ODB-Reasoner: un ambiente per la verifica di schemi e l'ottimizzazione di interrogazioni in OODB*

Domenico Beneventano[1] and Sonia Bergamaschi[1,2] and Alessandra Garuti[1] and Claudio Sartori[2,3] and Maurizio Vincini[1]

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

Convegno Nazionale Sistemi di Basi di Dati Evolute (SEBD95), Ravello, Italy. june 1995

Abstract.
In questo lavoro viene presentato ODB-Reasoner, un tool per l'acquisizione e la verifica di consistenza di schemi e per l'ottimizzazione semantica di interrogazioni nelle Basi di Dati Orientate agli Oggetti (OODB). L'ambiente teorico su cui è basato ODB-Reasoner include due ingredienti fondamentali. Il primo è *ODL* (*Object Description Logics*), proposto come formalismo comune per esprimere descrizioni di classi, vincoli d'integrità ed interrogazioni e dotato di tecniche di inferenza basate sul calcolo della *sussunzione* introdotte per le *Logiche Descrittive* nell'ambito dell'Intelligenza Artificiale. Il secondo ingrediente è costituito dal concetto di *espansione semantica* di un tipo e dalla sua computazione attraverso la sussunzione.

1 Introduzione

Il formalismo ODL è stato proposto in [1] ed estende l'espressività di linguaggi di logica descrittiva [2] al fine di rappresentare la semantica dei modelli di dati ad oggetti complessi (*CODMs*) recentemente proposti in ambito basi di dati deduttive [3] e basi di dati orientate agli oggetti [4, 5].

Le principali caratteristiche di ODL sono brevemente riportate nel seguito. In ODL viene assunta una ricca struttura per il *sistema di tipi atomici*; oltre ai tipi atomici *integer*, *boolean*, *string*, *real* e *tipi mono-valore*, viene considerata anche la possibilità di utilizzare dei sottoinsiemi di tipi atomici, come ad esempio intervalli di interi. Partendo da questi tipi atomici si possono creare *tipi valori*, attraverso gli usuali costruttori di tipo definiti nei CODMs, quali *tuple*

* Questa ricerca è stata parzialmente finanziata dal progetto M.U.R.S.T. 40% , "Metodi Formali e Strumenti per Basi di Dati Evolute".
e_mail: {benevent, garuti, sonia, vincini }@dsi.unimo.it, csartori@deis.unibo.it

ed *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 fra nomi per tipi–valore e nomi per tipi–classe (chiamati semplicemente *classi*). Ciò equivale a dire che i nomi di tipi sono partizionati fra quelli che rappresentano insiemi di oggetti e quelli che rappresentano insiemi di valori. L'ereditarietà, sia semplice che multipla, è espressa direttamente nella descrizione di una classe tramite l'operatore di *intersezione*.

ODL introduce inoltre la distinzione tra nomi di tipi *virtuali*, che descrivono condizioni necessarie e sufficienti di appartenenza di un oggetto del dominio ad un tipo, e nomi di tipi *base*, che descrivono condizioni necessarie di appartenenza. In altri termini, l'appartenenza di un elemento all'interpretazione di un nome di tipo base deve essere stabilita esplicitamente, mentre l'interpretazione dei nomi di tipo virtuale è calcolata.

In [6] ODL è stato esteso per permettere la formulazione dichiarativa di un insieme rilevante di vincoli d'integrità sulla base di dati. Gli schemi di basi di dati reali sono generalmente forniti in termini di classi base mentre l'ulteriore conoscenza è espressa attraverso vincoli d'integrità che garantiscono la consistenza dei dati. In particolare, le estensioni riguardano i *tipi path quantificati* e *regole d'integrità*. I path, che sono essenzialmente sequenze di attributi, rappresentano l'elemento centrale dei linguaggi d'interrogazione OODB per navigare attraverso le gerarchie di aggregazione di classi e tipi di uno schema. È possibile esprimere path *quantificati* per navigare attraverso gli attributi multivalore: le quantificazioni ammesse sono quella esistenziale e quella universale e possono comparire più di una volta nello stesso path. 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 espressioni di tipo ODL. È possibile, in tal modo, esprimere correlazioni fra proprietà strutturali della stessa classe o condizioni sufficienti per il popolamento di sottoclassi di una classe data.

In [1] è stato presentato il sistema ODL-Designer, per l'acquisizione e la validazione di schemi OODB che preserva la consistenza della tassonomia ed effettua inferenze tassonomiche. In particolare, il sistema prevede un algoritmo di *sussunzione*, che determina tutte le relazioni di *specializzazioni* tra tipi, e un algoritmo per rilevare tipi *inconsistenti*, cioè tipi necessariamente vuoti. In [6] l'ambiente teorico sviluppato in [1] è stato esteso per effettuare l'ottimizzazione semantica delle interrogazioni, dando vita al sistema ODB-QOptimizer.

Nel presente lavoro viene introdotto un sistema, denominato ODB-Reasoner, che estende con i tipi path quantificati e le regole d'integrità ODL-Designer ed integra ODB-QOptimizer. Nello sviluppo del sistema, particolare attenzione è stata dedicata al linguaggio di definizione dello schema in modo da renderlo compatibile con il linguaggio proposto dal gruppo di standardizzazione ODMG–93 [7]; tale linguaggio è stato integrato dall'espressione dei vincoli di integrità non ancora previsti nello standard.

