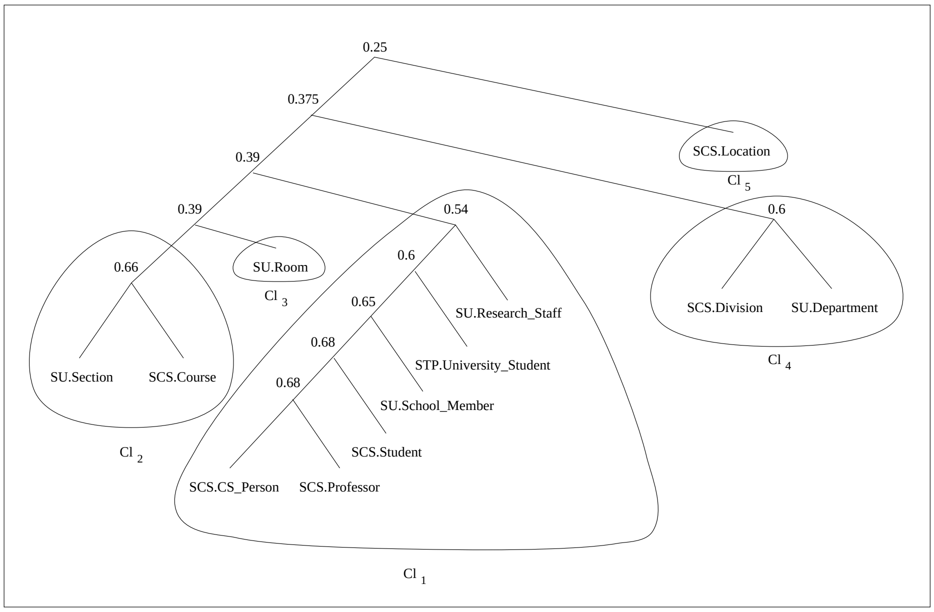

Figura 4: Albero di affinità

locali, come radice un l'insieme di tutti i *cluster*, ed i cui nodi sono proprio i *cluster* individuati (con valore di affinità calante spostandosi verso la radice).

Un esempio di come le classi vengano organizzate in un *Albero di Affinità* è riportato in Figura 4 in cui l'algoritmo di *Clustering* è stato applicato al nostro esempio di riferimento.

Dopo aver costruito l'albero di affinitá, il problema è quello di selezionare i cluster più appropriati per la definizione delle classi nello schema globale. La procedura di selezione dei cluster, in ARTEMIS, viene mantenuta interattiva attraverso la modifica del valore di soglia. Il progettista specifica il valore della soglia T ed i cluster caratterizzati da un valore di $GA()$ superiore o uguale a T sono selezionati e proposti. Per alti valori di T si ottengono cluster piccoli e molto omogenei tra loro. Diminuendo il valore di T, i cluster ottenuti contengono più classi e sono più eterogenei. Nel tool il valore di default di T viene posto pari a 0.5. Tale valore può essere modificato dinamicamente, sulla base della specifica applicazione in esame.

1.3.3 Costruzione dello schema globale di mediatore

Con il supporto di ODB-Tools i cluster di classi ODL$_{I^3}$ affini sono analizzate per costruire lo schema globale del Mediatore. Per ciascun cluster viene definita una classe globale ODL$_{I^3}$ che rappresenta tutte le classi che sono riferite al cluster, e che é caratterizzata dall'unione dei loro attributi. L'insieme delle classi globali definite costituisce lo schema globale di Mediatore che deve essere usato per porre le query alle sorgenti locali integrate. In questa fase OLCD e ODB-Tools sono utilizzati per realizzare una generazione semi-automatica delle classi globali.

La prima fase di questo processo viene realizzata automaticamente e genera, per ogni cluster, una *classe_globale$_i$* rappresentativa di tutte le classi che fanno parte del cluster (costituisce una visione unificata di queste classi locali).

Un esempio di definizione di un classe globale in linguaggio ODL$_{I^3}$ è mostrato in Fig. 5 per la classe `Hospital_Patient`. Come si può vedere da questa figura, per ciascun attributo sono definite regole di mapping, in modo da specificare sia informazioni sulla corrispondenza tra attributo globale ed il corrispondente nelle classi locali associate, sia su eventuali valori di default o

```
interface University_Person
(extent Research_Staffers, School_Members, CS_Person
       Professors, Students, University_Students
 key    name)
{ attribute string name
    mapping_rule  (University.Research_Staff.first_name and
                  University.Research_Staff.last_name)
                  (University.School_Member.first_name and
                  University.School_Member.last_name),
                  Computer_Science.CS_Person.name
                  Computer_Science.Professor.name
                  Computer_Science.Student.name
                  Tax_Position.University_Student.name;
  attribute string rank
    mapping_rule  University.Research_Staff = 'Professor',
                  University.School_Member = 'Student',
    ... }
```

Figura 5: Esempio di classe globale in ODL_{I^3}

di null (mancanza della corrispondenza). Per esempio, per l'attributo globale `name`, la regola di mapping specifica gli attributi che debbono essere considerati in ciascuna classe locale del cluster $\mathtt{Cl}_1$. Nel caso specifico, viene definita una *and* correspondence per la classe `SU.Research_Staff`. Una diversa mapping rule è definita per l'attributo globale `rank` per specificare il valore che deve essere ad esso associato per le istanze di `SU.Research_Staff` e `SU.School_Member`.

Come si può vedere, nella definizione di un attributo globale non viene specificato alcun dominio: questo significa che MOMIS accetta i domini definiti nelle sorgenti locali e, in risposta ad una query, sono visualizzati i dati nei formati specifici delle sorgenti locali.
In Figura 6 viene mostrato come la descrizione il linguaggio ODL_{I^3} della classe globale `Hospital_Patient` sia poi tradotta in una *Mapping Table* persistente.

University_Person	**name**	**rank**	**dept**	**faculty**	**...**
Research_Staff	first_name *and* last_name	'Professor'	dept_code	null	...
School_Member	first_name *and* last_name	'Student'	null	faculty	...
CS_Person	name	null	null	'Computer_Science'	...
Professor	name	rank	belongs_to	'Computer_Science'	...
Student	name	rank	null	'Computer_Science'	...
University_Student	name	'Student'	null	faculty_name	...

Workplace	**name**	**area**	**employee_nr**	**budget**	**...**
Department	dept_name	dept_area	null	budget	...
Division	description	sector	employee_nr	fund	...

Figura 6: Mapping table di University_Person e Workplace

In Figura 7 sono rappresentate le fasi del processo di integrazione estensionale evidenziando l'intervento del progettista da un lato e l'impiego di ODB-Tools dall'altro.

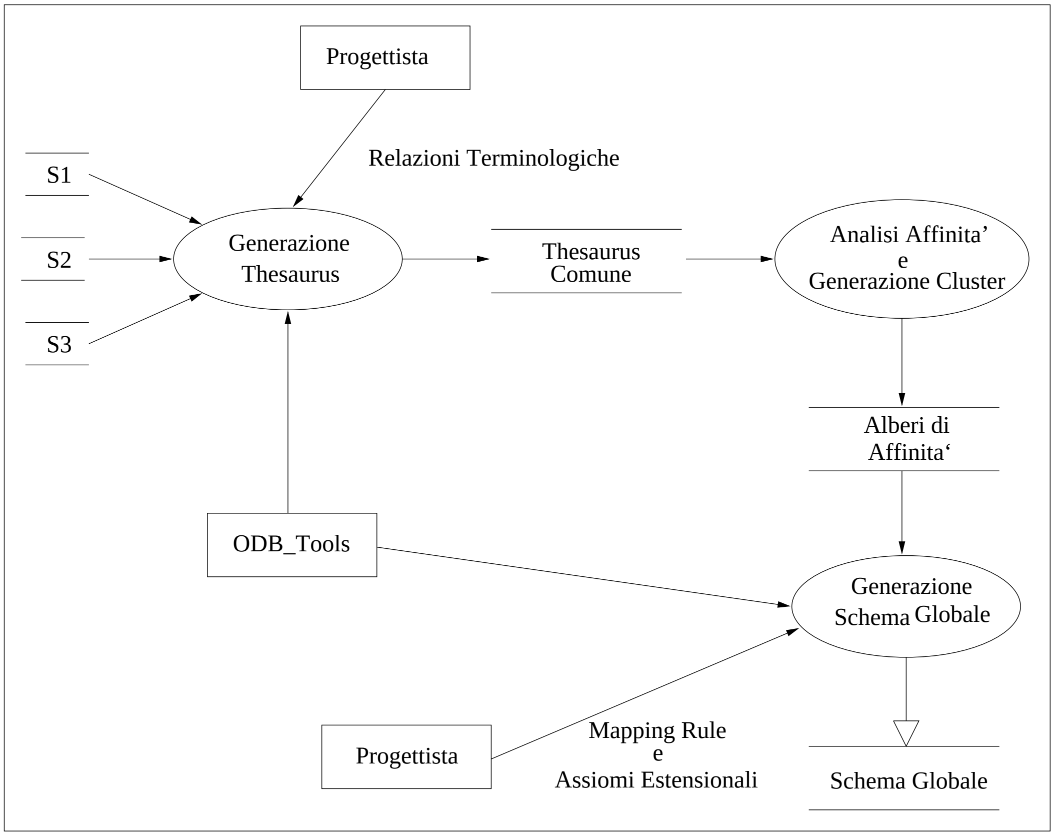

Figura 7: Fasi dell' Integrazione Intensionale

2 Integrazione estensionale di schemi

La gestione delle interrogazioni in un contesto distribuito ed eterogeneo richiede sia la capacità di reperire il giusto insieme di dati dalle sorgenti sia l'abilità nel combinarli in modo corretto. Perché un'interrogazione possa essere eseguita in un contesto eterogeneo, è necessario poter risolvere i conflitti intensionali e semantici ma occorre anche sapere come la conoscenza rappresentata nella vista globale sia effettivamente mappata sui dati presenti nelle sorgenti.
In questa sezione verranno illustrare le problematiche connesse alla gestione della conoscenza estensionale, mostrando come questo tipo di informazioni possano essere utilizzate per arricchire lo schema globale gestito dal Mediatore.

2.1 Le relazioni estensionali

Le attività descritte nella sezione precedente sono volte alla risoluzione dei conflitti intensionali. L'integrazione degli schemi non è però l'unico aspetto che occorre gestire per ottenere un'effettiva integrazione di sorgenti eterogenee, infatti, come descritto in [17, 16], è necessario risolvere anche i conflitti derivanti dalle sovrapposizioni delle estensioni, cioè dalla presenza, in sorgenti diverse, di informazioni relative alla stessa entità del "*mondo reale*".

Sorgenti autonome possono contenere *oggetti* corrispondenti alla stessa *entità* del dominio applicativo da modellare; in generale, ognuno di questi oggetti replicherà proprietà già presenti in altri oggetti e aggiungerà nuove proprietà. Pertanto l'obiettivo dell'integrazione deve essere non solo il reperimento dei singoli oggetti ma piuttosto la ricomposizione dell'entità a cui sono

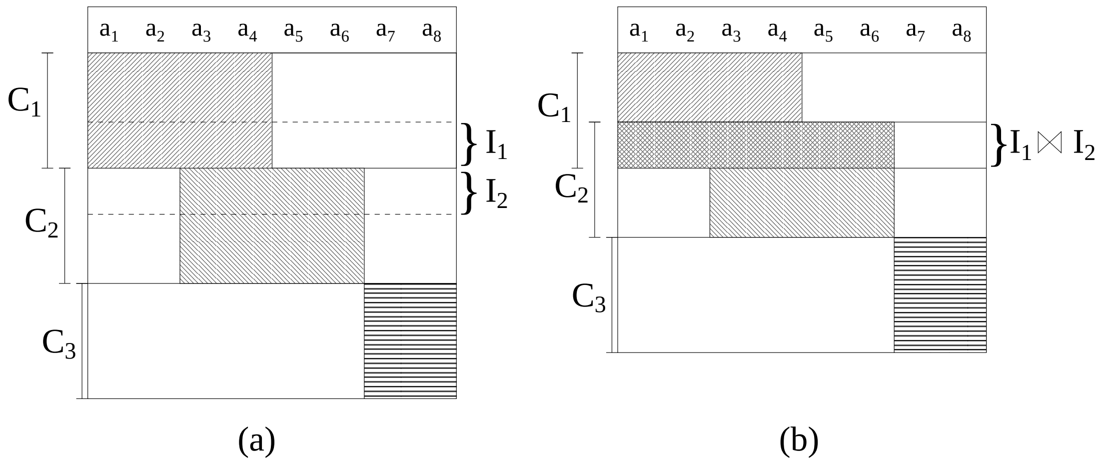

Figura 8: Estensione della classe di entità

associati.
Perché ciò sia possibile è necessario comprendere come le informazioni provenienti dalle varie sorgenti debbano essere combinate e quindi occorre impiegare sia le relazioni tra le estensioni sia quelle sulle intensioni. Le prime permettono di individuare e ricostruire le istanze della classe "*astratta*" di entità e le seconde specificano quali sono le proprietà effettivamente note di tali istanze.

