ODB-Tool: validazione di schemi e ottimizzazione semantica on-line per basi di dati object oriented *

Domenico Beneventano,[1] Alberto Corni,[1] Stefano Lodi,[2] Maurizio Vincini[1]

E-mail: {benevent, corni, vincini}@dsi.unimo.it
slodi@deis.unibo.it

[1] Dipartimento di Scienze dell'Ingegneria, Università di Modena,
Via G. Campi 213/B, I-41100 Modena
[2] Dipartimento di Elettronica, Informatica e Sistemistica
Centro di Studi per l'Informatica ed i Sistemi di Telecomunicazione
Viale Risorgimento 2, I-40136 Bologna

Abstract.
In questo lavoro viene presentato ODB-Tool, uno strumento software per la validazione di schemi e l'ottimizzazione semantica di interrogazioni per le Basi di Dati Orientate agli Oggetti (OODB), che sfrutta le potenzialità di Internet e del linguaggio JAVA per la rappresentazione grafica *on-line* dei risultati ottenuti.
Gli algoritmi operanti in ODB-Tool sono basati su tecniche di inferenza che sfruttano il calcolo della *sussunzione* e la nozione di *espansione semantica* di interrogazioni per la trasformazione delle query al fine di ottenere inferiori tempi di risposta. Entrambi i concetti sono stati introdotti nell'area dell'Intelligenza Artificiale, più precisamente nell'ambito delle Logiche Descrittive, e sono stati studiati e formalizzati in *OCDL* (*Object Constraint Description Logics*), un linguaggio che permette di esprimere descrizioni di classi, vincoli d'integrità ed interrogazioni.
Sia per quanto riguarda il linguaggio di definizione degli schemi che il linguaggio di interrogazione il tool utilizza lo standard ODMG-93, opportunamente esteso per rappresentare i vincoli di integrità.

1 Introduzione

La validazione degli schemi e l'ottimizzazione di interrogazioni sono temi centrali per la ricerca in ambito OODB. Questo lavoro presenta un prototipo che esegue la validazione automatica degli schemi e l'ottimizzazione di interrogazioni e che si

* Questa ricerca è stata parzialmente finanziata dal progetto M.U.R.S.T. 40% , *"Basi di dati evolute: modelli, metodi e sistemi"*.

basa sul modello OCDL [3] [1], un formalismo per la rappresentazione semantica dei modelli di dati ad oggetti complessi (*CODM*), recentemente proposti per le basi di dati deduttive [3] e basi di dati orientate agli oggetti [4, 5].

OCDL presenta analogie con i formalismi impiegati nella rappresentazione della conoscenza, noti come Description Logics, da cui eredita la possibilità di esprimere condizioni necessarie e sufficienti per l'appartenenza di un oggetto a una classe, e il concetto di sussunzione tra classi. Utilizzando i costruttori disponibili (tupla, set, classe, congiunzione tra classi e tra tipi), tutte le classi e i tipi vengono ottenuti dagli usuali tipi base, cioè interi, reali, stringhe, booleani e tipi mono valore, piú i tipi intervallo.

In tale formalismo, il calcolo della relazione di *sussunzione* permette la determinazione di tutte le relazioni di *specializzazione* tra classi e la rivelazione di classi *inconsistenti*, classi che non possono contenere alcun oggetto della base di dati. Il prototipo presentato in questo lavoro implementa queste tecniche rendendo disponibile un software per la validazione di schemi e l'ottimizzazione semantica di interrogazioni. L'approccio da noi seguito, concordemente alle proposte in [6, 7, 8], è quello di applicare la conoscenza fornita dai vincoli di integrità, espressi per imporre la consistenza di una base di dati, alle interrogazioni eseguite dall'utente, trasformando un'interrogazione in una *equivalente* che può essere eseguita con costi inferiori.

Come interfaccia verso l'utente esterno è stata scelta la proposta ODMG-93 [9], utilizzando il linguaggio ODL (Object Definition Language) per la definizione degli schemi ed il linguaggio OQL (Object Query Language) per la scrittura di query. Entrambi i linguaggi sono stati estesi per la piena compatibilità al formalismo OCDL.

Il lavoro è organizzato nel seguente modo: la sezione 2 presenta il formalismo ODL tramite un esempio di riferimento, le estensioni al linguaggio ODL, e le funzioni inferenziali di validazione e calcolo della sussunzione. Nella sezione 3 vengono presentati la teoria dell'ottimizzazione semantica ed il linguaggio di interrogazione OQL utilizzando alcune query di esempio. Nella sezione 4 viene descritta l'architettura del prototipo e nella sezione 5 vengono descritti la struttura della versione on-line e un esempio completo di utilizzo del prototipo.

2 Il linguaggio ODL

Object Definition Language (ODL) è il linguaggio per la specifica dell'interfaccia di oggetti e di classi nell'ambito della proposta di standard ODMG-93. Il linguaggio svolge negli ODBMS le funzioni di definizione degli schemi che nei sistemi tradizionali sono assegnate al Data Definition Language. Le caratteris-

[3] Nella proposta di standard ODMG-93, si designa con ODL (Object Definition Language) il linguaggio di definizione dei dati. ODL (Object Description Logics) era anche chiamato il formalismo originariamente introdotto in [1] e successivamente esteso in [2]. Nel presente lavoro, per chiarezza indicheremo con OCDL (Object Constraint Description Logics) tale formalismo.

tiche fondamentali di ODL, al pari di altri linguaggi basati sul paradigma ad oggetti, possono essere così riassunte:

- definizione di classi e tipi valore;
- distinzione tra intensione ed estensione di una classe di oggetti;
- definizione di attributi semplici e multivalore (set, list, bag);
- definizione di relazioni e relazioni inverse tra classi di oggetti;
- definizione della signature dei metodi.