La componente del sistema ODB-QOptimizer, che include un modulo per l'acquisizione di interrogazioni in un linguaggio SQL–like, si occupa in particolare dell'ottimizzazione semantica delle interrogazioni.

La nozione di ottimizzazione semantica di query è stata introdotta per le basi di dati relazionali da King [8, 9] e da Hammer e Zdonik [10]. L'idea di base di queste proposte è che i vincoli di integrità, espressi per forzare la consistenza di una base di dati, possono essere utilizzati anche per ottimizzare le interrogazioni fatte dall'utente, trasformando un'interrogazione in una *equivalente*, ovvero con la stessa risposta, ma che può essere elaborata in modo più efficiente.

Un insieme rilevante di query per OODB [11] può essere espresso come un tipo *virtuale*, vista la ricchezza del formalismo ODL. Tuttavia, poiché i linguaggi di interrogazione sono più espressivi del nostro formalismo, introduciamo, seguendo l'approccio proposto in [12], una separazione ideale di un'interrogazione in una parte *clean*, che può essere rappresentata come tipo in ODL, e una parte *dirty*, che va oltre l'espressività del sistema di tipi; l'ottimizzazione semantica sarà effettuata solo sulla parte *clean*.

Sia il processo di controllo di consistenza e classificazione delle classi dello schema che quello di ottimizzazione semantica delle interrogazioni sono basati sulla nozione di *espansione semantica* di un tipo. L'espansione semantica permette di incorporare ogni possibile restrizione che non è presente nel tipo originale ma che è *logicamente implicata* dallo schema globale (classi + tipi + regole d'integrità). L'espansione dei tipi è basata 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. La relazione di sussunzione è simile alla relazione di *raffinamento di tipi* (*subtyping relation* definita in [13] ed adottata negli OODBs [4]).

Il calcolo dell'espansione semantica di una classe permette di rilevare nuove relazioni *isa*, cioè relazioni che non sono esplicitate dal progettista ma che sono logicamente implicate dalla descrizione della classe e dello schema globale. In questo modo una classe può essere classificata automaticamente all'interno di una gerarchia di ereditarietà. La presenza delle regole di integrità rende questa classificazione significativa anche rispetto alle sole classi base, cioè con le regole si possono determinare nuove relazioni *isa* anche tra due classi base.

Per quanto riguarda l'ottimizzazione, seguendo l'approccio di [14], viene eseguita a run time l'espansione semantica di una interrogazione in modo da ottenere l'*interrogazione più specializzata* fra tutte quelle semanticamente equivalenti a quella iniziale. In questo modo l'interrogazione viene spostata verso il basso nella gerarchia delle classi e le classi presenti nell'interrogazione vengono sostituite con classi più specializzate; questo costituisce un'effettiva ottimizzazione dell'interrogazione, indipendente da ogni modello di costo, in quanto riduce l'insieme di oggetti da controllare per individuarne il risultato.

Il presente lavoro è organizzato come segue. In sezione 2 viene riportato un esempio che mostra sia l'attivita' di ODL-DESIGNER che quella di ODB-QOPTIMIZER. In particolare, sono anche riportate alcune idee preliminari relative all'eliminazione dall'interrogazione di fattori ridondanti; questa attività risulta utile

Classi dello Schema

```
interface Material
{
   attribute String name;
   attribute Integer risk;
   attribute Set <String> feature;
};
interface DMaterial: Material
{
   attribute Range [15->] risk;
};
interface SMaterial: Material{ };
interface Storage
{
   attribute String category;
   attribute Manager managed-by;
   attribute
   Set<Struct {
        attribute Material item;
        attribute Range [10->300] qty; }
      > stock;
   attribute Integer maxrisk;
};
```

```
interface Manager
{
   attribute String name;
   attribute Range [40K->] salary;
   attribute Range [1->15] level;
};
interface TManager: Manager
{
   attribute Range [8->12] level;
};

interface SStorage: Storage{ };
interface DStorage: Storage
{
   attribute Set< Struct {
        attribute DMaterial item; }
        > stock;
};
```

Vincoli di integrità

```
Ruledef (Material) & (Range [10->] ^.risk)         -> SMaterial                 R1;
Ruledef (Storage) & (SMaterial ^.stock.Set.item)   -> SStorage                  R2;
Ruledef (Storage) & (Range [6->12]                 -> (^.category = "A2")       R3;
                         ^.managed-by^.level)
Ruledef (Storage) & (^.category = "A2" ) &         -> (TManager managed-by)     R4;
        (Range [10->] ^.stock.ESet.item^.risk)
```

Tabella 1. Lo schema del Magazzino in sintassi ODMG–93

sia per l'ottimizzazione, in quanto l'eliminazione di un fattore può ridurre il numero di classi da visitare per risolvere l'interrogazione, sia per filtrare dall'interrogazione semanticamente estesa i fattori inutili. In sezione 3 viene brevemente riportato il formalismo ODL esteso con i path ed i vincoli di integrità e vengono formalizzate la sussunzione e l'espansione semantica dei tipi. In sezione 4 viene descritto il sistema ODB-Reasoner ed in sezione 5 è mostrato un esempio di funzionamento del sistema. Infine le sezioni 6 e 7 compiono un confronto con altri lavori e mostrano le evoluzioni future.