Example 1 Supponiamo di voler gestire un dominio applicativo in cui è presente il concetto di "*persona*" (P) ed immaginiamo di avere tre classi locali (C_1, C_2 e C_3) contenenti informazioni relative a persone. Immaginiamo poi che la classe di entità "*persona*" sia caratterizzata dalle proprietà $a_1, \ldots, a_8$ e che gli schemi delle classi locali siano parzialmente sovrapposti.

$$\mathrm{int(P)} = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$$

$$\mathrm{int}(C_1) = \{a_1, a_2, a_3, a_4\}$$

$$\mathrm{int}(C_2) = \{a_3, a_4, a_5, a_6\}$$

$$\mathrm{int}(C_3) = \{a_7, a_8\}$$

Per arrivare ad un'integrazione corretta può non essere sufficiente fare una semplice unione delle estensioni delle classi, in quanto dati relativi alla stessa entità potrebbero essere presenti in più classi. L'unione, in questo caso, comporterebbe un errore nella rappresentazione, infatti ogni oggetto nelle classi locali verrebbe considerato come un'istanza distinta della classe globale, Figura 8 (a), determinando una duplicazione di informazioni (la stessa persona sarebbe presente più volte), e l'incompletezza della risposta (le persone duplicate hanno un insieme incompleto di proprietà).
Questi inconvenienti possono essere risolti soltanto gestendo in modo corretto le relazioni tra le estensioni. Se ad esempio le estensioni di C_1 e C_2 risultano essere sovrapposte significa che un sottoinsieme delle loro istanze, pur avendo attributi diversi ed appartenendo a sorgenti differenti, corrispondono alle stesse entità del mondo reale. In Figura 8 (b) viene appunto mostrato come

gli oggetti presenti nei sottoinsiemi I_1 e I_2 ed appartenenti alla sovrapposizione tra C_1 e C_2, debbano essere "*fusi*" per fornire un'informazione corretta e completa[1].

Per realizzare un'effettiva integrazione delle informazioni occorre quindi risolvere i conflitti intensionali, in modo da recuperare insiemi di oggetti dalle singole sorgenti, ma è anche necessario fornire gli strumenti che consentano la "*fusione*" di tali oggetti per poter cos ì ricostruire le estensioni delle classi di entità del "*dominio applicativo*".

L'approccio seguito in MOMIS si basa sulla teoria della *formal context analisys* che, come descritto in [10], è volta alla generazione di una gerarchia di ereditarietà in cui viene rappresentata la conoscenza disponibile, nell'insieme di schemi locali, su di un determinato aspetto della realtà. Gli elementi che caratterizzano questo approccio teorico sono:

- definizione di *assiomi estensionali*
  Tali assiomi descrivono le relazioni insiemistiche esistenti tra le estensioni delle sorgenti, in particolare date due classi A e B sono individuabili; quattro tipi di situazioni[2]:

  - *disgiunzione*: $\forall t : S_A^t \cap S_B^t = \emptyset$
  - *equivalenza*: $\forall t : S_A^t = S_B^t$
  - *inclusione*: $\forall t : S_A^t \subseteq S_B^t$
  - *sovrapposizione*: $\forall t : S_A^t \cap S_B^t \neq \emptyset$

  La presenza di un insieme completo e corretto di assiomi è un prerequisito fondamentale per il conseguimento di un'effettiva integrazione. D'altro canto la loro definizione è a carico del progettista e solo in parte può essere automatizzata pertanto diviene fondamentale disporre di strumenti che aiutino il progettista nella fase di specifica.

- individuazione delle *base extension*
  Una "*base extension*" rappresenta un sottoinsieme di entità appartenenti ad uno stesso concetto del dominio applicativo. Presa quindi una classe di entità, un insieme di "*base extension*" ne rappresenta il partizionamento in modo che ogni istanza appartenga ad una ed una sola di esse e che tutte le istanze in una stessa "*base extension*" abbiano lo stesso insieme di proprietà [3].
  Le "*base extension*" sono quindi individuate dalle relazioni estensionali presenti tra classi locali e sono caratterizzate dall'avere come estensione l'insieme di entità formate dall'intersezione delle classi che la compongono e come intensione l'unione dei loro schemi.

  Example 2 Alla sorgente `University` viene aggiunta la classe `University_Worker` con il seguente schema:

  `University_Worker(first_name, last_name, dept_code, pay)`

  Questa classe è stata inserita nel cluster Cl_1. Supponiamo che tra le classi `SCS.CS_Person`, `SU.University_Worker`, e
  `SU.Research_Staff` sussistano le relazioni estensionali mostrate in Figura 9. Come mostrato in tabella 1, le "*base extension*" presenti sono quindi cinque. In particolare possiamo

[1] Ovviamente il concetto di completezza è relativo alle informazioni contenute in uno stesso cluster

[2] con S_A^t e S_B^t si sono indicati gli stati, all'istante t, rispettivamente delle classi A e B.

[3] al contrario di quanto accade per un normale database, una *classe di entità* sarà popolata da istanze caratterizzate dall'avere, in generale, solo un sottoinsieme delle proprietà definite nella classe stessa

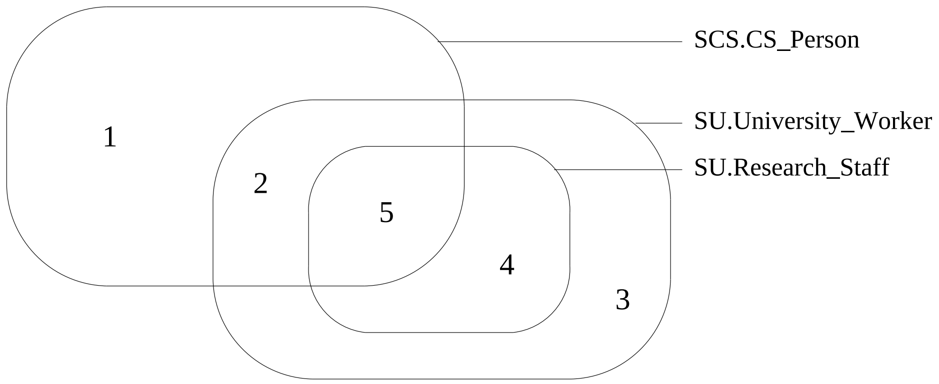

Figura 9: Relazioni estensionali tra classi

Base Extension	**1**	**2**	**3**	**4**	**5**
SCS.CS_Person	1	1	0	0	1
SU.University_Worker	0	1	0	1	1
SU.Research_Staff	0	0	1	1	1

Tabella 1: Rappresentazione delle "*base extension*"

dire che che le istanze in BE1 e BE3 sono caratterizzate dall'avere le sole proprietà, rispettivamente delle classi `SCS.CS_Person` e `SU.University_Worker`, mentre le altre godono delle proprietà presenti in tutte le classi di cui rappresentano l'intersezione.

- generazione della *gerarchia estensionale*
  Sfruttando la conoscenza intensionale, le "*base extension*" relative ad una stessa classe di entità vengono ora riorganizzate in un *concept lattice* che diviene il punto di partenza per ogni processo di interrogazione. L'idea è quella di associare ad ogni classe globale una gerarchia estensionale, composta da classi "*virtuali*". Queste classi sono caratterizzate dall'avere uno schema, ed un'estensione che è composta dall'insieme di base extension soddisfacenti lo schema dato.
  Questa gerarchia, fornita un'interrogazione, deve fornire tutte le informazioni necessarie alla "*costruzione*" delle entità aventi le proprietà richieste. Ciò significa poter individuare tutte le "*base extension*" aventi, almeno, l'insieme di attributi specificati nella query.

Nel seguito mostreremo come questi aspetti siano stati implementati in MOMIS allo scopo di realizzare il processo di integrazione estensionale, sfruttando i risultati dell'integrazione intensionale e le funzionalità offerte da ODB-Tools.

2.1.1 Definizione degli assiomi estensionali

La prima fase del processo di integrazione estensionale deve prevedere un'accurata analisi degli schemi, in modo da individuare un insieme il più possibile completo di relazioni estensionali tra classi. Parte degli assiomi possono essere ricavati direttamente dalle definizioni stesse degli schemi (una relazione di specializzazione tra classi corrisponde ad un assioma di inclusione e lo stesso vale per un'associazione con vincolo di integrità referenziale), ma per le relazioni inter-schema la fonte principale rimane sicuramente il progettista.
Pur avendo una buona conoscenza del dominio applicativo e della semantica degli schemi che devono essere integrati non è semplice individuare tutte le relazioni esistenti tra classi, pertanto

risulta fondamentale disporre di strumenti di supporto sia nella fase di definizione delle relazioni sia in quella di verifica delle stesse. In particolare durante la fase di definizione è importante fare ricorso a metodi che permettano di arrivare all'individuazione di un insieme corretto di assiomi mediante raffinamenti successivi, in questa direzione, ad esempio, una proposta decisamente interessante è quella illustrata in [17].
Occorre anche osservare che l'insieme di asserzioni a cui si è pervenuti può portare ad una parziale revisione dello schema globale prodotto dal modulo Global Schema Builder, infatti l'analisi estensionale fatta in MOMIS si basa su due presupposti:

- tra classi appartenenti ad uno stesso cluster e per le quali non è specificata nessuna relazione si assume che le loro estensioni siano sovrapposte.
- tra classi appartenenti a cluster diversi deve sussistere una relazione di disgiunzione estensionale.

La seconda ipotesi è stata introdotta in quanto le classi globali dovrebbero raccogliere tutte le informazioni relative ad uno stesso concetto del dominio applicativo, pertanto non possono essere presenti entità aventi proprietà partizionate su cluster distinti. Se uno o più assiomi violassero questa condizione allora sarebbe necessario o rimuoverli, perché non corretti, o modificare i cluster, ad esempio fondendoli in uno unico.

Rimane da dire che in MOMIS le relazioni estensionali vengono espresse come rule nel linguaggio ODL$_{I^3}$, senza bisogno di estendere ulteriormente la sintassi e senza creare ulteriori complicazioni all'utilizzatore. La ragione che consente di ricorrere alla sintassi delle rule risiede nel fatto che assiomi estensionali e rule sono semanticamente equivalenti infatti queste ultime non sono che un modo per dire che un insieme di istanze di un certo concetto c_1, che godono di certe proprietà, appartengono ad un altro concetto c_2. Date due classi A e B gli assiomi sono quindi rappresentati nel seguente modo[4]:

1. relazione di *disgiunzione*:

   `rule RE1 forall x in (A and B) then x in bottom`

2. relazione di *inclusione*:

   `rule RE2 forall x in B then x in A`

3. relazione di *equivalenza*

   `rule RE3 forall x in A then x in B`

   `rule RE4 forall x in B then x in A`

Si osserva che le relazioni di sovrapposizione non devono essere rappresentate in modo esplicito infatti, come già detto, in MOMIS si assume che le estensioni di tutte le coppie di classi in uno stesso cluster e per le quali non sia specificata nulla, siano parzialmente sovrapposte.

Facendo riferimento al nostro schema d'esempio possiamo immaginare di definire il seguente insieme di assiomi estensionali:

```
rule RE1a forall x in SU.School_Member
            then x in STP.University_Student;

rule RE1b forall x in STP.University_Student
```

[4]nella logica descrittiva OLCD un tipo o classe *bottom* rappresenta un "*concetto*" incongruente, cioè che non può essere in nessun caso popolato da dati o istanze

```
           then x in SU.School_Member;

rule RE2 forall x in SU.Research_Staff
           then x in SU.University_Worker;

rule RE3 forall x in SCS.Student
           then x in SU.School_Member;

rule RE4 forall x in SCS.Professor
           then x in SU.Research_Staff;

rule RE5 forall x in (SCS.Professor and SU.School_Member)
           then x in bottom;

rule RE6 forall x in (SU.Research_Staff and STP.University_Student)
           then x in bottom;

rule RE7 forall x in (SU.Research_Staff and SCS.Student)
           then x in bottom;
```

Traduzione degli assiomi estensionali in proprietà intensionali

Gli assiomi relativi alle classi di uno stesso cluster sono impiegati per la costruzione della corrispondente gerarchia estensionale. Perché ciò sia possibile occorre prima integrare le informazioni a disposizione, arricchendo le descrizioni delle classi locali con le conoscenze estensionali, a tale scopo viene effettuata una trasformazione degli assiomi in relazioni di ereditarietà con il seguente approccio:

1. ogni asserzione di equivalenza fra due classi porta alla generazione di una classe con intensione corrispondente all'unione delle due intensioni, questa *classe equivalente* va a sostituire quelle iniziali.
   Ad esempio date le seguenti definizioni:

   ```
   interface School_Member            interface University_Student
   { attribute string  name;          { attribute string name;
     attribute string  faculty;         attribute integer studcode;
     attribute integer year; } ;        attribute string faculty;
                                        attribute integer tax; } ;

   rule RE1a forall x in SU.School_Member
               then x in STP.University_Student;

   rule RE1b forall x in STP.University_Student
               then x in SU.School_Member;
   ```

   si introduce la *classe equivalente* che sostituisce **STP.University_Student** e **SU.School_Member** , e la cui intensione è pari all'unione delle rispettive intensioni:

   ```
   interface School_Member_University_Student
   { attribute string  name;
     attribute string  faculty;
     attribute integer studcode;
     attribute integer year;
     attribute integer tax; } ;
   ```

2. ogni asserzione di inclusione "*ridefinisce*" la classe inclusa introducendo l'ereditarietà dalla superclasse. Ad esempio:

```
interface Research_Staff           interface CS_Person
{ attribute string name            { attribute string name; } ;
  attribute string relation;
  attribute string email;          interface Professor : CS_Person
  attribute integer deptcode;      { attribute string title;
  attribute integer sectioncode;     attribute Division belongsto;
                                     attribute string relation; } ;


rule RE4 forall x in SCS.Professor
           then x in SU.Research_Staff;
```

`SCS.Professor` è così ridefinita:

```
interface Professor : CS_Person, Research_Staff
{ attribute string title;
  attribute Division belongsto;
  attribute string relation; } ;
```

3. ogni asserzione di disgiunzione introduce un tipo *bottom* che eredita dalle classi disgiunte. Con riferimento al nostro esempio gli assiomi:

```
rule RE5 forall x in (SCS.Professor and
   School_Member_University_Student) then x in bottom;

rule RE6 forall x in (SU.Research_Staff and
   School_Member_University_Student) then x in bottom;
```

generano le nuove classi *bottom*:

```
view Bottom_P_SS : Professor , School_Member_University_Student
{} ;
view Bottom_RS_SS: Research_Staff ,
                   School_Member_University_Student
{};
```

4. ogni asserzione di sovrapposizione (sono implicite per ogni coppia di classi per cui non sia stata predicata esplicitamente nessun'altra relazione estensionale), porta alla definizione di una nuova classe *virtuale* che specializza entrambe le classi di partenza.
Occorre osservare che oltre alle sovrapposizione tra le classi inizialmente presenti nel cluster occorrerà prendere in considerazione, e quindi trasformare, anche tutte le possibili sovrapposizioni presenti tra le nuove classi virtuali inserite dalla fase stessa di traduzione[5].

```
view Inter_CSP_UW : CS_Person , University_Worker
{} ;
```

[5] l'algoritmo che deve essere impiegato è quindi di tipo iterativo e terminerà solo quando non sarà più possibile aggiungere classi virtuali che non siano specializzazione di classi bottom

La giustificazione per queste trasformazioni deriva dal fatto che definire una relazione tra le estensioni di due classi significa affermare l'esistenza di istanze in classi distinte e corrispondenti alla stessa entità del dominio applicativo. Tale entità può quindi essere immaginata come appartenente ad un classe virtuale avente come schema l'unione delle intensioni delle classi di partenza (in caso di disgiunzione basta adottare il formalismo illustrato per imporre che la nuova classe introdotta abbia estensione sempre vuota).