La sintassi di ODL estende quella dell'Interface Definition Language, il linguaggio sviluppato nell'ambito del progetto Common Object Request Broker Architecture (CORBA) [10].

2.1 Schema di esempio

Introdurremo ora uno schema che esemplifica la sintassi ODL utilizzata dal nostro prototipo.

L'esempio descrive la realtà universitaria e considera i docenti, gli studenti, i moduli dei corsi e i legami che intercorrono tra loro. In particolare esiste la classe dei moduli (`Section`), distinti in moduli teorici (`STheory`) e di esercitazione (`STraining`). La popolazione universitaria è costituita da dipendenti (`Employee`) e studenti (`Student`). Un sottinsieme dei dipendenti è costituito dai professori (`Professor`) ed esiste la figura dell'assistente tecnico (`TA`) che è al tempo stesso sia un dipendente che uno studente dell'Università. I moduli vengono seguiti da studenti e sono tenuti da assistenti tecnici, o, solo per quelli teorici, da professori.

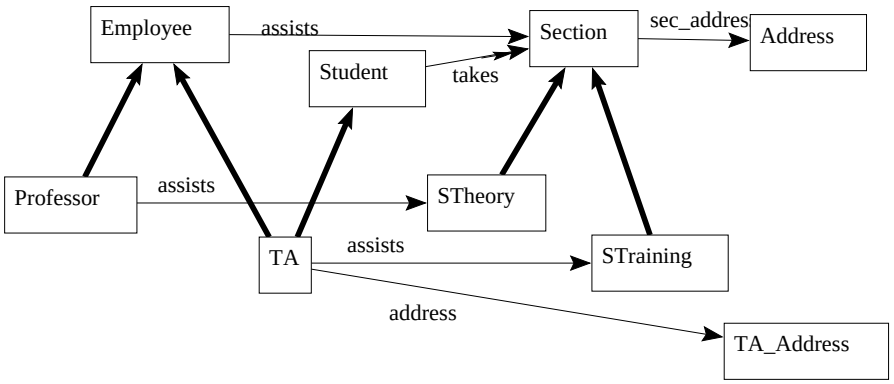

Fig. 1. Rappresentazione grafica dello schema

Nella figura 1 abbiamo la rappresentazione dello schema in cui le classi sono rappresentate da rettangoli, gli attributi corrispondono a frecce (semplici per quelli monovalore, doppie per quelli multivalore), e le relazioni di ereditarietà sono rappresentate graficamente da frecce più marcate.

```
struct Address
{   string city;
    string street; };

interface Section ()
{   attribute string number;
    attribute  Address sec_address; };

interface STheory : Section()
{   attribute integer level; };

interface STraining : Section()
{   attribute string features; };

interface Employee  ()
{   attribute string name;
    attribute unsigned short annual_salary;
    attribute string domicile_city;
    attribute Section assists; };

interface Professor: Employee  ()
{   attribute string rank;
    attribute STheory assists; };

interface TA: Employee, Student ()
{   attribute STraining assists;
    attribute struct TA_Address { string city;
                                  string street;
                                  string tel_number; }
                                  address; };

interface Student ()
{    attribute string name;
     attribute integer student_id;
     attribute set<Section> takes; };
```

Table 1. Esempio ODL: database universitario

Nella tabella 1 è riportata la definizione in ODL delle interfacce per i tipi dello schema.

2.2 ODL e il modello OCDL

In questa sezione viene brevemente descritto il formalismo OCDL, introdotto in [1] ed esteso in [2], [4] e sono discusse le differenze tra ODL e il modello OCDL.

[4] Lavori a cui si rimanda il lettore interessato per ulteriori approfondimenti.

OCDL prevede una ricca struttura per la definizione dei *tipi atomici*: sono presenti gli *integer*, *boolean*, *string*, *real*, *tipi mono-valore* e sottoinsiemi di tipi atomici, quali ad esempio intervalli di interi. A partire dai tipi atomici si possono definire *tipi valore*, attraverso gli usuali costruttori di tipo definiti nei CODM, quali *tuple*, *insiemi* e *tipi classe*, che denotano insiemi di oggetti con una identità ed un valore associato. Ai tipi può essere assegnato un nome, con la distinzione tra nomi per tipi-valore e nomi per tipi-classe (chiamati semplicemente *classi*). In tale assegnamento, il tipo può rappresentare un insieme di condizioni necessarie e sufficienti, o un insieme di condizioni solo necessarie. L'ereditarietà, sia semplice che multipla, è espressa direttamente nella descrizione di una classe tramite l'operatore di *congiunzione*.

Nel seguito analizzeremo i concetti propri del formalismo OCDL che non trovano riscontro nello standard ODMG-93 e le relative estensioni proposte all'ODL standard.

- tipo base `range`.
  In OCDL è possibile esprimere intervalli di interi. Abbiamo perciò introdotto nella sintassi ODL riconosciuta dalla implementazione il costrutto `range`. Ad esempio, si può introdurre un modulo di teoria avanzato `ADVSTheory` come un modulo di teoria `STheory` il cui livello è opportunamente elevato (compreso tra 8 e 10):

  ```
  interface ADVSTheory : STheory()
  {       attribute  range {8, 10}  level;   };
  ```

  Il costrutto `range` è una conveniente estensione in quanto si presta all'impiego per l'ottimizzazione semantica di query in OQL con predicati di confronto.
- viste o classi virtuali (`view`).
  OCDL introduce inoltre la distinzione tra *classe virtuale*, la cui descrizione rappresenta condizioni necessarie e sufficienti di appartenenza di un oggetto del dominio alla classe (corrispondente quindi alla nozione di vista) e *classe base*, la cui descrizione rappresenta solo condizioni necessarie (corrispondente quindi alla classica nozione di classe). In altri termini, l'appartenenza di un oggetto all'interpretazione di una classe base deve essere stabilita esplicitamente, mentre l'interpretazione delle classi virtuali è calcolata sulla base della loro descrizione.
  Le classi base vengono naturalmente introdotte tramite `interface`, mentre per le classi virtuali si introduce un costrutto con le stesse regole sintattiche denominato `view`. Ad esempio, la seguente dichiarazione:

  ```
  view Assistant: Employee, Student ()
  { attribute Address address; };
  ```

  introduce la classe virtuale `Assistant` che rappresenta tutti gli oggetti che sono sia studenti che dipendenti e che hanno un indirizzo rappresentato come una struttura comprendente via e città.