2 Esempio: il dominio Magazzino

Per illustrare il nostro metodo, consideriamo il seguente esempio relativo alla struttura organizzativa di una società (vedi Tabella 1 per la sua descrizione nella sintassi basata sullo standard ODMG–93): i materiali (`Material`) sono descritti da un nome, un rischio e da un insieme di caratteristiche; gli `SMaterial` sono un sottoinsieme dei `Material`. I `DMaterial` (materiali "pericolosi") sono un sottoinsieme dei `Material`, caratterizzati da un rischio superiore a 15. I manager hanno un nome, un salario superiore ai 40K dollari e un livello compreso tra 1 e 15. I `TManager` sono manager che hanno un livello compreso tra 8 e 12. I magazzini (`Storage`) sono descritti da una categoria, sono diretti (`managed-by`) da un manager e contengono (`stock`) un insieme di articoli (`item`) che sono dei materiali, per ciascuno dei quali è indicata la quantità presente; è inoltre riportato il rischio massimo (`maxrisk`) ammissibile per i materiali conservati. Gli `SStorage` sono un sottoinsieme dei magazzini. Infine vi sono i magazzini "pericolosi" (`DStorage`) che sono un sottoinsieme degli `Storage` caratterizzati dallo stoccaggio di soli materiali "pericolosi" (`DMaterial`). Si noti che le classi che abbiamo descritto sono tutte di tipo "base", esprimono cioè solo condizioni necessarie per l'appartenenza di un oggetto alla classe. A queste classi vengono aggiunti i vincoli di integrità (espressi in Tabella 1 come regole *if then*, introdotte dalla parola chiave **Ruledef**) per garantire la consistenza della base di dati. La regola R1 stabilisce che, per un `Material`, avere un rischio superiore a 10 è condizione sufficiente per essere considerato anche un `SMaterial`. Nello stesso modo, la regola R2 forza gli `Storage` in cui *tutti* gli articoli immagazzinati sono `SMaterial` ad essere anche `SStorage`. R3 dice che i magazzini guidati da un manager con livello compreso tra 6 e 12 debbono appartenere alla categoria "A2". Infine, R4 forza i magazzini con categoria di tipo "A2" e contenenti *almeno* un materiale con rischio superiore a 10 ad essere guidati da un `TManager`.

L'attività di inferenza iniziale da parte del sistema, effettuata da ODL-Designer, consiste nel determinare l'espansione semantica per ogni classe dello schema. Questo permette di rilevare relazioni *isa* non esplicitate nella definizione delle classi stesse. Ad esempio, nel calcolo dell'espansione semantica della classe `DMaterial` si rileva che tale classe è sussunta (implica) l'antecedente della regola R1 quindi congiungendone il conseguente si determina che `DMaterial` è una specializzazione di `SMaterial`. Nel caso in cui vengono applicate più regole durante l'espansione semantica di una classe, le relazioni *isa* ricavate sono meno ovvie di quella appena descritta. Ad esempio, nel calcolo dell'espansione semantica della classe `DStorage` si applica prima la regola R1 e poi la regola R2, concludendo che `DStorage` è una specializzazione di `SStorage`. Si noti che queste due nuove relazioni di specializzazioni riguardano classi base.

L'altra componente del sistema, ODB-QOptimizer, si occupa dell'ottimizzazione semantica delle interrogazioni a run time; anche l'ottimizzazione dell'interrogazione è basata sull'espansione semantica, in modo da specializzare le classi interessate dall'interrogazione. Consideriamo ad esempio la seguente interrogazione:

Q_1 : "Seleziona i magazzini che contengono materiali che hanno tutti rischio maggiore o uguale a 15".

Nel nostro sistema la query puo' essere scritta in linguaggio SQL–like nel seguente modo (dove il simbolo ^ indica il riferimento ad oggetti):

select * **from** `Storage`
where for all `stock` : `item` = `Material` **and** `^risk` $\geq$ 15 ;

L'espansione semantica di quest'interrogazione, che si ottiene applicando prima la regola R1 e poi R2, porta alla seguente interrogazione equivalente:

select * **from** `SStorage`
where for all `stock` : `item` = `SMaterial` **and** `^risk` $\geq$ 15 ;

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 `Storage` con `SStorage` e `Material` con `SMaterial` 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_1' : "Seleziona i magazzini che contengono materiali che hanno tutti rischio maggiore o uguale a 15 e nei quali è presente almeno un materiale con rischio superiore al massimo ammissibile per il magazzino".

select * **from** `Storage`
where for all `stock` : `item` = `Material` **and** `^risk` $\geq$ 15 **and**
exists `stock` : `item.risk`> `Storage.maxrisk` ;

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

Un'altra forma di ottimizzazione altrettanto generale può essere ottenuta riducendo il numero di classi coinvolte nell'interrogazione. In letteratura [14, 15] questo problema è generalmente noto come *rimozione di vincoli*, cioè eliminazione di fattori ridondanti all'interno di una interrogazione.