In questo modo viene creato uno schema virtuale che costituisce la base per le elaborazioni successive. È importante sottolineare che questo schema e soprattutto la gerarchia estensionale da esso ottenuta non andranno a sostituire l'insieme iniziale di classi ma si affiancheranno ad esso per arricchire la conoscenza disponibile sul cluster.

2.1.2 Individuazione delle Base Extension

In questa fase viene impiegato il componente *Schema Validator* [3, 2] di ODB-Tools che, mediante il calcolo della subsumption, è in grado di riorganizzare lo schema virtuale precedentemente ottenuto, in una gerarchia di ereditarietà. Questa gerarchia ha un duplice scopo, infatti da un lato permette la scoperta di incongruenze nelle specifiche estensionali e dall'altro porta all'individuazione delle "*base extension*".
Gli assiomi inconsistenti sono evidenziati dalla presenza di una relazione di specializzazione tra classi e concetti bottom e devono essere risolti dal progettista il quale deve valutare come modificare le relazioni.

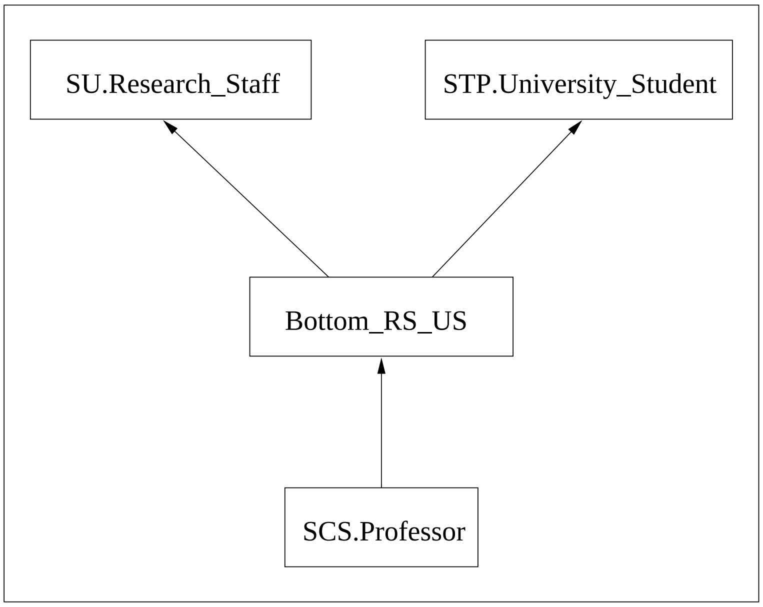

Figura 10: Esempio di gerarchia di ereditarietà

Example 3 Dalle relazioni introdotte sull'esempio di riferimento (`RE7, RE4`) si ricava che le classi `SU.Research_Staff` e `STP.University_Student` sono disgiunte e che `SCS.Professor` è inclusa in `SU.Research_Staff`. Supponendo che venga specificata anche la relazione:

```
rule RE8 forall x in Professor then x in University_Student
```

In questo caso ODB-Tools produrrebbe la tassonomia mostrata in Figura 10, in cui è evidente la presenza di un incongruenza, in quanto la classe `SCS.Professor` non potrebbe mai essere

Base Extension	1	2	3	4	5	6	7	8	9	10	11
STP.University_Student		x					x	x	x	x	
SU.School_Member		x					x	x	x	x	
SCS.CS_Person	x	x		x		x		x	x		x
SCS.Professor	x										
SCS.Student		x						x			
SU.University_Worker	x	x	x		x	x	x				x
SU.Research_Staff	x				x						x

Tabella 2: Composizione delle "*base extension*" del Cluster University_Person

popolata. Per risolvere tale situazione il progettista è chiamato decidere se eliminare la rule `RE8` o se togliere la condizione di disgiunzione tra `SU.Research_Staff` e `STP.University_Student`.

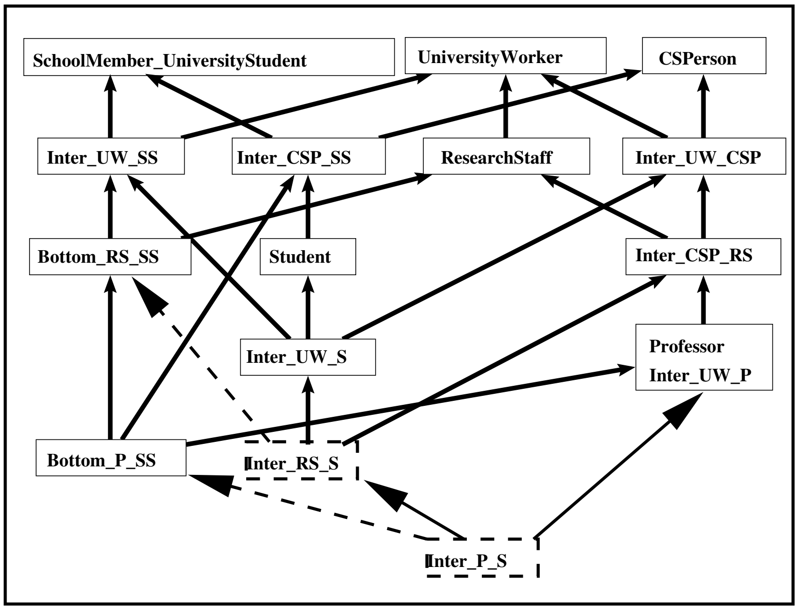

Figura 11: Esempio di Verifica di Congruenza

In Figura 11 è riportata la gerarchia di ereditarietà relativa alla classe globale `University_Person` dell'esempio di riferimento. Questa gerarchia permette, terminata la fase di revisione e dopo aver eliminato i concetti bottom, di ottenere una descrizione completa delle base extension. Ad ogni classe presente nello schema corrisponderà, infatti, una nuova "*base extension*" che è composta da tutte le classi locali di cui, in modo diretto o indiretto, è specializzazione. Ad esempio la classe `Inter_CSP_RS` di Figura 11 rappresenta la "*base extension*" generata dall'intersezione tra le classi `SU.Research_Staff` , `SU.University_Worker` e `SCS.CS_Person` . L'insieme completo delle "*base extension*" individuato dalla gerarchia precedentemente ricavata è riportato in Tabella 2.

L'intera struttura delle "*base extension*" può quindi essere ricavata a partire dall'insieme di classi che la compongono, in particolare:

- l'intensione è data dall'unione degli attributi **globali** descritti nelle classi locali dell'insieme;

- l'estensione è costituita dall'intersezione delle estensioni delle classi locali che la compongono;

Occorre dire che, mentre le informazioni relative all'intensione vengono solamente impiegate per la generazione della gerarchia estensionale, quelle relative alla composizione dell'estensione vanno ad arricchire la rappresentazione globale fornita dal Mediatore. Deve essere infatti costruita un struttura dati permanente, affiancata quindi alla mapping table, che indichi, per ogni base extension, come debbano essere combinate le classi che la compongono. In altri termini deve specificare quali operazioni di join devono essere eseguite, in quale ordine e quali attributi di collegamento si devono impiegare.

Le "*base extension*" rappresentano insiemi disgiunti di entità, però per ricostruirle è necessario effettuare il join delle classi locali che le compongono e quindi ciò che viene prodotto durante questa operazione è in realtà un sovrainsieme della "*base extension*" cercata.

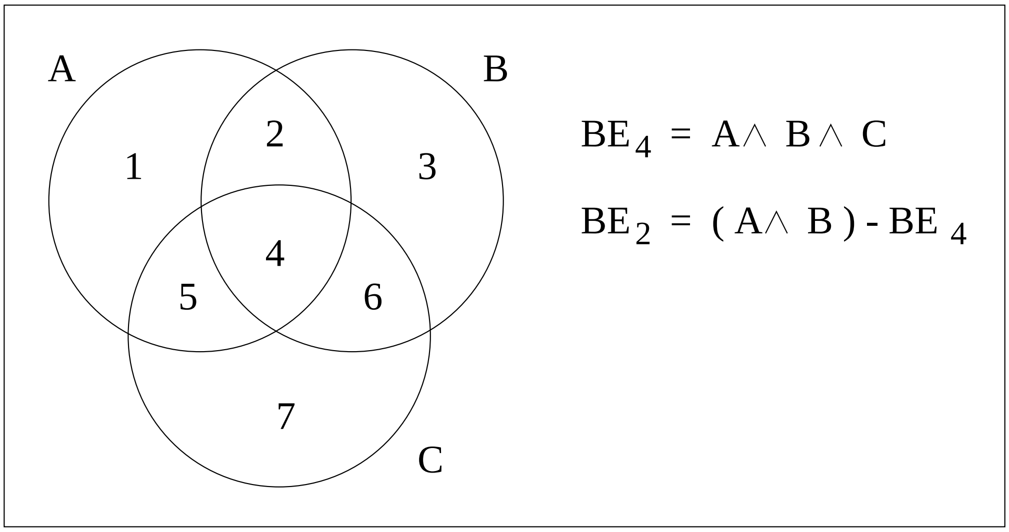

Figura 12: esempio di base extension

Osservando le relazioni estensionali tra le classi A, B e C di Figura 12, risulta evidente che l'intersezione, cioè il join, delle classi A e B produce oltre alle entità della *Base Extension* 2, anche una rappresentazione parziale, non sono infatti recuperati tutti gli attributi, delle entità nella *Base Extension* 4.
Queste considerazioni portano ad introdurre un concetto di *dominazione* tra base extension:

Definition 1 (Dominazione) *Date due* Base Extension BE_1, BE_2 *ed un insieme di attributi* $A = \{a_1, \ldots, a_n\}$, *allora* BE_1 domina BE_2 *rispetto ad A, se A è compreso nelle intensioni di entrambe e se le classi che compongono* BE_1 *sono un sottoinsieme di quelle che formano* BE_2.

Questo concetto diviene importante in fase di interrogazione infatti dovendo recuperare le entità che godono delle proprietà in A sarà sufficiente ricostruire soltanto la "*base extension*" dominante.