– regole di integrità.
  Le regole di integrità permettono la formulazione dichiarativa di un insieme rilevante di vincoli d'integrità sotto forma di regole *if then* i cui antecedenti e conseguenti sono esprimibili come tipi OCDL. È possibile, in tal modo, dichiarare correlazioni fra proprietà strutturali della stessa classe o condizioni sufficienti per il popolamento di sottoclassi di una classe data.
  Per la rappresentazione delle regole di integrità è stata introdotta una sintassi intuitiva coerente con la proposta ODMG-93. In particolare si sono sfruttati il costrutto `for all`, il costrutto `exists`, gli operatori booleani e i predicati di confronto utilizzati nell'OQL.
  Una regola di integrità è dichiarata attraverso la seguente sintassi:

  rule `<nome-regola>` **for all** `<nome-iteratore>` **in** `<nome-classe>` **:**
  `<condizione-antecedente>`
  then `<condizione-conseguente>`

  Le `condizioni`, antecedente e conseguente, hanno la medesima forma e sono costituite da una lista di espressioni booleane in `and` tra loro; all'interno di una condizione, attributi e oggetti sono identificati mediante la *dot notation*. Ad esempio, introduciamo le seguenti regole di integrità nello schema di riferimento:

```
rule rule_1 forall X in Professor: X.rank = "Full"
   then X.annual_salary >= 60000 ;

rule rule_2 forall X in Employee:  X.annual_salary < 30000
   then X in TA  ;

rule rule_3 forall X in Professor: exists S in X.teaches: S.level > 7
   then X.rank = "Full" ;
```

  La `rule_1` stabilisce che un professore di *rango* "Full" abbia un salario superiore a \$60000. La `rule_2` impone l'appartenenza alla classe degli assistenti a tutti i dipendenti con salario inferiore a \$30000. La `rule_3` dichiara che se un professore tiene *almeno* una sezione di livello superiore a 7 allora il suo rango deve essere "Full".

Essendo un linguaggio di definizione di schemi del tutto generale, ODL contiene alcuni costrutti non previsti nel formalismo OCDL considerati non rilevanti per il nostro prototipo:

Restrizioni :

– OCDL prevede soltanto la definizione di relazioni di aggregazione unidirezionali, quindi non è rappresentata l'operazione di inversione.
– Non sono supportati i tipi `enum` e `union`.
– In OCDL la parte intensionale ed estensionale di una classe sono referenziate dallo stesso nome.

- ODL permette una suddivisione gerarchica dello schema mediante l'uso dei *moduli*, mentre lo schema OCDL è invece piatto.
- Nell'attuale realizzazione del traduttore, le costanti possono essere solo dei *letterali*. Non possono essere ad esempio espressioni algebriche o valori strutturati costanti.

Altri costrutti di ODL non supportati da OCDL, come le *operazioni* e le *eccezioni*, sono ignorati.

2.3 Validazione e Sussunzione

Uno dei problemi principali che il progettista di una base di dati deve affrontare è quello della consistenza delle classi introdotte dello schema. Infatti, molti modelli e linguaggi di definizione dei dati sono sufficientemente espressivi da permettere la rappresentazione di classi inconsistenti, cioè classi che non possono contenere alcun oggetto della base di dati. Tale eventualità sussiste anche in OCDL; la possibilità di esprimere intervalli di interi permette la dichiarazione di classi con attributi omonimi vincolati a intervalli disgiunti. Il prototipo rivela durante la fase di validazione dello schema come inconsistente una eventuale congiunzione di tali classi. Ad esempio, introduciamo un modulo di teoria fondamentale `FSTheory` (con un livello compreso tra 2 e 6) e un modulo di teoria intermedio `ISTheory` che eredita sia da quello fondamentale che da quello avanzato `ADVSTheory`:

```
interface FSTheory : STheory()
{       attribute  range {2, 6}  level;   };

interface ISTheory : FSTheory, ADVSTheory() { };
```

La classe `ISTheory` risulta essere inconsistente in quanto il suo attributo `level` ha come dominio l'intersezione dei due intervalli disgiunti specificati in `FSTheory` e `ADVSTheory`.

Un'altra fonte di inconsistenza nella dichiarazione di classi deriva da attributi omonimi vincolati a strutture differenti, come nel seguente esempio:

```
interface New_STraining : STraining()
{  attribute  struct New_Address { string street;
                                   struct City { string name;
                                                 string state; }
                                                 city; }
                                    sec_address; };
```

Il concetto di *sussunzione* esprime la relazione esistente tra due classi di oggetti quando l'appartenenza di un oggetto alla seconda comporta necessariamente l'appartenenza alla prima. La relazione di sussunzione può essere calcolata automaticamente tramite il confronto sintattico tra le descrizioni delle classi; l'algoritmo di calcolo è stato presentato in [1]. Poichè accanto alle relazioni di ereditarietà definite esplicitamente dal progettista possono esistere altre relazioni

di specializzazione implicite, queste possono essere esplicitate dal calcolo della relazione di sussunzione presenti nell'intero schema: il prototipo, dopo aver verificato la consistenza di ciascuna classe, determina tali relazioni di specializzazione implicite fornendo un valido strumento inferenziale per l'utente progettista della base dei dati.