Nel nostro approccio i fattori eliminabili di un'interrogazione, cioè i fattori che sono logicamente implicati dagli altri fattori dell'interrogazione, possono essere individuati tramite la sussunzione. Ad esempio, consideriamo la seguente interrogazione:

Q_2 : "Seleziona i magazzini di categoria A2 diretti da un `TManager`".

select * **from** `Storage`
where `category` = "A2" **and**
`managed-by` = `TManager`;

In base alla regola R4 e alla descrizione di `TManager` si ottiene che il fattore (`category` = "A2") è implicato da (sussume) il resto dell'interrogazione e quindi può essere eliminato da Q_2.

Nei citati lavori sulla rimozione di vincoli, la ridondanza di un fattore è controllata rispetto all'interrogazione originale; questo controllo costituisce però soltanto una condizione sufficiente per l'eliminazione di un fattore. Allo scopo di ottenere condizioni necessarie e sufficienti noi proponiamo di controllare la ridondanza di un fattore rispetto all'espansione semantica dell'interrogazione. Questo è mostrato tramite il seguente esempio:

Q_3 : "Seleziona i magazzini diretti da un `TManager` e che contengono almeno un materiale con rischio maggiore o uguale a 10".

select * **from** `Storage`
where `managed-by` = `TManager` **and**
exists `stock` : `item^risk` ≥ 10 ;

Analizzando l'eliminabilità del fattore (`managed-by` = `TManager`) si rileva che esso non sussume il resto dell'interrogazione e quindi non è ridondante rispetto a Q_3. Comunque, considerando l'espansione semantica dell'interrogazione (ottenuta applicando le regole R1, R3 e R4):

select * **from** `Storage`
where `category` = "A2" **and**
`managed-by` = `TManager` **and**
exists `stock` : `item` = `SMaterial` **and** `^risk` ≥ 10 ;

il fattore (`managed-by` = `TManager`) risulta ridondante, in quanto sussume il resto dell'interrogazione; tale fattore può essere quindi eliminato, evitando l'accesso alla classe `TManager`.

Lo sviluppo futuro del sistema prevede di determinare i fattori ridondanti di una interrogazione direttamente dall'analisi della sua espansione semantica interrogazione e dalle regole applicate per ottenerle. Intuitivamente, in entrambi i due esempi di eliminazione, considerati il fattore eliminabile corrisponde ad un conseguente di una delle regole applicate durante il calcolo dell'espansione semantica dell'interrogazione.

3 ODL esteso con i vincoli di integrità

In questa sezione viene brevemente riportato il formalismo ODL introdotto in [1] ed esteso in [6] con i vincoli di integrità e vengono formalmente definite la sussunzione e l'espansione semantica.

3.1 Schema e Istanza del Database

Sia $\mathbf{D}$ l'insieme infinito numerabile dei valori atomici (che saranno indicati con $d_1, d_2, \ldots$), e.g., l'unione dell'insieme degli interi, delle stringhe e dei booleani. Sia $\mathbf{B}$ l'insieme di designatori di tipi atomici, con $\mathbf{B} = \{$`Int`, `String`, `Bool`, `Real`, i_1–j_1, i_2–j_2, $\ldots$, $d_1, d_2, \ldots\}$, dove i d_k indicano tutti gli elementi di `Int` $\cup$ `String` $\cup$ `Bool` e dove gli i_k–j_k indicano tutti i possibili intervalli di interi (i_k può essere $-\infty$ per denotare il minimo elemento di `Int` e j_k può essere $+\infty$ per denotare il massimo elemento di `Int`).

Sia $\mathbf{A}$ un insieme numerabile di *attributi* (denotati da $a_1, a_2, \ldots$) e $\mathcal{O}$ un insieme numerabile di *identificatori di oggetti* (denotati da $o, o', \ldots$) disgiunti da $\mathbf{D}$. Si definisce l'insieme $\mathcal{V}(\mathcal{O})$ dei *valori su* $\mathcal{O}$ (denotati da v, v') come segue (assumendo $p \geq 0$ e $a_i \neq a_j$ per $i \neq j$):

$$v \;\rightarrow d \mid o \mid \{v_1, \ldots, v_p\} \mid [a_1 : v_1, \ldots, a_p : v_p]$$

Gli identificatori di oggetti sono associati a valori tramite una *funzione totale* δ da $\mathcal{O}$ a $\mathcal{V}(\mathcal{O})$; in genere si dice che il valore $\delta(o)$ è lo *stato* dell'oggetto identificato dall'oid o. Sia $\mathbf{N}$ l'insieme numerabile di *nomi di tipi* (denotati da $N, N', \ldots$) tali che $\mathbf{A}$, $\mathbf{B}$, e $\mathbf{N}$ siano a due a due disgiunti. $\mathbf{N}$ è partizionato in tre insiemi $\mathbf{C}$, $\mathbf{V}$ e $\mathbf{T}$, dove $\mathbf{C}$ consiste di nomi per *tipi-classe base* $(C, C' \ldots)$, $\mathbf{V}$ consiste di nomi per *tipi-classe virtuali* $(V, V' \ldots)$, e $\mathbf{T}$ consiste di nomi per *tipi-valori* $(T, T', \ldots)$. Un *path* p è una sequenza di elementi $p = e_1 . e_2 . \ldots . e_n$, con $e_i \in \mathbf{A} \cup \{\triangle, \forall, \exists\}$. Con ϵ si indica il path vuoto. $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$[4] indica l'insieme di tutte le *descrizioni di tipo finite* $(S, S', \ldots)$, dette brevemente *tipi*, su di un dato $\mathbf{A}, \mathbf{B}, \mathbf{N}$, ottenuto in accordo con la seguente regola sintattica:

$$S \to \top \mid B \mid N \mid [a_1 : S_1, \ldots, a_k : S_k] \mid \forall\{S\} \mid \exists\{S\} \mid \ \triangle\, S \mid S \sqcap S' \mid (p : S)$$

$\top$ denota il *tipo universale* e rappresenta tutti i valori; [ ] denota il costruttore di tupla. $\forall\{S\}$ corrisponde al comune costruttore di insieme e rappresenta un insieme i cui elementi sono *tutti* dello stesso tipo S. Invece, il costruttore $\exists\{S\}$ denota un insieme in cui *almeno* un elemento è di tipo S. Il costrutto $\sqcap$ indica la *congiunzione*, mentre $\triangle$ è il costruttore di oggetto. Il tipo $(p\colon S)$ è detto *tipo path* e rappresenta una notazione abbreviata per i tipi ottenuti con gli altri costruttori.

Dato un dato sistema di tipi $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, uno *schema* σ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ è una funzione totale da $\mathbf{N}$ a $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, che associa ai nomi di tipi la loro descrizione. Diremo che un nome di tipo N *eredita* da un altro nome di tipo N', denotato con $N \prec\!\!| N'$, se $\sigma(N) = N' \sqcap S$. Si richiede che la relazione di ereditarietà sia priva di cicli, i.e., la chiusura transitiva di $\prec\!\!|$, denotata $\prec$, sia un ordine parziale stretto.

Dato un dato sistema di tipi $\mathbf{S}$, una *regole di integrità* R è espressa nella forma $R = S^a \to S^c$, dove S^a e S^c rappresentano rispettivamente l'antecedente e il conseguente della regola R, con $S^a, S^c \in \mathbf{S}$. Una regola R esprime il seguente vincolo: per tutti gli oggetti v, se v è di tipo S^a allora v deve essere di tipo S^c. Con $\mathbf{R}$ si denota un insieme finito di regole.

Uno *schema con regole* è una coppia $(\sigma, \mathbf{R})$, dove σ è uno schema e $\mathbf{R}$ un insieme di regole. L'esempio del Magazzino rappresentato in Tabella 1 è descritto in ODL tramite lo schema con regole mostrato in Tabella 2.

La *funzione interpretazione* $\mathcal{I}$ è una funzione da $\mathbf{S}$ a $2^{\mathcal{V}(\mathcal{O})}$ tale che: $\mathcal{I}[\top] = \mathcal{V}(\mathcal{O})$, $\mathcal{I}[B] = \mathcal{I}_{\mathbf{B}}[B]$[5] , $\mathcal{I}[C] \subseteq \mathcal{O}$, $\mathcal{I}[V] \subseteq \mathcal{O}$, $\mathcal{I}[T] \subseteq \mathcal{V}(\mathcal{O}) - \mathcal{O}$. L'interpretazione è estesa agli altri tipi come segue:

$$\mathcal{I}[[a_1\colon S_1, \ldots, a_p\colon S_p]] = \Big\{ [a_1\colon v_1, \ldots, a_q\colon v_q] \Big| p \leq q, v_i \in \mathcal{I}[S_i], 1 \leq i \leq p, \; v_j \in \mathcal{V}(\mathcal{O}), p+1 \leq j \leq q \Big\}$$

$$\mathcal{I}[\forall\{S\}] = \Big\{ \{v_1, \ldots, v_p\} \Big| v_i \in \mathcal{I}[S], 1 \leq i \leq p \Big\}$$

---

[4] In seguito, scriveremo $\mathbf{S}$ in luogo di $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ quando i componenti sono ovvi dal contesto.

[5] Assumendo $\mathcal{I}_{\mathbf{B}}$ funzione di *interpretazione standard* da $\mathbf{B}$ a $2^{\mathbf{B}}$ tale che per ogni $d \in \mathbf{D} : \mathcal{I}_{\mathbf{B}}[d] = \{d\}$.