2.2 Generazione della gerarchia estensionale

Impiegando l'algoritmo illustrato in [15] ad ogni classe globale viene associata una gerarchia estensionale. Queste gerarchie sono costruite a partire dalle informazioni relative alle "*base extension*", infatti le classi che la compongono sono caratterizzate dall'avere:

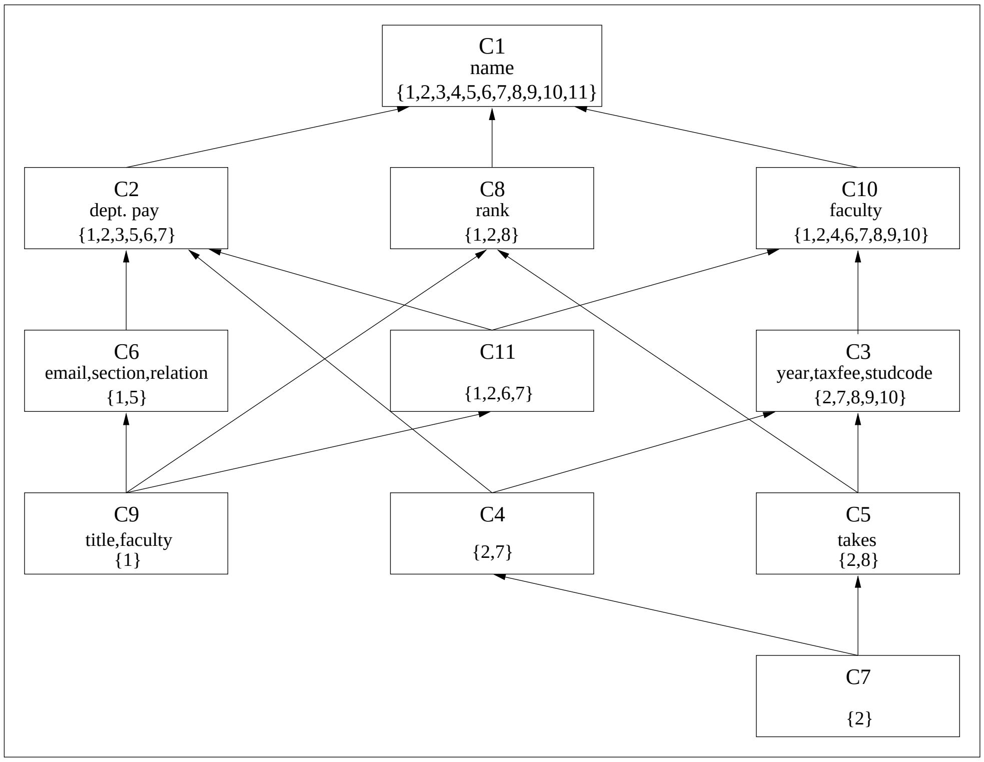

Figura 13: Rappresentazione della gerarchia estensionale

- intensioni corrispondenti agli schemi delle "*base extension*" presenti nella classe globale. In questo modo si individua un insieme di intensioni (sovrapposte), che rappresentano le descrizioni di tutti i tipi di entità[6] raccolte nel cluster. Ciò significa che per ogni intensione esisterà almeno un tipo di entità, appartenente alla classe globale, caratterizzato da tutti e soli gli attributi dell'intensione stessa e che non sono previsti tipi di entità non associati ad almeno una delle intensioni;

- estensione data dall'unione di tutte le "*base extension*" che hanno almeno tutti gli attributi presenti nell'intensione della classe;

Queste gerarchie di classi "*virtuali*"devono dunque essere aggiunte, assieme alle descrizioni delle "*base extension*", allo schema globale fornito dal Mediatore, in modo da riorganizzare la conoscenza espressa nei cluster e migliorare il processo di interrogazione. Disponendo di tali informazioni, è infatti possibile determinare le classi locali coinvolte in un'interrogazione, facendo un'analisi della query ed individuando la classe virtuale più generale che dispone di tutte le proprietà richieste.
In Figura 13 è riportata una rappresentazione della gerarchia estensionale associata alla classe globale `University_Person` dell'esempio di riferimento.

Perchè possano essere conseguiti i risultati descritti, non basta individuare le "*base extension*" ma occorre anche descrivere il modo per fondere le istanze delle classi che le compongono. In mancanza di queste informazioni, sarebbe infatti possibile riconoscere quando un certo insieme

[6]si parla di tipi di entità e non di entità perché tutto ciò che riguarda la descrizione estensionale esprime condizioni di potenziale esistenza, quindi individuare un insieme di entità non significa che sia necessariamente popolato

di istanze corrisponde, in realtà, alle stesse entità del dominio applicativo, ma non saremmo in grado di combinarle, perdendo così la possibilità di ricostruire un'informazione corretta e completa.
Questa ulteriore conoscenza non può essere generata in modo automatico, pertanto diventa chiara l'importanza del progettista, non solo nella fase di definizione degli assiomi estensionali ma durante tutto il processo di integrazione. In particolare si dovrà prevedere una sezione interattiva, che consenta al progettista di individuare le operazioni necessarie alla generazione delle "*base extension*" a partire dalle classi locali che le compongono. In questa fase il progettista deve quindi:

1. trovare, per ogni "*base extension*", una più chiavi semantiche, cioè insiemi di attributi che individuino le entità del dominio in modo univoco[7];

2. determinare quali operazioni di join possono essere eseguite tra le classi locali di una stessa "*base extension*". Ciò significa stabilire, per ogni coppia di classi locali, se è presente almeno una chiave semantica comune che permetta il join;

3. individuare almeno un insieme, o sequenza, di join che consenta la fusione di tutte le classi di una "*base extension*" (ciò significa che se non è possibile effettuare il join tra due classi devono comunque essere presenti altre classi di collegamento);

Queste operazioni sono dunque volte al completamento della struttura dati che descrive le "*base extension*".

3 Il modulo Query Manager

Il sistema MOMIS gestisce una pluralità di sorgenti distribuite ed eterogenee adottando un approccio "*virtuale*", che quindi non prevede la replicazione dei dati nel Mediatore. Ciò significa che dovrà essere il Query Manager a gestire, per ogni interrogazione, la rappresentazione globale ottenuta mediante le fasi di integrazione intensionale ed estensionale al fine di materializzare presso l'utente le entità che popolano questa vista virtuale e che costituiscono la risposta cercata. Tale risposta prodotta dovrà rispondere ad esigenze di correttezza e completezza, in modo da recuperare tutte e sole le entità del dominio applicativo che godono delle proprietà richieste soddisfando le condizioni imposte, ma dovrà anche essere minima. Uno degli aspetti più innovativi del sistema MOMIS consiste appunto nell'impiego di componenti intelligenti che realizzano, sia sulla query globale sia su quelle locali, passi di ottimizzazione semantica capaci di ridurre il numero di sorgenti accedute ed il volume di dati ritornati.
Le fasi che caratterizzano il processo di gestione delle interrogazioni sono:

- *ottimizzazione semantica globale*: sfruttando le informazioni semantiche presenti a livello di schema globale, ed eventuali regole di integrità definite dal progettista, viene realizzata un'ottimizzazione semantica delle interrogazioni poste dall'utente;

- *individuazione delle sorgenti coinvolte*: analizzando la query vengono individuate le classi globali coinvolte e per ognuna di esse si determina a quali classi sorgenti si deve accedere;

- *generazione delle query locali*: sfruttando le regole di mapping tra rappresentazione globale e schemi locali viene prodotto un insieme di sottoquery direttamente eseguibile sulle sorgenti;

[7]questa operazione richiede una profonda conoscenza della semantica sia del contesto applicativo, sia delle singole sorgenti

- *ottimizzazione semantica locale*: Una volta generate le subquery per ogni sorgente, si può pensare di sfruttare la presenza di vincoli di integrità sugli schemi delle sorgenti, unitamente alle capacità di ODB-Tools, per ridurre ulteriormente il costo di accesso ai dati. Questa opportunità può essere sfruttata purché siano rappresentate a livello di Mediatore le conoscenze semantiche relative agli schemi locali;

- *composizione della risposta*: i dati ritornati dalle sottoquery vengono ora combinati per generare la risposta all'interrogazione posta sullo schema globale

I primi passi di tale processo sono volti alla generazione di un piano di esecuzione che, per la singola query, specifichi quali dati devono essere reperiti dalle sorgenti e quali operazioni devono essere eseguite su di essi, mentre l'ultimo rappresenta la messa in esecuzione del suddetto piano per ottenere la risposta.

3.1 Ottimizzazione semantica globale

In questa fase MOMIS opera sulla query dell'utente sfruttando le tecniche di ottimizzazione semantica [18, 19] supportate da ODB-Tools al fine di ridurre il costo del piano di accesso che sarà generato.
Naturalmente, condizione necessaria per questa fase di ottimizzazione è la presenza di regole di integrità inter-sorgenti definite sullo schema globale. È pertanto il progettista che deve analizzare la semantica del dominio applicativo per individuare la presenza di regole di integrità applicabili a tutte le sorgenti. I vincoli inter-sorgenti sono definiti in termini di rule ODL_{I^3} espresse sullo schema globale e vengono impiegati da ODB-Tools per riformulare la query iniziale producendone una semanticamente equivalente ma eseguibile in modo più efficiente. Supponiamo, per esempio, che nel nostro dominio universitario esista, a livello globale, una relazione che unisce i fondi di ricerca (rappresentati dall'attributo globale `budget`) all'area (`area`) di appartenenza del dipartimento a cui questi fondi sono stati assegnati. Attraverso il linguaggio ODL_{I^3}, il progettista può quindi definire la seguente regola di integrità sulla classe globale `Workplace`:

rule R1 **for all** `X` **in** `Workplace`: (`X.budget` > 60000)
then `X.area` = 'Engineering'

Si consideri ora la seguente interrogazione: "Seleziona i nomi dei professori che lavorano in un dipartimento che ha un budget di ricerca maggiore di 60000 dollari"::

```
select  name
from    University_Person
where   rank = 'professor'
and     works.budget > 80000
```

Il mediatore, sfruttando la regola R1, trasforma la precedente query in:

```
select  name
from    University_Person
where   rank = 'professor'
and     works.budget > 80000
and     works.area = 'Engineering'
```

L'interrogazione trasformata ha un maggior numero di predicati presenti nella clausola *where*: questo processo, nonostante appesantisca la fase di query plan (ora deve essere riformulata una query più complessa rispetto all'originale), potrebbe alleggerire il lavoro di query processing delle singole sorgenti, nel caso in cui siano presenti indici secondari definiti sugli attributi aggiunti nella nuova query.
È opportuno osservare che perché questa fase di ottimizzazione possa effettivamente migliorare l'attività svolta da Query Manager è indispensabile fornire gli strumenti necessari a rendere la rappresentazione globale quanto più possibile espressiva. L'integrazione intensionale deve quindi essere seguita da una fase di raffinamento durante la quale, oltre all'individuazione delle regole di integrità globali, sia possibile arricchire lo schema evidenziando la presenza di eventuali gerarchie di aggregazione ed ereditarietà.

3.2 Individuazione delle sorgenti

Ricevendo una interrogazione posta su di una classe globale il Query Manager, innanzitutto, deve essere in grado di individuare l'insieme di classi locali contenenti i dati necessari alla generazione della risposta cercata. Questa fase deve essere svolta in modo da pervenire ad una risposta che sia il più possibile *corretta* e *completa*, dove per correttezza si intende la possibilità di reperire le sole istanze che godono delle proprietà richieste, soddisfando tutte le condizioni imposte, mentre la completezza indica la capacità di individuare tutte le sorgenti che possono contribuire alla risposta.
Mediante la mapping table si riesce solamente in parte a conseguire questi obiettivi, infatti data una query globale le informazioni di collegamento in essa rappresentate permettono di determinare, quali delle proprietà richieste nella query sono presenti in ogni classe locale appartenente al cluster, ma non consente la ricostruzione degli oggetti virtuali che rappresentano le entità descritte in termini globali. Impiegando soltanto questa struttura dati alcune delle politiche che possono essere adottate per la scelta delle sorgenti da interrogare sono ad esempio:

- eliminazione di tutte le classi locali che non prevedono nel loro schema almeno uno degli attributi globali richiesti (l'attributo corrisponde ad un valore "*null*" nella mapping table).
- eliminazione delle classi per le quali esiste almeno una proprietà non *verificabile*, cioè la query ha un predicato di selezione definito su di un attributo non presente nella sorgente.
- definizione di un livello di *credibilità* delle risposte che esprima la percentuale di condizioni che l'utente desidera siano verificate dai dati ricevuti.

Questi approcci portano alla valutazione di una risposta che può essere più o meno corretta (ciò dipende dalla politica adottata), ma che sicuramente è incompleta, infatti le sorgenti vengono considerate singolarmente perdendo la possibilità di individuare e ricostruire le entità proprie del contesto applicativo. Tali entità, come descritto in 2.1, sono caratterizzate dall'avere proprietà distribuite su più classi locali appartenenti a sorgenti differenti, pertanto la completezza della risposta può essere garantita solamente valutandone la correttezza in termini globali.
Considerando un certo insieme di oggetti locali corrispondenti alla stessa entità astratta, può infatti accadere che nessuno di essi goda di tutte le proprità richieste dall'interrogazione ma che opportunamente fusi generino un'informazione corretta.

Example 4 Consideriamo una semplice interrogazione che richieda i campi `title` e `relation` dalla classe `University_Person` .

```
select title, relation
from University_Person
```

Dall'analisi della mapping table si scopre che nessuna delle classi locali possiede entrambi gli attributi presenti nell'interrogazione, quindi facendo riferimento a singole istanze si potrebbe generare solamente una risposta parziale, ottenuta recuperando `title` da `SCS.Professor` e `relation` da `Research_Staff`.
Sfruttando però le informazioni estensionali si perviene ad una risposta più completa, infatti analizzando la gerarchia estensionale di Figura 13 si individua una classe virtuale, C_9, che presenta i campi richiesti e la cui estensione è rappresentata dalla *Base Extension* 1. Pertanto fondendo le classi `SCS.CS_Person` , `SCS.Professor` , `SU.University_Worker` e `Research_Staff`, che formano la "*base extension*" individuata, si è in grado di risolvere la query in modo corretto.

Per capire quali concetti del mondo reale contengono le informazioni richieste e quali sorgenti debbano essere interrogate per ricostruirli occorre allora gestire oltre alle mapping table anche la conoscenza estensionale rappresentata dalle "*base extension*" e dalla Gerarchia Estensionale. L'impiego di tale conoscenza oltre a consentire il soddisfacimento dei requisiti di correttezza e completezza, introduce anche significativi elementi di ottimizzazione, permettendo l'eliminazione di inutili, duplicazioni.
Se ad esempio dovessimo accedere alle classi `SU.School_Member` e `STP.University_Student` per reperire gli attributi `name`, `rank` e `faculty` è possibile sfruttare la relazione di equivalenza tra le due classi e decidere di interrogarne soltanto una. La stessa situazione potrebbe verificarsi in presenza di relazioni di inclusione infatti se la superclasse possiede tutti gli attributi richiesti allora è del tutto inutile accedere anche alla classe inclusa. Si noti che le due classi possono appartenere a database differenti e quindi la specializzazione dell'interrogazione non potrebbe essere fatta da nessun DBMS locale.
In merito alla duplicazione occorre anche dire che se non fosse gestita in modo corretto, oltre a determinare un maggiore costo di esecuzione (un numero maggiore di dati che deve essere trasferito), porterebbe ad un errore nella risposta, infatti gli stessi dati, cioè relativi ad una stesa entità del contesto applicativo, potrebbero essere presentati all'utente più volte, figurando di fatto come associati ad entità differenti.
Sfruttando la conoscenza estensionale viene quindi completato il processo che porta all'individuazione delle sorgenti ed i passi che devono essere seguiti sono:

1. **analisi della query**
   in questa fase vengono esaminati i predicati di proiezione e selezione, al fine di stabilire quali sono gli attributi globali richiesti e, rispetto ai quali, deve essere valutata la correttezza della risposta. Ad esempio, nella query seguente:

   ```
   select name, year
   from University_Person
   where pay > 80000
   ```

   sono presenti gli attributi `name`, `year` e `pay`.