Ad esempio, il tipo `Address` sussume il tipo `TA_Address` in quanto tutti gli attributi di `Address` sono contenuti in `TA_Address` con lo stesso dominio. Come conseguenza della relazione `Address` sussume `TA_Address` si ha che la classe virtuale `Assistant` sussume la classe `TA`, in quanto tutti gli attributi di `Assistant` sono contenuti in `TA` ed inoltre l'attributo `address` di `Assistant` è definito su `Address` che sussume il dominio (`TA_Address`) dell'attributo `address` della classe `TA`. È importante notare che per l'individuazione della relazione `Assistant` sussume `TA`, la classe `Assistant` sia virtuale: infatti solo considerando la descrizione di `Assistant` come condizione sufficiente (e necessaria) si può affermare che ogni oggetto di `TA` è anche in `Assistant`.

3 Ottimizzazione semantica delle interrogazioni

L'ottimizzazione semantica di una interrogazione utilizza il processo di *espansione semantica*, attraverso il quale viene generata una interrogazione equivalente che incorpora ogni possibile restrizione non presente nell'interrogazione originale e tuttavia *logicamente implicata* dallo schema globale del database (classi + tipi + regole d'integrità). Il processo di espansione semantica di una interrogazione è basato sull'iterazione di questa trasformazione: se un tipo implica l'antecedente di una regola d'integrità allora il conseguente di quella regola può essere aggiunto alla descrizione del tipo stesso. Le implicazioni logiche fra i tipi (il tipo da espandere e l'antecedente di una regola) sono determinate attraverso la relazione di sussunzione. L'algoritmo di calcolo, di complessità polinomiale, è stato proposto in [11] e rappresenta il motore del modulo di ottimizzazione del prototipo.

Il linguaggio di interrogazione utilizzato è compatibile con OQL (Object Query Language), proposto in ODMG-93, sia per l'input delle query che per l'output restituito dopo l'ottimizzazione. Una interrogazione espressa in OQL potrà beneficiare del processo di ottimizzazione semantica se può essere tradotta, in modo equivalente, in una espressione di tipo di OCDL: vista la ricchezza di tale formalismo, un insieme rilevante di interrogazioni può essere ottimizzato. In particolare, si riescono a trattare interrogazioni riferite ad una singola classe con condizioni espresse sulla gerarchia di aggregazione. Tuttavia, poiché il linguaggio di interrogazione OQL è più espressivo del formalismo OCDL, introduciamo, seguendo l'approccio proposto in [12], una separazione ideale dell'interrogazione in una parte *clean*, che può essere rappresentata come tipo in OCDL, e una parte *dirty*, che va oltre l'espressività del sistema di tipi; l'ottimizzazione semantica sarà effettuata solo sulla parte *clean*.

Allo scopo di evidenziare le trasformazioni effettuate dal processo di ottimizzazione semantica, l'interrogazione viene riportata in uscita differenziandone graficamente i vari fattori sulla base della seguente classificazione:

- fattori specificati dall'utente e non modificati (`stampati`)
- fattori modificati o introdotti dall'ottimizzatore (sottolineati)
- fattori dirty, specificati dall'utente ma non trattati dall'ottimizzatore (*corsivati*)

Vediamo alcuni esempi d'interrogazioni che illustrano il nostro metodo (lo schema di riferimento è quello introdotto in 2.1):

Q_1 :

"Seleziona i dipendenti con stipendio annuale inferiore a \$18000 che sono assistenti di una sezione A".

Nel nostro prototipo la query può essere scritta in linguaggio OQL come segue:

```
select  *
from    Employee as E
where   annual_salary < 18000
and     assists in ( select  S
                     from    Section as S
                     where   number = "A" )
```

L'espansione semantica dell'interrogazione, che si ottiene applicando la regola di integrità `rule_2`, porta alla seguente interrogazione equivalente:

```
select  *
from    TA as E
where   annual_salary < 18000
and     assists in ( select  S
                     from    STraining as S
                     where   number = "A" )
```

Questo esempio mostra un'effettiva ottimizzazione dell'interrogazione, indipendente da ogni specifico modello di costo, dovuta alla sostituzione delle classi presenti nell'interrogazione con loro specializzazioni. Infatti, sostituendo `Employee` con `TA` e `Section` con `STraining` si riduce l'insieme di oggetti da controllare per individuare il risultato dell'interrogazione.

Per illustrare la separazione di un'interrogazione nella parte *clean* e *dirty*, modifichiamo la precedente interrogazione:

Q_2 : "Seleziona i dipendenti con stipendio annuale inferiore a \$18000 che sono assistenti di una sezione A tenuta nella stessa città in cui essi vivono".

```
select  *
from    Employee as E
where   annual_salary < 18000
and     assists in ( select  S
                     from    Section as S
                     where   number = "A"
                     and     domicile_city ! = S.sec_address.city)
```

Il nuovo fattore rappresenta la parte *dirty* dell'interrogazione che non può essere tradotta nel formalismo OCDL (essenzialmente perchè esprime un confronto tra due attributi); mentre la parte *clean*, che corrisponde a Q_1, può essere ottimizzata come illustrato in precedenza:

```
select  *
from    TA as E
where   annual_salary < 18000
and     assists in ( select  S
                     from    STraining as S
                     where   number = "A"
                     and     domicile_city != S.sec_address.city )
```

L'ultimo esempio che viene riportato ha lo scopo di evidenziare sia l'uso dei quantificatori nella formulazione delle interrogazioni sia l'applicazione di più regole nel processo di ottimizzazione semantica:

Q_3 : "Seleziona i professori con stipendio annuale inferiore a \$35000 che insegnano *almeno* un corso di livello 9".

```
select  *
from    Professor as P
where   annual_salary < 35000
and     exists S in P.teaches : S.level = 9
```

A tale interrogazione è applicabile la regola `rule_3`, che, introducendo il fattore (`rank` = "Full") rende applicabile anche la regola `rule_1`, la quale aggiunge il fattore (`annual_salary` >= 60000), incompatibile con (`annual_salary` < 35000) specificato nella interrogazione originale. Pertanto tale interrogazione viene rilevata come inconsistente rispetto allo schema del database e quindi, senza effettuare alcun accesso, si può stabilire che il suo risultato è l'insieme vuoto.