$\mathbf{C} = \{\texttt{Material}, \texttt{SMaterial}, \texttt{DMaterial}, \texttt{Storage}, \texttt{SStorage}, \texttt{DStorage}, \texttt{Manager}, \texttt{TManager}\}$,
$\mathbf{V} = \emptyset, \quad \mathbf{T} = \emptyset$

$$\sigma \begin{cases} \sigma(\texttt{Material}) & = \triangle[\texttt{name}\colon \texttt{String}, \texttt{risk}\colon \texttt{Int}, \texttt{feature}\colon \{\texttt{String}\}] \\ \sigma(\texttt{SMaterial}) & = \texttt{Material} \\ \sigma(\texttt{DMaterial}) & = \texttt{Material} \sqcap \triangle[\texttt{risk}\colon 15 \div \infty] \\ \sigma(\texttt{Storage}) & = \triangle[\texttt{managed-by}\colon \texttt{Manager}, \texttt{category}\colon \texttt{String}, \texttt{maxrisk}\colon \texttt{Int}, \\ & \qquad \texttt{stock}\colon \{[\texttt{item}\colon \texttt{Material}, \texttt{qty}\colon 10 \div 300]\}] \\ \sigma(\texttt{SStorage}) & = \texttt{Storage} \\ \sigma(\texttt{DStorage}) & = \texttt{Storage} \sqcap \triangle[\texttt{stock}\colon \{[\texttt{item}\colon \texttt{DMaterial}]\}] \\ \sigma(\texttt{Manager}) & = \triangle[\texttt{name}\colon \texttt{String}, \texttt{salary}\colon 40K \div \infty, \texttt{level}\colon 1 \div 15] \\ \sigma(\texttt{TManager}) & = \texttt{Manager} \sqcap \triangle[\texttt{level}\colon 8 \div 12] \end{cases}$$

$$\mathbf{R} \begin{cases} R_1 : & \texttt{Material} \sqcap (\triangle.\texttt{risk}\colon 10 \div \infty) \rightarrow \texttt{SMaterial} \\ R_2 : & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\forall.\texttt{item}\colon \texttt{SMaterial}) \rightarrow \texttt{SStorage} \\ R_3 : & \texttt{Storage} \sqcap (\triangle.\texttt{managed-by}.\texttt{level}\colon 6 \div 12) \rightarrow (\triangle.\texttt{category}\colon \text{"}A2\text{"}) \\ R_4 : & \texttt{Storage} \sqcap (\triangle.\texttt{category}\colon \text{"}A2\text{"}) \sqcap (\triangle.\texttt{stock}.\exists.\triangle.\texttt{item}.\texttt{risk}\colon 10 \div \infty) \\ & \qquad \rightarrow \triangle[\texttt{managed-by}\colon \texttt{TManager}] \end{cases}$$

Tabella 2. Schema con Regole di Integrità in linguaggio ODL

$$\begin{aligned} \mathcal{I}[\exists\{S\}] &= \left\{ \{v_1, \ldots, v_p\} \middle| \exists i, 1 \leq i \leq p, v_i \in \mathcal{I}[S] \right\} \\ \mathcal{I}[\triangle S] &= \left\{ o \in \mathcal{O} \middle| \delta(o) \in \mathcal{I}[S] \right\} \\ \mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S'] \end{aligned}$$

Per i tipi cammino abbiamo $\mathcal{I}[(p\colon S)] = \mathcal{I}[(e\colon (p'\colon S))]$ se $p = e.p'$ dove

$$\mathcal{I}[(\epsilon\colon S)]=\mathcal{I}[S],\ \mathcal{I}[(a\colon S)]=\mathcal{I}[[a\colon S]],\ \mathcal{I}[(\triangle\colon S)]=\mathcal{I}[\triangle S],$$

$$\mathcal{I}[(\forall\colon S)]=\mathcal{I}[\forall\{S\}],\ \mathcal{I}[(\exists\colon S)]=\mathcal{I}[\exists\{S\}]$$

Quindi il tipo path è un'abbreviazione per un altro tipo: ad esempio, il tipo path $(\triangle.\texttt{stock}.\forall.\texttt{item}\colon \texttt{SMaterial})$ è equivalente a $\triangle[\texttt{stock}\colon \forall\{[\texttt{item}\colon \texttt{SMaterial}]\}]$.

Si introduce ora la nozione di istanza legale di uno schema con regole come una interpretazione nella quale un valore istanziato in un nome di tipo ha una descrizione corrispondente a quella del nome di tipo stesso e dove sono valide le relazioni di inclusioni stabilite tramite le regole.

Definizione 1 Istanza Legale. Una funzione di interpretazione $\mathcal{I}$ è una *istanza legale* di uno schema con regole $(\sigma, \mathbf{R})$ sse l'insieme $\mathcal{O}$ è *finito* e per ogni $C \in \mathbf{C}, V \in \mathbf{V}, T \in \mathbf{T}, R \in \mathbf{R} : \mathcal{I}[C] \subseteq \mathcal{I}[\sigma(C)]$, $\mathcal{I}[T] = \mathcal{I}[\sigma(T)]$, $\mathcal{I}[V] = \mathcal{I}[\sigma(V)]$, $\mathcal{I}[S^a] \subseteq \mathcal{I}[S^c]$.