2. **individuazione delle "*base extension*"**
   analizzando gli schemi delle classi virtuali che compongono la gerarchia estensionale si scopre su quale tra esse deve essere posta la query. In particolare dovrà essere scelta come classe target la più generalizzata tra le classi che godono di tutte le proprietà richieste ed è composta dal maggior numero di "*base extension*", cioè quella che ha l'estensione massima. Rispetto alla query `Q1` e alla gerarchia mostrata in Figura 13, si scopre che la classe target è C_4 la cui estensione è costituita dalle *Base Extension* 2 e 7.

3. **individuazione delle classi locali e semplificazione del query plan**
Le strutture dati associate alle "*base extension*" indicano quali classi locali devono essere combinate per materializzarne l'estensione, pertanto, a partire dalle base extension trovate al passo precedente, si è in grado di determinare le classi locali alle quali devono essere rivolte le richieste di dati.
Noto l'insieme di "*base extension*" e le classi locali che le compongono è possibile, e necessario, realizzare ulteriori passi di semplificazione che, eliminando eventuali ridondanze, portino ad una riduzione del numero di subquery da generare e al conseguimento di una risposta corretta e completa. Durante questa fase occorre valutare quali "*base extension*" debbano essere effettivamente ricostruite e a quali classi locali si debba accedere. La semplificazione avviene quindi in due momenti distinti:

   - eliminazione di "*base extension*":
   Il concetto di dominazione introdotto in 1 viene a questo punto utilizzato per eliminare eventuali "*base extension*" ridondanti, infatti analizzando gli attributi che devono essere reperiti, si può scoprire che una "*base extension*" ne domina altre dunque queste devono essere scartate dal piano di accesso. Nell'esempio riportato (Tabella 2), si ha che la "*base extension*" 2 è dominata dalla 7 e quindi deve essere mantenuta soltanto quest'ultima. Occorre comunque osservare che questo passo di semplificazione non elimina tutte le possibili ridondanze, infatti se vi sono due "*base extension*" che ne dominano una terza allora questa può essere scartata ma le restanti, una volta ricostruite, producono insiemi di entità parzialmente sovrapposti (la sovrapposizione è rappresentata proprio dall'estensione della "*base extension*" scartata). In una situazione di questo tipo non sarà sufficiente fare una semplice unione delle entità ma si dovrà effettuare un *outer join* tra le "*base extension*" in modo da presentare una sola volta la porzione comune.
   - eliminazione di classi locali:
   Per ogni "*base extension*" rimasta è possibile ridurre il numero di classi locali alle quali occorre accedere per ottenerne la ricostruzione.
   Per capire quali classi locali possano essere tralasciate occorre considerare il tipo di relazioni estensionali esistenti ed anche il tipo di database. Presa una "*base extension*" può accadere infatti che due delle classi siano estensionalmente equivalenti pertanto, se nessuna delle due aggiunge informazioni rispetto all'altra (perché contiene attributi richiesti dalla query e non presenti nell'altra classe), allora è possibile effettuare una semplificazione al piano di accesso scartandone una.
   Considerando ancora l'esempio fatto ed osservando la sola "*base extension*" rimasta (7), dalla Tabella 2 si nota che essa è formata dalle classi locali `STP.University_Student` , `SU.School_Member` e `SU.University_Worker` . Le classi `SU.School_Member` e `STP.University_Student` sono estensionalmente equivalenti però la prima è in grado di fornire un'informazione più completa, contenendo due degli attributi richiesti (`name` e `year`). In queste condizioni è possibile decidere di scartare la classe `STP.University_Student` e dunque la query `Q1` sarebbe risolta interrogando le sole classi locali `SU.School_Member` e `SU.University_Worker`.
   Un'altra situazione in cui è possibile operare una semplificazione all'interno di una "*base extension*", si verifica quando due classi appartenenti allo stesso database sono una la specializzazione dell'altra. In queste condizioni, infatti, se la sorgente è object-oriented o se, comunque, la classe più specializzata contiene tutte le proprietà richieste, allora è corretto scartare dal piano di ricostruzione della "*base extension*" la superclasse, dal momento che non produrrebbe nessuna conoscenza aggiuntiva.

Occorre aggiungere che nel processo di semplificazione è necessario considerare, oltre agli attributi presenti nell'interrogazione, anche quelli necessari alla ricostruzione delle "*base extension*". In particolare, questi attributi sono rilevanti al fine di decidere se scartare o no una classe, infatti quando la chiave primaria che permette la ricombinazione dei componenti della "*base extension*" non è la stessa per tutte le classi, è possibile che il join diretto tra due di esse non sia effettuabile e sia necessario utilizzarne una terza di collegamento. È evidente che per fondere le prime due classi occorre recuperare anche la terza.
Immaginiamo, ad esempio, di avere una "*base extension*" formata dalle classi A, B, C, con i seguenti schemi:

$\text{int(A)} = \{a_1, a_2, a_3, a_4, a_5\}$

$\text{int(B)} = \{b_1, b_2, b_3, b_4\}$

$\text{int(C)} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$

Supponiamo ora di avere individuato due distinte chiavi semantiche K_1 e K_2, corrispondenti, rispettivamente, alle coppie di attributi: $\{a_1, a_2\}$, $\{c_1, c_2\}$ e $\{b_1, b_2\}$, $\{c_3, c_4\}$
Supponiamo ora che le coppie di attributi $\{a_1, a_2\}$ e $\{c_1, c_2\}$ rappresentino la stessa chiave semantica, consentendo il join tra le classi A e B, mentre le coppie $\{b_1, b_2\}$ e $\{c_3, c_4\}$ corrispondono alla chiave che permette di combinare le classi B e C.
A questo punti è evidente che soltanto la terza classe dispone di tutte le informazioni necessarie a ricostruire la "*base extension*", infatti per fondere le classi A e B è necessario fare prima il join tra A e C, quindi fare il join, del risultato ottenuto, con B .

Da quanto illustrato emerge che il risultato di questa fase non deve essere un semplice insieme di classi locali ma piuttosto un piano di accesso che specifichi, le sorgenti coinvolte ma anche le operazioni che il Query Manager dovrà eseguire per la generazione della risposta. Questo piano deve quindi indicare quali sono le operazioni di join da effettuare per ricomporre le "*base extension*" e come queste ultime devono essere unite, specificando se fare delle semplici unioni piuttosto che degli outer-join.

3.3 Gestione di query complesse

Nella descrizione del processo che porta all'individuazione delle classi da interrogare si è fatta l'ipotesi che la query fosse posta su una singola classe globale e si è implicitamente assunto che non prevedesse operatori complessi. Questa tipologia di interrogazioni costituisce la percentuale maggiore di query significative per un Mediatore, pertanto è quella più interessante in un sistema come MOMIS.
Tuttavia volendo mantenere una completa compatibilità con il linguaggio OQL standard, è necessario estendere il tipo di situazioni gestibili prevedendo una fase di analisi che porti alla determinazione degli elementi che compongono la query stessa.
Come già spiegato, in MOMIS, è stato adottato un approccio virtuale per cui lo schema globale è formato da classi che non contengono istanze ma un insieme di strutture dati necessarie per reperire le informazioni direttamente dalle sorgenti.
Una query, in generale, conterrà richieste che non possono essere soddisfatte localmente quindi,

in modo analogo a quanto avviene ad un normale DBMS; infatti, la query prima deve essere analizzata per individuare le classi globali coinvolte e, per ognuna di esse, le proprietà che devono essere recuperate. Quindi deve essere tradotta ed inoltrata alle sorgenti.
Note le richieste rivolte alle singole classi globali e la query "*globale*" di partenza è quindi possibile produrre un piano di esecuzione che specifichi la sequenza di operazioni che il Query Manager dovrà eseguire in fase di generazione della risposta. La generazione di questo tipo di piano comporta due fasi:

fase1: determinazione delle Basic Query
una Basic Query è una query che contiene tutte le richieste rivolte alle singole classe globale e tale da poter essere soddisfatta direttamente dalle sorgenti.

fase2: ricostruzione delle query globali
si specifica come devono essere impiegati e manipolati i dati restituiti dalle Basic Query in modo da ottenere la risposta cercata. Queste operazioni comprendono ad esempio i join espliciti tra classi globali, la costruzione di collezioni (struct, array, list, ecc..) e gli ordinamenti;

Le *Basic Query* costituiscono quindi gli elementi di base del piano di esecuzione e rappresentano di fatto una restrizione del linguaggio OQL. In particolare rispetto ad una query standard si ha che:

- è possibile navigare attraverso aggregazioni e associazioni per ricostruire oggetti complessi ma non sono ammessi join espliciti tra classi;
- non è prevista la presenza di subquery;
- non vengono restituite strutture complesse come **list**, **array** o **struct**;
- non sono presenti operatori di ordinamento (**order by**) o di conversione come:
  - **listtoset**: trasforma una lista di elementi in un set, quindi privo di oggetti duplicati. Ad esempio lo statement *listtoset(list(1,2,3,2))* restituisce il set composto dagli elementi 1,2,3;
  - **element**: data una collezione di oggetti, ritorna l'elemento in esso contenuto a condizione che esso sia unico;
  - **flatten**: trasforma un collezione di collezione in una collezione ad un solo livello. Ad esempio lo statement *flatten(list(list(1,2),list(1,2,3))* restituisce la lista *list(1,2,1,2,3)*;

Questi operatori devono essere implementati ad un livello superiore (come funzioni invocabili nel piano di esecuzione) in quanto vanno ad agire su insiemi di informazioni già integrate.
Di seguito viene riportato un esempio di come possa essere decomposta una query globale in modo da ottenere il corrispondente piano:

```
Q1: select a.name, a.account.balance, b.employee_code
    from Person as a,
         Employee as b
    where a.name = b.name
    and a.sex = ''male''
    and b.salary > 100.000
```

Questa query globale contiene un join tra classi che viene sostituito dall'insieme di query:

```
Q1.1: select name, account.balance
      from Person
      where sex = ‘‘male’’

Q1.2: select name, employee_code
      from Employee
      where salary > 100.000

Q1.3: select x.name, x.account.balance, y.employee_code
      from Q1 as x,
           Q2 as y
      where x.name = y.name
```

In particolare Q1.1 e Q1.2 sono le basic query rivolte alle classi globali e per le quali deve essere eseguito il processo di individuazione delle classi sorgenti, ottenuto dalla fase uno, mentre Q1.3 rappresenta l'operazione di join che deve essere fatta dal Mediatore mediante l'invocazione della corrispondente funzione, ottenuta dalla fase due.

Le basic query possono essere descritte mediante una grammatica restrizione di quella relativa al linguaggio OQL standard, la cui descrizione in forma BNF è riportata in appendice A.

3.4 Generazione delle query locali

La fase di individuazione delle sorgenti, descritta in precedenza, deve essere applicata ad ogni basic query presente nel piano di esecuzione , individuando, per ognuna di esse un insieme di classi locali. Il Query Manager, disponendo delle descrizioni delle sorgenti e della struttura della basic query, pertanto ha tutte le informazioni necessarie per la generazione di quelle che possiamo definire *Local Query*, cioè le query che da inviare alle sorgenti per il reperimento dei dati.
Questa fase vede l'impiego della mapping table la quale contiene al suo interno tutte le informazioni necessarie alla risoluzione dei conflitti intensionali permettendo quindi la trasformazione della basic query, che ricordiamo è comunque posta sullo schema globale, in query eseguibili sulle sorgenti. Occorre osservare che il Query Manager deve "*riscrivere*" queste interrogazioni in funzione dei singoli schemi locali ma non deve preoccuparsi di quali linguaggi o formalismi essi impieghino. Ciò è reso possibile dall'impiego di un modello comune dei dati grazie al quale il Mediatore comunica con i Wrapper posti sulle sorgenti, saranno infatti questi moduli locali a tradurre l'interrogazione formulata in linguaggi OQL [8] nella sintassi opportuna.
Per meglio comprendere il significato della fase di trasformazione consideriamo un esempio:

```
select name, title, email, dept.name
from University_Person
where dept.employee > 200
```

analizzando la gerarchia estensionale e la tabella delle "*base extension*" si scopre che nella ricostruzione della risposta è coinvolta la sola "*base extension*" formata dalle classi locali `SCS.CS_Person` , `SCS.Professor` e `SU.University_Worker` e `SU.Research_Staff` . Semplificando ulteriormente si trova che basta generare soltanto due query locali[9]:

```
select a.first_name, a.last_name, a.email, b.dept_name
```

[8] ovviamente si intende la forma ristretta descritta precedentemente

[9] Nell'esempio si è assunto che la chiave semantica per ricostruire la "*base extension*" fosse costituita dall'insieme di attributi che formano il nome

```
from School_Member as a
Department as b
where a.dept_code = b.dept_code

select a.name, a.title
from Professor as a
where a.employee_nr > 200
```