4 Architettura di ODB-Tool

ODB-Tool, sviluppato al Dipartimento di Scienze dell'Ingegneria dell'Università di Modena, è un prototipo software per la validazione di schemi e l'ottimizzazione di interrogazioni in ambiente OODB. L'architettura, mostrata in figura 2, presenta i vari moduli integrati che definiscono un ambiente *user-friendly* basato sul linguaggio standard ODMG-93. L'utente inserisce gli schemi in linguaggio ODL e le query in OQL ottenendo come risultato la validazione dello schema, l'ottimizzazione dell'interrogazione (in OQL) e la rappresentazione grafica della gerarchia di ereditarietà e di aggregazione dello schema.

Vediamo in dettaglio la descrizione di ciascun modulo:

- **ODL Interface**
  È il modulo di input degli schemi. Accetta la sintassi descritta in sezione 2 e trasforma le classi in descrizioni native del formalismo OCDL.

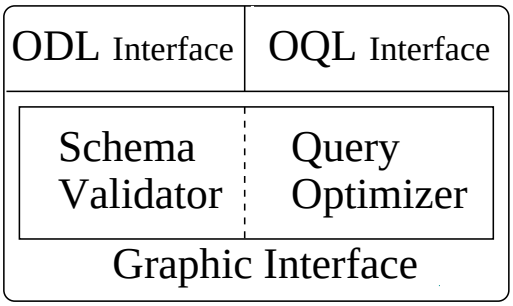

Fig. 2. Architettura di ODB-Tool

- **OQL Interface**
  È il modulo di input e output delle interrogazioni. Utilizza il linguaggio OQL sia per l'input che per l'output della query ottimizzata, che viene trasformata in descrizioni del formalismo OCDL. I predicati booleani in output sono differenziati a seconda del proprio significato:
  - i fattori introdotti o modificati dall'ottimizzazione sono mostrati in colore rosso
  - i fattori non modificati vengono mostrati in colore grigio
  - i fattori ignorati vengono mostrati in colore nero

  In output all'ottimizzazione non sono visualizzati i fattori ridondanti, cioè quei fattori identici a quelli descritti nelle classi referenziate dalla query.
- **Schema Validator**
  È il modulo di validazione degli schemi, ottenuta dal calcolo delle relazioni di sussunzione. Produce come output un insieme di file utilizzati dagli altri moduli per interpretare e rappresentare i risultati.
- **Query Optimizer**
  È il modulo che genera l'ottimizzazione delle interrogazioni. La query viene inserita come descrizione nativa OCDL dal modulo `OQL Interface` e, tramite l'interazione con lo `Schema Validator`, viene ottimizzata calcolandone l'espansione semantica. La query così ottimizzata viene nuovamente inviata all'`OQL Interface` che genera l'output corretto.
- **Graphic Interface**
  È il modulo per la visualizzazione dello schema. Tale rappresentazione è costituita da un grafo i cui nodi rappresentano le classi e gli archi orientati le relazioni di ereditarietà e di aggregazione (opportunamente distinte); per ciascuna classe è possibile visualizzare i nomi ed i domini degli attributi (sia semplici che complessi). Lo schema contiene anche i vincoli di integrità rappresentati ciascuno tramite due classi che specificano l'antecedente ed il conseguente della regola *if then*. Durante il processo di ottimizzazione la query entra a far parte dello schema con la dignità di classe e di conseguenza viene automaticamente inserita nella gerarchia di ereditarietà.

I moduli di interfaccia, validazione ed ottimizzazione sono stati realizzati in

linguaggio C, utilizzando il compilatore gcc 2.7.2, i generatori flex 2.5 e bison 1.24, mentre il modulo di rappresentazione grafica è stato realizzato in JAVA, utilizzando il compilatore JDK 1.1. La piattaforma utilizzata è una SUN SPARC-STATION 20, con sistema operativo Solaris 2.5.

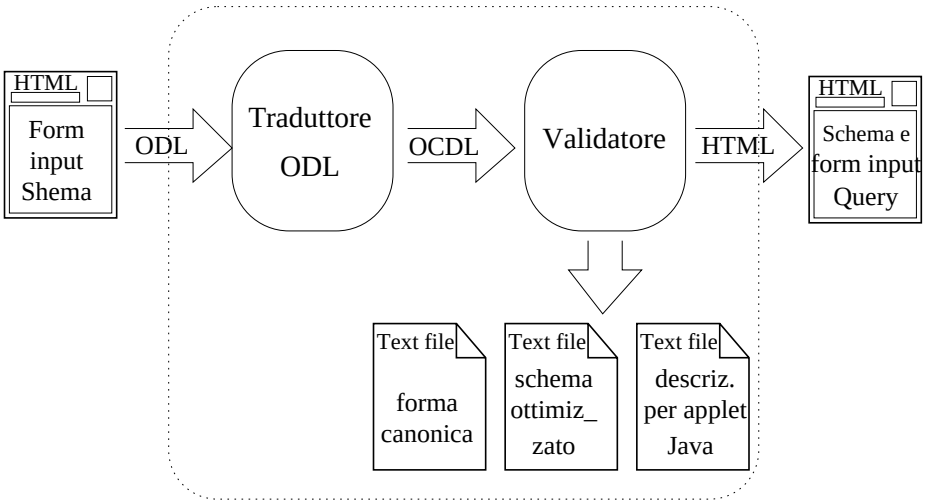

Fig. 3. Validazione dello schema

5 Struttura del sito e demo di esempio

In questa sezione presentiamo le operazioni che l'utente compie durante l'utilizzo di ODB-Tool e le corrispondenti elaborazioni del sistema, considerando come schema di esempio quello presentato nella sezione 2.1.