Si noti come, in una istanza legale $\mathcal{I}$, l'interpretazione di un nome di classe base *è contenuta* nell'interpretazione della sua descrizione, mentre per un nome di classe virtuale, come per un nome di tipo-valore, l'interpretazione *coincide* con l'interpretazione della sua descrizione. In altri termini, mentre l'interpretazione di una classe base è fornita dall'utente, l'interpretazione di una classe virtuale è calcolata sulla base della sua descrizione. In presenza di classi virtuali cicliche questa computazione non è univoca: fissata un'interpretazione per le classi base, una classe virtuale ciclica può avere più di una interpretazione possibile. Tra tutte queste interpretazioni, noi selezioniamo come istanza legale quella *più grande*, cioè adottiamo una semantica di *massimo punto fisso*. Le definizioni formali di tale semantica e le motivazioni della scelta sono discusse in [1]; in questa sede si vuole soltanto evidenziare il fatto che il modello ammette delle classi virtuali cicliche e quindi il metodo di ottimizzazione proposto è applicabile anche alle query cicliche o ricorsive [16] .

3.2 Sussunzione ed Espansione Semantica di un tipo

Introduciamo la relazione di sussunzione in uno schema con regole.

Definizione 2 Sussunzione. Dato uno schema con regole $(\sigma, \mathbf{R})$, la *relazione di sussunzione* rispetto a $(\sigma, \mathbf{R})$, scritta $S \sqsubseteq_{\mathrm{R}} S'$ per ogni coppia di tipi $S, S' \in \mathbf{S}$, è data da: $S \sqsubseteq_{\mathrm{R}} S'sse\mathcal{I}[S] \subseteq \mathcal{I}[S']$ per tutte le istanze legali $\mathcal{I}$ di $(\sigma, \mathbf{R})$.

Segue immediatamente che $\sqsubseteq_{\mathrm{R}}$ è un preordine (i.e., transitivo e riflessivo ma antisimmetrico) che induce una *relazione di equivalenza* $\simeq_{\mathrm{R}}$ sui tipi: $S \simeq_{\mathrm{R}} S'$ sse $S \sqsubseteq_{\mathrm{R}} S'$ e $S' \sqsubseteq_{\mathrm{R}} S$. Diciamo, inoltre, che un tipo S è *inconsistente* sse $S \simeq_{\mathrm{R}} \perp$, cioè per ciascun dominio l'interpretazione del tipo è sempre vuota.

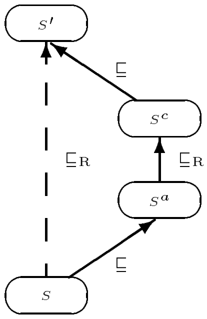
È importante notare che la relazione di sussunzione rispetto al solo schema σ, cioè considerando $\mathbf{R} = \emptyset$, denotata con $\sqsubseteq$, è simile alle relazioni di *subtyping* o *refinement* tra tipi definite nei *CODMs* [5, 4]. Questa relazione può essere calcolata attraverso una comparazione sintattica sui tipi; per il nostro modello tale l'algoritmo è stato presentato in [1].

È immediato verificare che, per ogni S, S', se $S \sqsubseteq S'$ allora $S \sqsubseteq_{\mathrm{R}} S'$. Comunque, il viceversa, in generale, non vale, in quanto le regole stabiliscono delle relazioni di inclusione tra le interpretazioni dei tipi che fanno sorgere *nuove* relazioni di sussunzione. Intuitivamente, come viene mostrato nell'esempio di figura 1, se $S \sqsubseteq S^a$ e $S^c \sqsubseteq S'$ allora $S \sqsubseteq_{\mathrm{R}} S'$.

La relazione esistente tra $\sqsubseteq_{\mathrm{R}}$ e $\sqsubseteq$ può essere espressa formalmente attraverso la nozione di *espansione semantica*.

Definizione 3 Espansione Semantica. Dato uno schema con regole $(\sigma, \mathbf{R})$ e un tipo $S \in \mathbf{S}$, l'*espansione semantica* di S rispetto a $\mathbf{R}$, EXP(S), è un tipo di $\mathbf{S}$ tale che:

1. $EXP(S) \simeq_{\mathrm{R}} S$;
2. per ogni $S' \in \mathbf{S}$ tale che $S' \simeq_{\mathrm{R}} S$ si ha $EXP(S) \sqsubseteq S'$.



$$
\begin{aligned}
S &= \texttt{Manager} \sqcap (\triangle.\texttt{level}\colon 5 \div 8)\\
S' &= (\triangle.\texttt{salary}\colon 30 \div 70)\\
R_1 &= \texttt{Manager} \sqcap (\triangle.\texttt{level}\colon 5 \div 10)\\
&\quad\rightarrow (\triangle.\texttt{salary}\colon 40 \div 60)
\end{aligned}
$$

$$S \sqsubseteq_{\mathrm{R}} S', \quad S \not\sqsubseteq S'$$

Fig. 1. Relazione di sussunzione dovuta alle regole

In altri termini, EXP(S) è il tipo più specializzato (rispetto alla relazione $\sqsubseteq$) tra tutti i tipi $\simeq_{\mathrm{R}}$–equivalenti al tipo S.

L'espressione $EXP(S)$ permette di esprimere la relazione esistente tra $\sqsubseteq_{\mathrm{R}}$ e $\sqsubseteq$: per ogni $S, S' \in \mathbf{S}$ si ha $S \sqsubseteq_{\mathrm{R}} S'$ se e solo se $EXP(S) \sqsubseteq S'$. Questo significa che, dopo aver determinato l'espansione semantica, anche la relazione di sussunzione nello schema con regole può essere calcolata tramite l'algoritmo presentato in [1].