È importante osservare che la fase di trasformazione delle query può portare ad ulteriori semplificazioni, infatti utilizzando gli attributi di default presenti nella mapping table e valutando i predicati di selezione delle interrogazioni si è in grado di ridurre il numero di query da inviare alle sorgenti. Se ad esempio fosse formulata una richiesta relativa a tutti i nomi delle persone della facoltà di "*Biology*" allora prima di iniziare la trasformazione in query locali sarebbe possibile eliminare dal piano tutte le "*base extension*" che coinvolgono le classi `SCS.CS_Person` , `SCS.Professor` e `CSPS` per le quali l'attributo `faculty` è settato al valore di default "*CS*", cioè "$Computer_Science$".
Un altro miglioramento nel piano di accesso è ottenibile quando deve essere ricostruita una sola "*base extension*" e tutte le classi locali sono nella stessa sorgente. In queste condizioni infatti è opportuno raggruppare le query locali in una unica ed effettuare il join localmente. Riprendendo l'esempio impiegato nella sezione precedente si nota che la query globale:

```
select name, year
from University_Person
where pay > 80000
```

porterebbe alla generazione delle due query locali:

```
select a.first_name, a.last_name
from University_Worker as a
where a.pay > 80000

select b.first_name, b.last_name, b.year
from School_Member as b
```

Entrambe le interrogazioni sono rivolte alla sorgente `University`, pertanto potrebbero essere convenientemente riunite nell'unica query:

```
select a.first_name, a.last_name, b.year
from University_Worker as a
     School_Member as b
where a.first_name = b.first_name
and a.last_name = b.last_name
and a.pay > 80000
```

3.5 Ottimizzazione semantica locale

La presenza di regole di integrità definite sulle singole sorgenti può essere sfruttata, mediante l'impiego di ODB-Tools, per effettuare un'ulteriore passo di ottimizzazione semantica volta a produrre query locali meno costose. Ad esempio supponendo di disporre della rule:

```
reule RTP1: forall x in University_Student : x.faculty = ``Economisc''
            then x.tax_fee > 12000
```

la seguente interrogazione viene rilevata come inconsistente (la sua risposta è sempre l'insieme vuoto) e questo permette di evitare un inutile accesso ai dati:

```
select name
from University_Student
where faculty = ''Economics''
and tax_fee < 10000
```

L'ottimizzazione semantica locale viene svolta a livello di mediatore poiché è improbabile che tutte le sorgenti siano in grado di realizzarla ed è sembrato troppo pesante includere queste funzionalità nei wrapper.

3.6 Composizione della risposta

Le attività svolte dal Query Manager sino a questo punto sono volte alla preparazione della query all'esecuzione, cioè alla definizione del piano che permette la materializzazione delle informazioni presso il Mediatore . L'ultimo passo deve quindi seguire le indicazioni presenti in questo piano per accedere alle sorgenti e presentare all'utente una risposta integrata.
In questa fase, per prima cosa, il Query Manager dovrà inviare alle sorgenti le query locali presenti nei piani di associati ad ogni classe globale. Queste query restituiranno collezioni di dati che devono essere fuse in modo da ricostruire le "*base extension*" e quindi, tramite queste, l'intera vista sulla singola classe globale. Le operazioni di unione, join e outer-join che devono essere effettuate ed il loro ordine sono indicate nei piani di accesso presenti nell'insieme di informazioni relative alle basic query.
Tutto questo corrisponde alla valutazione delle basic query presenti nel piano di esecuzione , pertanto ora il Query Manager dispone della porzione di schema globale, o meglio della porzione della sua estensione, necessaria a produrre la risposta cercata. Ciò che deve essere fatto consiste infatti nell'applicazione su tale insieme di dati delle elaborazioni indicate nel piano di esecuzione stesso, cioè devono essere eseguiti tutti i join tra classi globali ed aggregazioni che non potevano essere effettuate localmente e che sono necessarie per arrivare ad ottenere la risposta corrispondente alla query globale inizialmente formulata.

4 Progetto e Realizzazione del Query Manager

La progettazione del Query Manager è stata eseguita con l'obiettivo di realizzare un modulo software componibile ed estendibile, in modo da poter affrontare un sottoinsieme delle problematiche individuate e consentire una facile realizzazione di estensioni future. A tale scopo è stata effettuata una modellazione di tipo object-oriented, adottando poi il linguaggio Java nella fase implementativa.

4.1 Query Manager

Le fasi di integrazione intensionale ed estensionale, sono volte alla generazione dello schema globale che dovrà poi essere impiegato dal Query Manager nella fase di gestione delle interrogazioni.

In Figura 14 viene appunto descritto, con uno schema funzionale di alto livello, l'insieme di attività che caratterizzano il sistema MOMIS. In particolare durante la fase di integrazione intensionale il modulo **Global Schema Builder** riceve gli schemi delle sorgenti (espressi in linguaggio ODL$_{I^3}$), ed apre una sezione interattiva che, mediante l'intervento dell'utente e

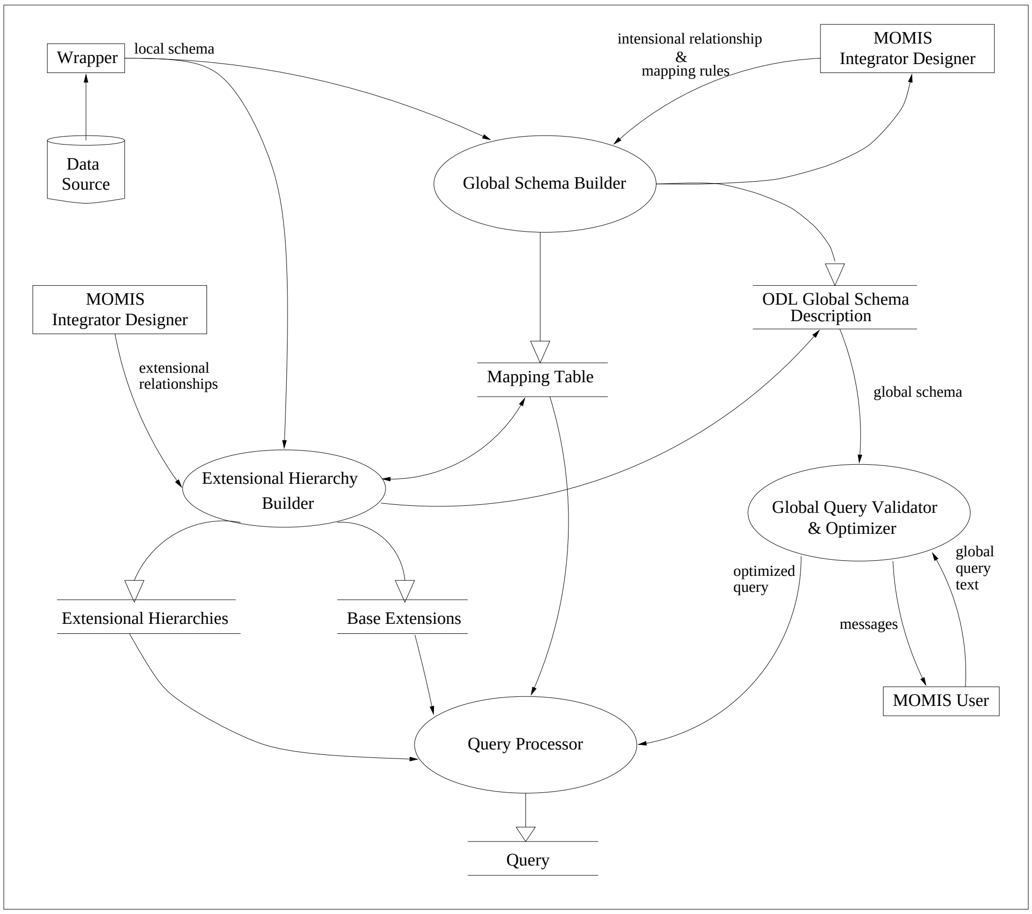

Figura 14: Schema funzionale del Mediatore MOMIS

l'impiego di ODB-Tools, porta alla generazione della **Mapping Table**. Un altro prodotto di questa fase è costituito dalla rappresentazione, sempre in sintassi ODL_{I^3}, dello schema globale integrato, questo schema potrà essere usato dall'utente come supporto durante la formulazione delle interrogazioni e, comunque, rappresenta il punto di partenza per la fase di ottimizzazione globale effettuata mediante il modulo **Query Optimizer** di ODB-Tools.
Il modulo **Extensional Hierarchy Builder** ha, invece, come obiettivo la creazione della **Extensional Hierarchy** e delle **Base Extension**. Questa fase viene svolta interagendo con l'utente che è chiamato ad individuare le relazioni intra, e soprattutto, inter-schema[10]. Come si può osservare, è prevista anche una fase di revisione dello schema globale, in quanto, durante la generazione della gerarchia estensionale, può essere necessario modificare la composizione dei cluster e quindi devono essere riviste le mapping table come anche la rappresentazione dello schema globale.

L'attività di query processing deve comunque utilizzare le strutture dati prodotte nella fase di integrazione, per risolvere ed eseguire le interrogazioni poste dall'utente. Le funzionalità di base che devono essere offerte sono due:

Fase1: *Definizione del Query Plan*

Prima di porre la query in esecuzione occorre "*risolverla*", cioè individuare il piano ad

[10]questa attività si basa quindi, oltre che sulla descrizione delle sorgenti anche sul risultato del processo di clusterizzazione infatti deve essere noto il partizionamento delle classi locali nella mapping table.

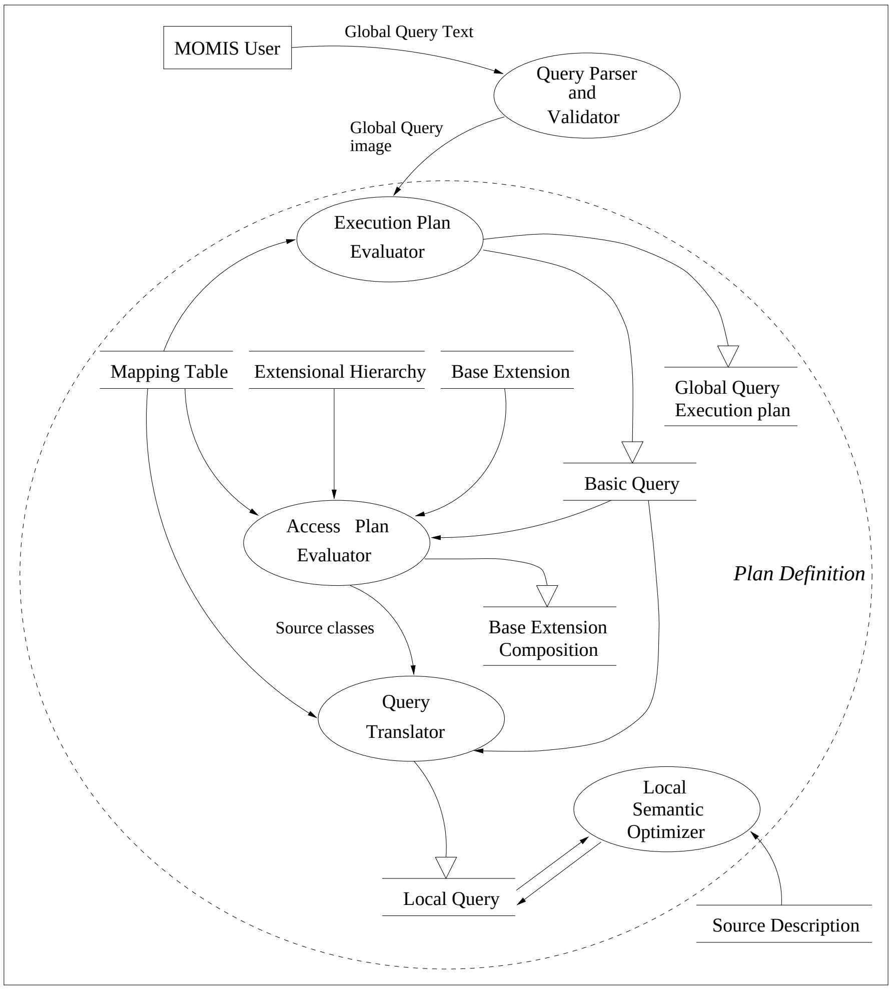

Figura 15: Definizione del piano di esecuzione di una Query

essa associato. L'interrogazione posta sullo schema globale, o *Global Query*, non può essere inviata ai Wrapper e quindi agli schemi locali così come è, infatti:

- è posta su una vista integrata quindi, in generale, conterrà elementi e richiederà dati non presenti in tutte le sorgenti;
- è espressa in termini globali che possono essere differenti sia nella sintassi che nella semantica rispetto a quelli usati localmente;

Il primo punto richiede, come illustrato in figura 15, la definizione di un piano di esecuzione che, a partire dalla query globale e dallo schema globale, porti all'individuazione delle informazioni che possono essere recuperate dalle singole sorgenti. A questo punto è possibile attivare il processo di definizione del piano di accesso che, impiegando le mapping table e la conoscenza estensionale, porta all'individuazione delle classi sorgenti che devono essere effettivamente interrogate. Rimane ora da risolvere anche il secondo aspetto elencato ed effettuare la trasformazione delle interrogazioni locali, in modo che esse possano essere eseguite sulle singole sorgenti.