ODB-Tool è disponibile su internet (`http://sparc20.dsi.unimo.it`) con possibilità di inviare schemi e interrogazioni da macchine remote sia per la validazione degli schemi che per l'ottimizzazione di interrogazioni.

Collegandosi al sito è possibile raggiungere la pagina di DEMO del tool, che richiede in ingresso uno schema di database scritto in linguaggio ODL. È possibile specificare lo schema sia inviando un file generato localmente sia digitando direttamente lo schema in un apposito campo di testo. Lo schema cosí specificato viene analizzato sintatticamente, tradotto in sintassi OCDL e diviene l'input per il modulo di validazione. Al termine di tale processo, viene mostrata una pagina HTML che contiene la rappresentazione grafica dello schema generata dall'applet JAVA e una form per l'inserimento delle query sullo schema dato (vedi fig. 3). In caso di errori sintattici o di rivelazione di classi inconsistenti, durante il processo di validazione viene creata un pagina HTML che informa dettagliatamente l'utente al riguardo.

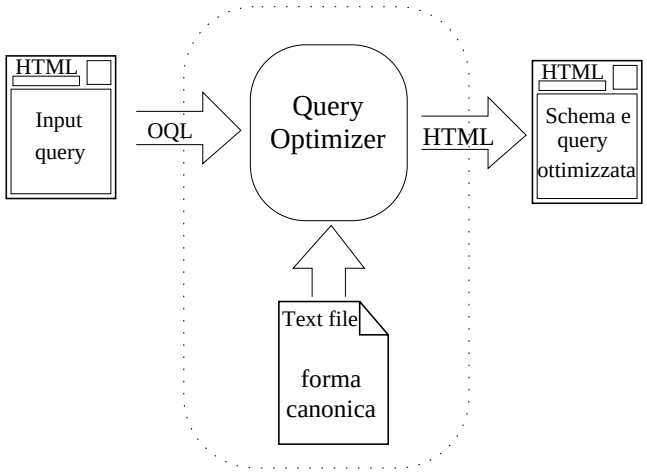

Fig. 4. Ottimizzazione di una query

Ad esempio, per lo schema considerato nella sezione 2.1, il risultato è riportato nella figura 5, dove si può notare la nuova posizione della vista `Assistant` nella gerarchia di ereditarietà. Agendo sull'elemento che rappresenta una classe è possibile aprire una finestra che ne descrive gli attributi (ad es. nella figura 6 viene mostrata la finestra relativa alla classe `Employee`).

Dalla pagina generata in fase di validazione è possibile eseguire interrogazioni in OQL sullo schema, al fine di analizzare il comportamento dell'ottimizzatore semantico. Come per lo schema è possibile specificare un'interrogazione sia inviando un file che digitando la query. La query inviata viene elaborata dal modulo di ottimizzazione e al termine viene mostrata una pagina HTML con la rappresentazione dello schema contenente la query ottimizzata ed opportunamente inserita nella gerarchia di ereditarietà e la descrizione in sintassi OQL della query ottimizzata (vedi fig. 4). Nel caso in cui il processo di ottimizzazione sia avvenuto in più passi è possibile mostrare la classificazione della query all'interno dello schema al termine di ciascun passo.

Come esempio utilizziamo la query Q_1 vista nella sezione 3, che viene ottimizzata fornendo come uscita la pagina riportata nella figura 7, in cui è mostrata la interrogazione originale introdotta dall'utente (`query`) e la interrogazione ottenuta dall'ottimizzazione al primo (e in questo caso unico) passo del calcolo dell'espansione semantica (`query1`).

6 Risultati sperimentali

L'efficienza del prototipo è stata valutata verificando la diminuzione dei costi di esecuzione delle query ottimizzate negli OODBMS commerciali. In particolare