Fase2: *Query Execution*

La generazione della risposta avviene grazie all'applicazione del piano associato alla query

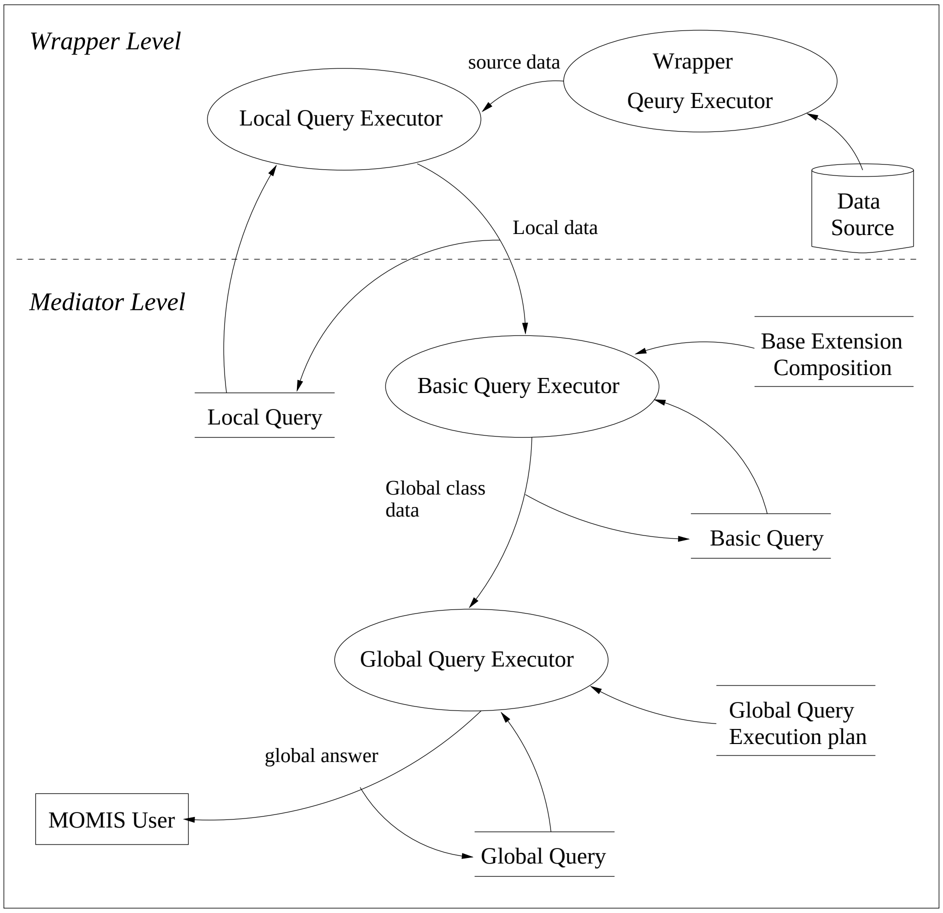

Figura 16: Esecuzione del piano

e generato durante la fase precedente. Tale piano viene applicato a ritroso, pertanto i Wrapper eseguono le query locali per recuperare i dati direttamente dalle sorgenti. Queste informazioni vengono ora combinate, in base alle indicazioni contenute nella struttura dati *Basic Query Composition*, al fine di ottenere le informazioni integrate che costituiscono il contenuto delle singole classi globali. Giunti a questo punto sono state risolte le Basic Query ed i loro risultati devono essere ulteriormente elaborati al fine di ottenere la risposta complessiva. Le operazioni che devono essere effettuate sono dunque descritte nel piano di esecuzione associato alla query globale.

Le funzionalità descritte sono volte esclusivamente alla gestione ed esecuzione delle interrogazioni, occorre però dire che un sistema di questo tipo deve anche fornire strumenti di supporto per la definizione stessa delle query. Nelle future estensioni del sistema è opportuno progettare e sviluppare tool grafici che aiutino l'utente, anche inesperto, nella fase di creazione della query, consentendo un accesso efficace alle informazioni gestite dal sistema. In questa prima parte di sviluppo è stato progettato il modulo Query Manager, modellando le classi necessarie al suo funzionamento e sono stati implementati i moduli "*Query Parser and Validator*" e "*Query Translator*", della Fase 1;

Prima di tutto, mostriamo gli approcci seguiti nella realizzazione delle due funzionalità indicate.

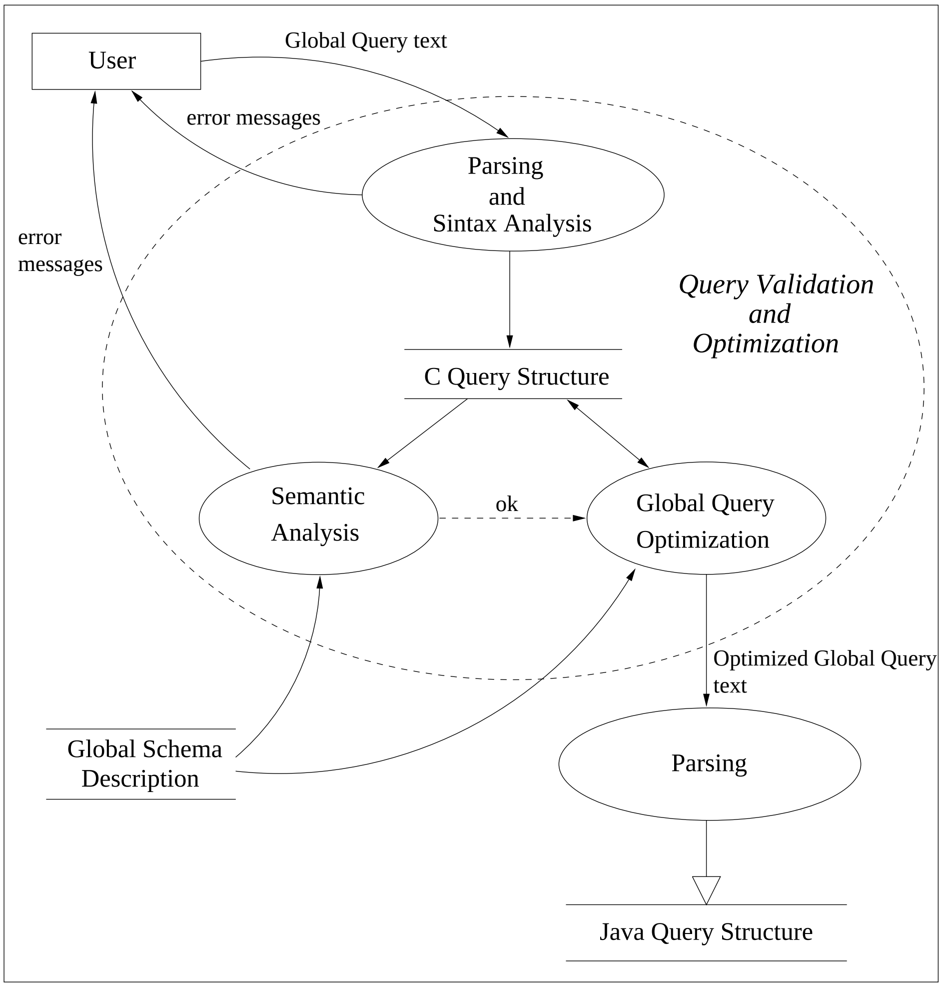

Figura 17: Schema di acquisizione ed ottimizzazione della query

4.1.1 Query Parser and Validator

In Figura 17 vengono illustrate le fasi di acquisizione, validazione ed ottimizzazione dell'interrogazione realizzate nel sistema MOMIS. Si noti che, siccome il sistema ODB-Tools responsabile della validazione ed ottimizzazione è stato sviluppato in linguaggio C, è necessaria una ulteriore fase di parsing per ottenere le strutture Java della query.

4.1.2 Query Translator

In questa fase, vengono esaminate le basic query presenti nel piano di esecuzione al fine di generare le interrogazioni che devono essere rivolte alle sorgenti.
L'adozione di un linguaggio comune di interrogazione tra Mediatore e Wrapper fa sì che il processo di traduzione venga svolto in modo indipendente dai linguaggi impiegati localmente, pertanto ciò che deve essere fatto consiste nel trasformare gli operatori presenti nella query in funzione dei costrutti usati localmente. Come illustrato nella sezione 1.3.3 le corrispondenze tra attributi globali e schemi locali sono riportati nella mapping table, ed alcune delle situazioni più significative che possono presentarsi sono:

- un attributo globale non ha una corrispondente proprietà locale o corrisponde ad un valore costante, in questo caso assegnato da un progettista;
- corrispondenza diretta tra un attributo globale ed uno locale;
- un attributo globale corrisponde a d un insieme di attributi locali;
- un attributo globale corrisponde localmente ad una foreign key;

Le regole di mapping illustrate, pur essendo relativamente semplici, introducono notevoli complicazioni al processo di trasformazione, infatti, non è sufficiente fare una semplice sostituzione di nomi ma occorre provvedere alla trasformazione dell'intero fattore booleano in cui l'attributo è inserito.
Per capire quanto detto è possibile mostrare alcune delle situazioni caratteristiche, ad esempio, utilizzando ancora lo schema di riferimento introdotto nella sezione 1.2, immaginiamo di dover tradurre, per la classe `SU.Research_Staff` della sorgente `University`, la seguente query:

```
select e_mail
from University_Person
where name = "Tracy Miller"
```

la proprietà globale `name` viene mappata localmente nella coppia di attributi `first_name` e `last_name`, pertanto, il predicato di selezione dovrà essere sostituito da una coppia di condizioni poste in *and* ed ottenute decomponendo la stringa "*Tracy Miller*". Il risultato di tale traduzione è dunque:

```
select e_mail
from Research_Staff
where first_name = "Tracy"
and last_name = "Miller"
```

Un'altra situazione tipica è rappresentata dalla presenza di una navigazione implicita, la quale deve essere tradotta localmente in un insieme di join espliciti. In questo caso occorre modificare sia la clausola from, sia il predicato di selezione, in modo da introdurre la condizione di join. Ad esempio la query:

```
select dept.budget_found
from University_Person
```

per la classe `SU.Research_Staff` porterà alla generazione della seguente query locale:

```
select b.budget
from Research_Staff as a,
     Department as b
where a.dept_code = b.dept_code
```

Allo scopo di gestire queste situazioni, si è realizzato il processo di trasformazione dotando tutti gli elementi coinvolti delle funzionalità necessarie a generare la loro traduzione. In particolare:

- gli elementi della mapping table sono stati modellati come oggetti che implementano le regole di mapping specializzando il metodo *toQuery()* della classe **MappingElement**. Questo metodo restituisce un insieme di oggetti, specializzazione della classe **Oql_Query**, che devono essere impiegati per ottenere la rappresentazione locale;

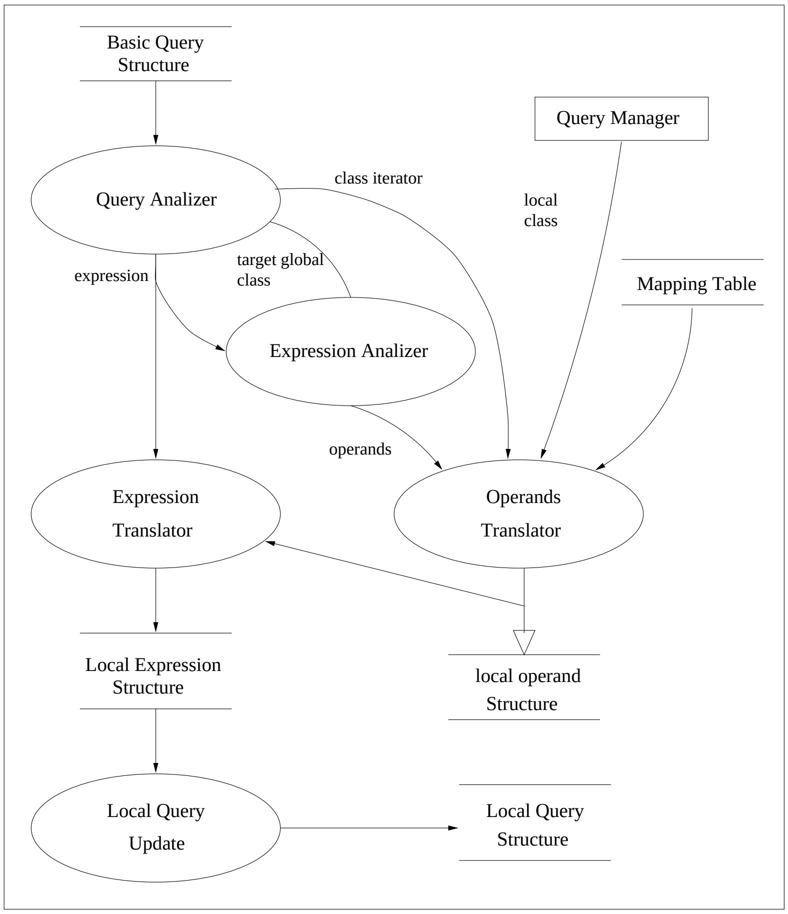

Figura 18: Trasformazione di una query

- gli oggetti che rappresentano le espressioni della query implementano il metodo *translateQuery()* che, gestendo le informazioni relative all'espressione stessa e ai suoi operandi, restituisce gli oggetti che devono essere usati per generare la query locale.

In questo modo, viene adottato un approccio modulare che, da un lato, permette una più semplice ed efficiente gestione del processo di trasformazione e, dall'altro, rende possibile introdurre nuove regole di mapping o modificare quelle esistenti semplicemente definendo nuove classi di oggetti che specializzino i metodi indicati.
Come illustrato in Figura 18, il meccanismo che è stato implementato è di tipo **top-down**, infatti i componenti complessi presenti nell'interrogazione (come possono essere espressioni relazionali o la query stessa), forniscono la loro traduzione agli operatori di livello superiore elaborando le informazioni ottenute dai propri componenti. Questo procedimento viene iterato sino ad arrivare ai singoli operandi, cioè attributi semplici o costanti, che otterranno le rappresentazioni locali direttamente dagli elementi della mapping table.

Per concludere, descriviamo brevemente i quattro package che raccolgono le le classi Java implementate:

1. *package queryman*
   contiene le classi che descrivono gli elementi del piano di esecuzione delle query globali.

Queste classi sono dunque necessarie alla rappresentazione ed esecuzione dell'interrogazione.

2. *package globalschema*
   raccoglie le classi che descrivono lo schema globale, in particolare è stata sviluppata la struttura dati che rappresenta la mapping table e consente la trasformazione delle interrogazioni.

3. *package oql*
   le classi di questo package costituiscono gli elementi della struttura dati usata per rappresentare la query OQL. Questa struttura è dunque il supporto necessario a tutte le elaborazioni che devono essere svolte sulle query.

4. *package utility*
   le classi in questo modulo svolgono funzionalità di utilità generale nell'ambito del Query Manager.

Il software prodotto è stato sviluppato utilizzando la versione **jdk1.2beta4** dell'interprete Java. Le classi implementate sono complessivamente 71, suddivise, in quattro package, nel seguente modo:

- package *oql*: 42 classi;
- package *queryman*: 10 classi;
- package *globalschema*: 14 classi;
- package *utility*: 5 classi.

Il codice prodotto e la relativa documentazione possono essere trovati all'indirizzo *http://sparc20.dsi.unimo.it/tesi/index.html*.

A Restrizione del OQL per le BasicQuery

Le "*base extension*" costituiscono di fatto una restizione del linguaggio OQL. Di seguito viene riportata la descrizione in forma BNF di tale restrizione.

$$
\begin{array}{rcl}
\langle\text{Query}\rangle & ::= & \textbf{(}\ \langle\text{Query}\rangle\ \textbf{)}| \\
 & & \langle\text{SelectExpr}\rangle \\
\langle\text{SelectPreamble}\rangle & ::= & \textbf{select distinct}\ | \\
 & & \textbf{select} \\
\langle\text{SelectExpr}\rangle & ::= & \langle\text{SelectPreamble}\rangle\ \langle\text{ProjectionList}\rangle \\
 & & \langle\text{FromClause}\rangle \\
 & & \langle\text{WhereClause}\rangle \\
\langle\text{ProjectionList}\rangle & ::= & \langle\text{ProjectionAttributes}\rangle\ | \\
 & & \boldsymbol{*} \\
\langle\text{ProjectionAttributes}\rangle & ::= & \langle\text{Attribute}\rangle\ | \\
 & & \langle\text{ProjectionAttributes}\rangle\ \textbf{,}\ \langle\text{Attribute}\rangle \\
\langle\text{Attribute}\rangle & ::= & \langle\text{Projection}\rangle\ | \\
 & & \langle\text{Property}\rangle \\
\langle\text{Projection}\rangle & ::= & \textbf{(}\ \langle\text{Projection}\rangle\ \textbf{)}\ | \\
 & & \langle\text{Identifier}\rangle\ \textbf{,}\ \langle\text{Property}\rangle\ | \\
 & & \langle\text{Property}\rangle\ \textbf{as}\ \langle\text{Identifier}\rangle\ | \\
\langle\text{Property}\rangle & ::= & \textbf{(}\ \langle\text{Property}\rangle\ \textbf{)}\ | \\
 & & \langle\text{Basic}\rangle\ | \\
 & & \langle\text{Identifier}\rangle\ | \\
 & & \langle\text{Accesor}\rangle \\
\langle\text{Basic}\rangle & ::= & \textbf{nil}\ | \\
 & & \textbf{true}\ | \\
 & & \textbf{false}\ | \\
 & & \langle\text{FloatLiteral}\rangle\ | \\
 & & \langle\text{IntegerLiteral}\rangle\ | \\
 & & \langle\text{StringLiteral}\rangle \\
\langle\text{StringLiteral}\rangle & ::= & \text{“}\ \langle\text{String}\rangle\ \text{“} \\
\langle\text{IntegerLiteral}\rangle & ::= & \langle\text{UnsignedLong}\rangle\ | \\
 & & \langle\text{Sign}\rangle\ \langle\text{UnsignedLong}\rangle \\
\langle\text{FloatLiteral}\rangle & ::= & \textbf{.}\ \langle\text{UnsignedLong}\rangle\ | \\
 & & \langle\text{Sign}\rangle\ \textbf{.}\ \langle\text{UnsignedLong}\rangle\ | \\
 & & \langle\text{IntegerLiteral}\rangle\ \textbf{.}\ \langle\text{UnsignedLong}\rangle \\
\langle\text{Sign}\rangle & ::= & +\ |\ - \\
\langle\text{Digit}\rangle & ::= & \textbf{0}\ |\ \textbf{1}\ |\ \ldots|\ \textbf{9} \\
\langle\text{Char}\rangle & ::= & \textbf{a}\ |\ \textbf{b}\ |\ \ldots|\ \textbf{z}\ |\ \textbf{A}\ |\ \textbf{B}\ |\ \ldots|\ \textbf{Z} \\
\langle\text{UnsignedLong}\rangle & ::= & \langle\text{Digit}\rangle\ | \\
 & & \langle\text{Digit}\rangle\ \langle\text{UnsignedLong}\rangle \\
\langle\text{String}\rangle & ::= & \langle\text{Char}\rangle\ |\ \langle\text{Digit}\rangle\ |\ \_\ | \\
 & & \langle\text{Char}\rangle\ \langle\text{String}\rangle\ | \\
 & & \langle\text{Digit}\rangle\ \langle\text{String}\rangle\ | \\
 & & \_\ \langle\text{String}\rangle \\
\langle\text{Identifier}\rangle & ::= & \langle\text{Char}\rangle\ | \\
 & & \langle\text{Char}\rangle\ \langle\text{String}\rangle
\end{array}
$$

$$
\begin{array}{rcl}
\langle\text{Accessor}\rangle & ::= & \langle\text{Identifier}\rangle\ \langle\text{Path}\rangle\ \langle\text{Accessor}\rangle \mid \\
 & & \langle\text{Identifier}\rangle\ \langle\text{Path}\rangle\ \langle\text{Identifier}\rangle \\
\langle\text{Path}\rangle & ::= & .\ \mid\ -> \\
\langle\text{FromClause}\rangle & ::= & \textbf{from}\ \langle\text{VariableDeclaration}\rangle \\
\langle\text{VariableDeclaration}\rangle & ::= & \langle\text{Identifier}\rangle \mid \\
 & & \langle\text{Identifier}\rangle\ \textbf{as}\ \langle\text{Identifier}\rangle \mid \\
 & & \langle\text{Identifier}\rangle\ \langle\text{Identifier}\rangle \\
\langle\text{WhereClause}\rangle & ::= & \epsilon \mid \\
 & & \textbf{where}\ \langle\text{Predicates}\rangle \\
\langle\text{Predicates}\rangle & ::= & \textbf{(}\ \langle\text{Predicates}\rangle\ \textbf{)} \mid \\
 & & \langle\text{Comparison}\rangle \mid \\
 & & \langle\text{BooleanExpr}\rangle \mid \\
 & & \langle\text{CollectionExpr}\rangle \\
\langle\text{Comparison}\rangle & ::= & \langle\text{Operand}\rangle\ \langle\text{ComparisonOperator}\rangle\ \langle\text{Quantifier}\rangle \\
 & & \langle\text{Operand}\rangle \mid \\
 & & \langle\text{Property}\rangle\ \textbf{like}\ \langle\text{StringLiteral}\rangle \\
\langle\text{ComparisonOperator}\rangle & ::= & \mathbf{>} \mid \mathbf{>=} \mid \mathbf{<} \mid \mathbf{<=} \mid \mathbf{=} \mid \mathbf{!=} \\
\langle\text{Quantifier}\rangle & ::= & \epsilon \mid \\
 & & \textbf{some} \mid \\
 & & \textbf{any} \mid \\
 & & \textbf{all} \\
\langle\text{Operand}\rangle & ::= & \langle\text{Basic}\rangle \mid \\
 & & \langle\text{SimpleExpr}\rangle \mid \\
 & & \langle\text{Property}\rangle \\
\langle\text{SimpleExpr}\rangle & ::= & \langle\text{Property}\rangle + \langle\text{Property}\rangle \mid \\
 & & \langle\text{Property}\rangle\ \text{-}\ \langle\text{Property}\rangle \mid \\
 & & \langle\text{Property}\rangle\ /\ \langle\text{Property}\rangle \mid \\
 & & \langle\text{Property}\rangle\ *\ \langle\text{Property}\rangle \mid \\
 & & \text{-}\ \langle\text{Property}\rangle \mid \\
 & & +\ \langle\text{Property}\rangle \mid \\
 & & \langle\text{Property}\rangle\ \textbf{mod}\ \langle\text{Property}\rangle \mid \\
 & & \textbf{abs(}\ \langle\text{Property}\rangle\ \textbf{)} \mid \\
 & & \langle\text{Property}\rangle\ ||\ \langle\text{Property}\rangle \\
\langle\text{BooleanExpr}\rangle & ::= & \textbf{not}\ \langle\text{Predicates}\rangle \mid \\
 & & \langle\text{Predicates}\rangle\ \textbf{and}\ \langle\text{Predicates}\rangle \mid \\
 & & \langle\text{Predicates}\rangle\ \textbf{or}\ \langle\text{Predicates}\rangle \\
\langle\text{CollectionExpr}\rangle & ::= & \textbf{for all}\ \langle\text{Identifier}\rangle\ \textbf{in}\ \langle\text{Property}\rangle\ \textbf{:}\ \langle\text{Condition}\rangle \mid \\
 & & \textbf{exists}\ \langle\text{Identifier}\rangle\ \textbf{in}\ \langle\text{Property}\rangle\ \textbf{:}\ \langle\text{Condition}\rangle \mid \\
 & & \textbf{exists(}\ \langle\text{Accessor}\rangle\ \textbf{)} \mid \\
 & & \textbf{unique(}\ \langle\text{Accessor}\rangle\ \textbf{)} \mid \\
 & & \langle\text{Property}\rangle\ \textbf{in}\ \langle\text{Condition}\rangle \mid \\
 & & \textbf{count(}\ \langle\text{Property}\rangle\ \textbf{)} \mid \\
 & & \textbf{sum(}\ \langle\text{Property}\rangle\ \textbf{)} \mid \\
 & & \textbf{min(}\ \langle\text{Property}\rangle\ \textbf{)} \mid \\
 & & \textbf{max(}\ \langle\text{Property}\rangle\ \textbf{)} \mid \\
 & & \textbf{avg(}\ \langle\text{Property}\rangle\ \textbf{)}
\end{array}
$$

Riferimenti bibliografici

[1] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Reasoning with constraints in database models. Technical Report 103, CIOC - CNR, Viale Risorgimento, 2 Bologna, Italia, September 1994.

[2] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. Odb-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Sesto Convegno AIIA - Roma*, 1997.

[3] S. Bergamaschi, D. Beneventano, C. Sartori, and M. Vincini. ODB-QOPTIMIZER: A tool for semantic query optimization in oodb. In *Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K.*, page 578. IEEE Computer Society Press, 1997.

[4] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. Exploiting schema knowledge for the integration of heterogeneous sources. In *Sesto Convegno Nazionale su Sistemi Evoluti per Basi di Dati - SEBD98, Ancona*, pages 103–122, 1998.

[5] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. An intelligent approach to information integration. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, june 1998.

[6] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, and M. Vincini. A semantic approach to information integration: the momis project. In *Sesto Convegno della Associazione Italiana per l'Intelligenza Artificiale AI*IA98, Padova*, pages 78–82, 1998.

[7] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Records*, 28(1), March 1999.

[8] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[9] Sonia Bergamaschi, Beneventano Domenico, and MAurizio Vincini. Il sistema momis per l'integrazione di sorgenti strutturate e semistrutturate. Technical Report T3–R08, INTERDATA, 1999.

[10] C. Carpineto and G. Romano. Galois: An order-theoretic approach to conceptual clustering. In *Macine Learning Conference*, pages 33–40, 1993.

[11] S. Castano and V. De Antonellis. Deriving global conceptual views from multiple information sources. In *preProc. of ER'97 Preconference Symposium on Conceptual Modeling, Historical Perspectives and Future Directions*, 1997.

[12] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[13] B. Everitt. *Computer-Aided Database Design: the DATAID Project.* Heinemann Educational Books Ltd, Social Science Research Council, 1974.

[14] R. Hull and R. King et al. Arpa i^3 reference architecture, 1995. Available at http://www.isse.gmu.edu/I3_Arch/index.html.

[15] Alberta Rabitti. Architettura di un mediatore per un sistema di integrazione di sorgenti distribuite ed autonome. Tesi di laurea, Università di Modena, Facoltà di ingegneria Informatica, 1997-1998. http://sparc20.dsi.unimo.it/tesi/index.html.

[16] I. Schmitt and G. Saake. Merging inheritance hierarchies for database integration. *IEEE Trans. on Knowledge and Data Engineering.*

[17] I. Schmitt and C. Türker. An Incremental Approach to Schema Integration by Refining Extensional Relationships. In G. Gardarin, J. French, N. Pissinou, K. Makki, and L. Bougamin, editors, *Proc. of the 7th ACM CIKM Int. Conf. on Information and Knowledge Management, November 3–7, 1998, Bethesda, Maryland, USA*, pages 322–330, New York, 1998. ACM Press.

[18] S. Shenoy and M. Ozsoyoglu. A system for semantic query optimization. *ACM-SIGMOD*, pages 181–195, May 1987.

[19] S. Shenoy and M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. and Data Engineering*, 1(3):344–361, September 1989.