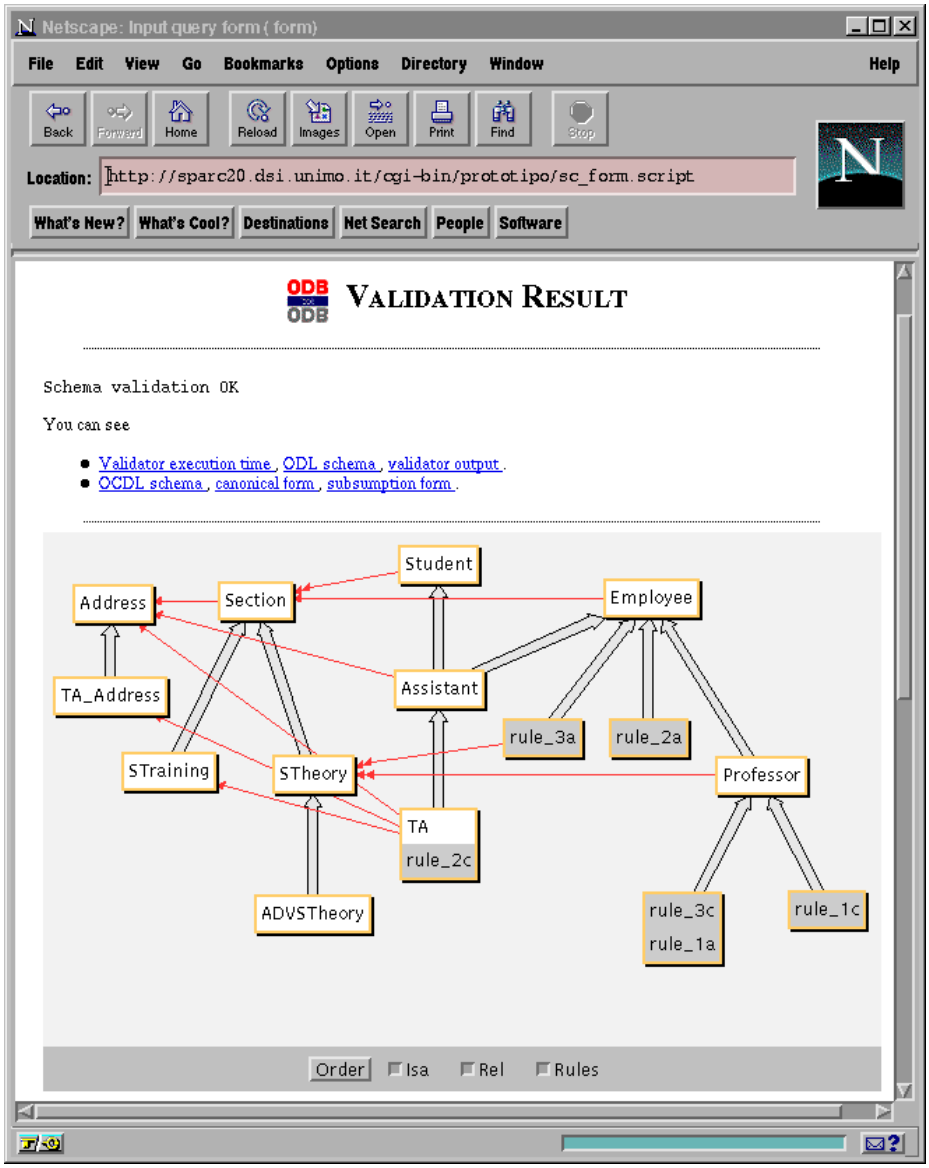

Fig. 5. Pagina di presentazione dello schema validato

sono stati utilizzati i sistemi O2 ver. 4.2 ed UniSQL ver. 3.1.2.

Per l'analisi sperimentale si è costruito un database relativo ad uno schema comprendente sei classi e, mediamente, due regole associate a ciascuna classe.

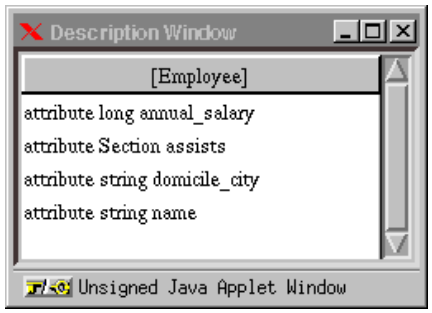

Fig. 6. Finestra di presentazione degli attributi di una classe

Sono state successivamente scelte cinque query legate allo schema per le quali si è determinata l'ottimizzazione semantica e relativo costo di trasformazione del prototipo. Per ciascuna interrogazione si è calcolato il tempo di esecuzione su quattro istanze di DB differenti, le cui statistiche sono riportate in tabella 2.

	Class A	**SubClass A**	**Class B**	**SubClass B**	**Class C**	**SubClass C**
DB1	164	56	1151	534	1700	950
DB2	1555	725	1718	691	2600	1481
DB3	3220	1621	3436	1382	4900	2912
DB4	4869	2515	5154	2073	8200	4668

Table 2. Dimensioni delle Classi del Database

Infine, per ciascun OODBMS, abbiamo riportato i valori medi del rapporto tra il costo di esecuzione per le query originali e quelle ottimizzate in funzione degli oggetti presenti nel database. Tali valori sono graficati in fig. 8: come è facilmente prevedibile il risparmio di costo è funzione crescente del numero di oggetti presenti nel DB. Tale considerazione è più evidente nel caso di UniSQL, mentre per O2 il risparmio di costo è da subito rilevante (già con DB1) per poi rimanere pressochè costante.

7 Conclusioni

In questo lavoro abbiamo presentato ODB-Tool, un sistema integrato per la validazione e l'ottimizzazione di interrogazioni in OODB. È stato presentato sia l'ambito teorico che lo stato attuale dell'implementazione del prototipo. In futuro prevediamo di migliorare la facilità d'uso da parte dell'utente, in particolare

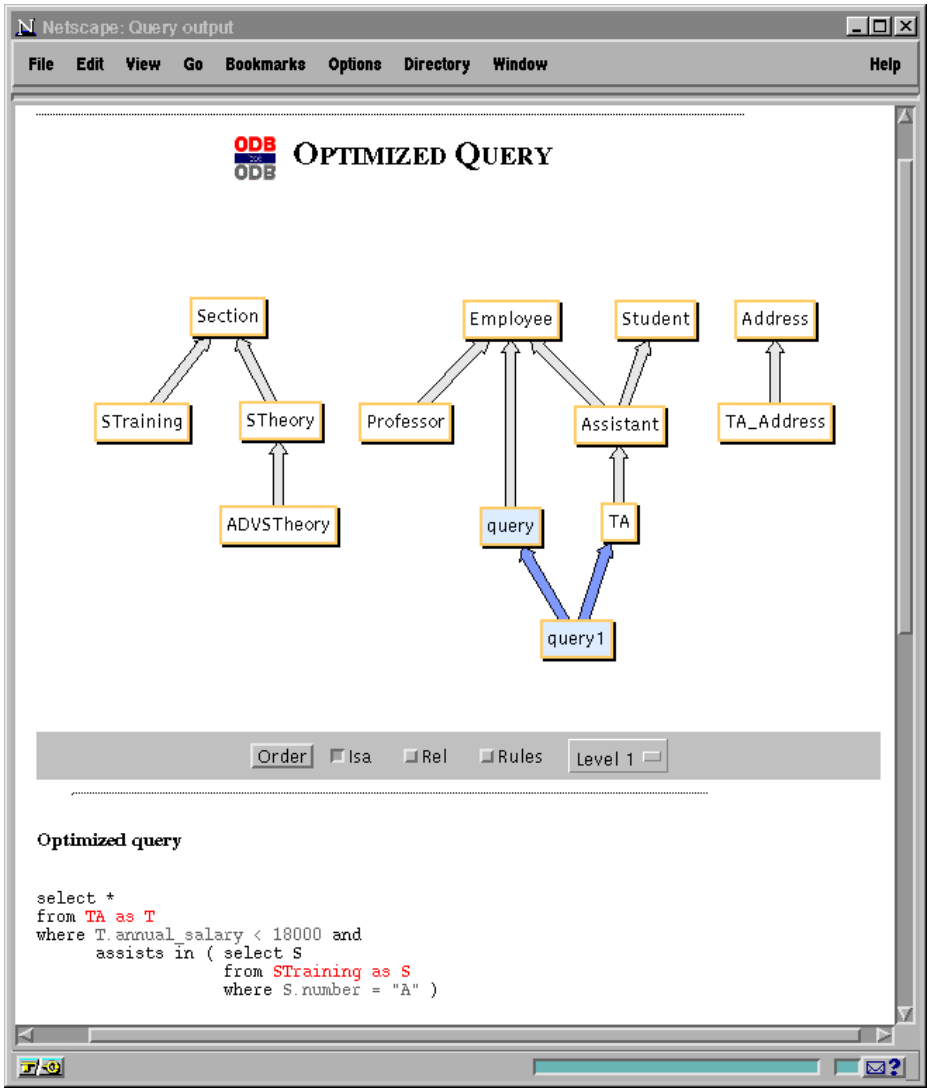

Fig. 7. Pagina di presentazione della query ottimizzata

riguardo alla rappresentazione grafica dello schema. Inoltre si procederà allo sviluppo, sia teorico che implementativo, del processo di valutazione dei fattori ottenuti dopo l'ottimizzazione, discriminando tra quelli ridondanti (parzialmente implementato) e quelli eliminabili perchè non rilevanti.

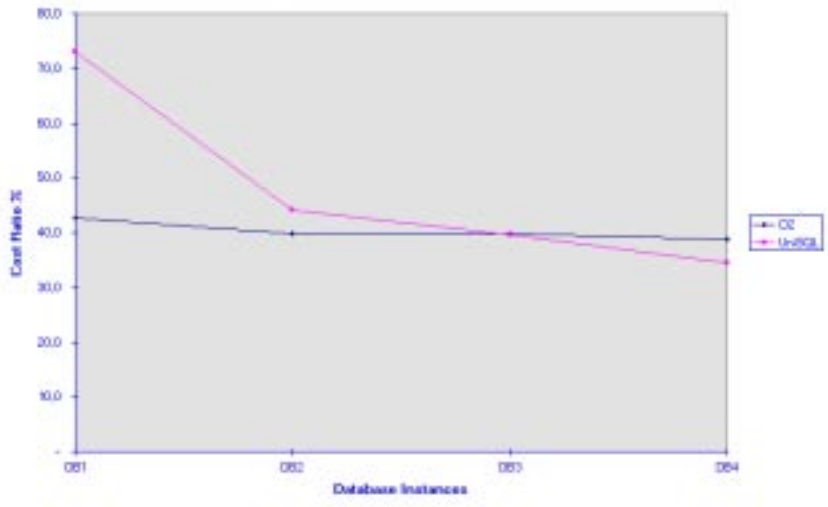

Fig. 8. Rapporto tra Costo Ottimizzato ed Originale in funzione della dimensione de DB

References

1. S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.
2. D. Beneventano J.P. Ballerini, S. Bergamaschi, and M. Vincini. Odb-qoptimizer: un ottimizzatore semantico di interrogazioni per oodb. In A. Albano, S. Salerno, F. Arcelli, M. Gaeta, S. Rizzo, and G. Vantini, editors, *Convegno su Sistemi Evoluti per Basi di Dati*, June 1995.
3. S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.
4. C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, Philadelphia, PA, 1989.
5. P. Atzeni, editor. *LOGIDATA$^+$: Deductive Databases with Complex Objects.* Springer-Verlag: LNCS n. 701, Heidelberg - Germany, 1993.
6. J. J. King. *Query optimization by semantic reasoning.* PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.
7. J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.
8. M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.
9. R.G.G. Cattell. *The Object Database Standard: ODMG-93, Release 1.1.* Morgan Kaufmann Publishers, Inc., 1994.
10. Jon Siegel et al. *CORBA Fundamentals and Programming.* John Wiley and Sons, New York, 1996.

11. D. Beneventano, S. Bergamaschi, A. Garuti, C. Sartori, and M. Vincini. Odb-reasoner: un ambiente per la verifica di schemi e l'ottimizzazione di interrogazioni in oodb. In E. Ricciardi F. Rabitti, G. Mainetto, editor, *Convegno su Sistemi Evoluti per Basi di Dati*, july 1996.
12. M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.

This article was processed using the LaTeX macro package with LLNCS style