È facile verificare che, per ogni $S \in \mathbf{S}$ e per ogni $R \in \mathbf{R}$, se $S \sqsubseteq (p : S^a)$ allora $S \sqcap (p\colon S^c) \simeq_{\mathrm{R}} S$. Questa trasformazione di S in $S \sqcap (p\colon S^c)$ è la base del calcolo della $EXP(S)$: essa viene effettuata iterativamente, tenendo conto che l'applicazione di una regola può portare all'applicazione di altre regole. Per individuare tutte le possibili trasformazioni di un tipo implicate da uno schema con regole $(\sigma, \mathbf{R})$, si definisce la funzione totale $\Gamma : \mathbf{S} \longrightarrow \mathbf{S}$, come segue:

$$\Gamma(S) = \begin{cases} S \sqcap (p : S^c) & \text{se esistono } R \text{ e } p \text{ tali che } S \sqsubseteq (p\colon S^a) \text{ e } S \not\sqsubseteq (p\colon S^c) \\ S & \text{altrimenti} \end{cases}$$

e poniamo $\tilde{\Gamma} = \Gamma^i$, dove i è il più piccolo intero tale che $\Gamma^i = \Gamma^{i+1}$. L'esistenza di i è garantita dal fatto che il numero di regole è finito e una regola non può essere applicata più di una volta con lo stesso cammino ($S \not\sqsubseteq (p\colon S^c)$). Si può dimostrare che, per ogni $S \in \mathbf{S}$, EXP(S) è effettivamente calcolabile tramite $\tilde{\Gamma}(S)$.

4 ODB-Reasoner: il tool

ODB-Reasoner è un sistema software per la validazione degli schemi di database e l'ottimizzazione delle interrogazioni che utilizza l'espansione semantica. ODB-Reasoner si compone di due prototipi software: ODL-Designer ed ODB-QOptimizer sviluppati entrambi presso il Dipartimento di Scienze dell'Ingegneria (DSI) dell'Università di Modena in linguaggio C, gcc gnu compiler, su piattaforma hardware SUN SPARCstation 20, sistema operativo Solaris 2.x.

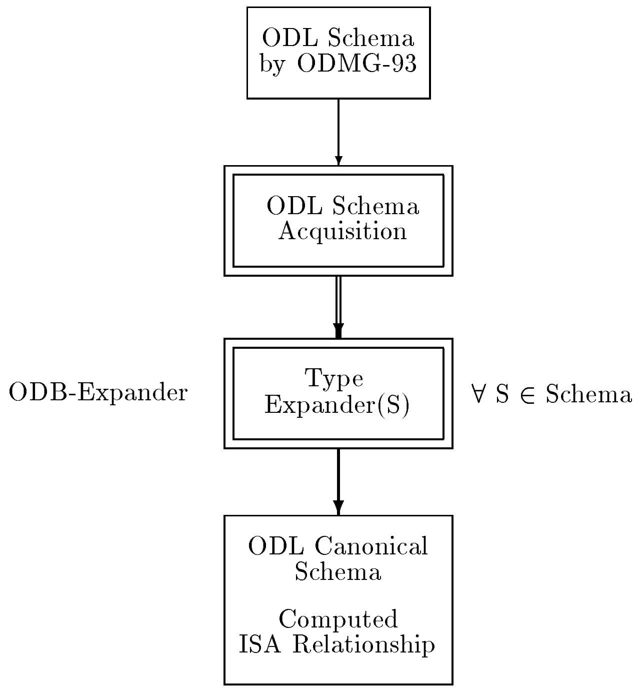


Fig. 2. Architettura funzionale di ODL-Designer

- ODL-Designer, la cui architettura è mostrata in figura 2 realizza la fase di precompilazione dello schema descritta in sezione 1: lo schema delle classi e dei vincoli di integrità viene descritto dal progettista nella sintassi basata su ODMG–93 e portato in input ad un interprete che lo converte nella corrispondente rappresentazione in forma ODL. A questo punto può partire la fase di espansione realizzata da Type Expander che restituisce lo schema compilato contenente tutte le relazioni *isa* che sono implicitamente definite dai vincoli.
- A run time possiamo eseguire ODB-QOptimizer (vedi fig 3 per l'architettura) che realizza l'ottimizzazione di una interrogazione. La query viene portata in input al sistema nel linguaggio SQL–like insieme al nuovo schema ottenuto grazie ad ODL-Designer. Nuovamente un interprete compie la trasformazione delle classi e della query nel modello ODL per poter invocare l'esecuzione dell'espansione semantica della query (tramite Type Expander) che fornisce come risultato finale l'interrogazione ottimizzata.

In figura 4 è illustrata l'architettura di Type Expander, che è il modulo che genera l'espansione semantica del generico tipo S ricevuto come input. L'algoritmo di calcolo (vedi algoritmo 1) è basato sulla definizione di $EXP(S)$ e di $\Gamma(S)$ vista in sezione 3 e sulla nozione di forma canonica di un tipo definita in [6]: in sintesi, nella forma canonica (denotata ν) viene assegnato un nome ad

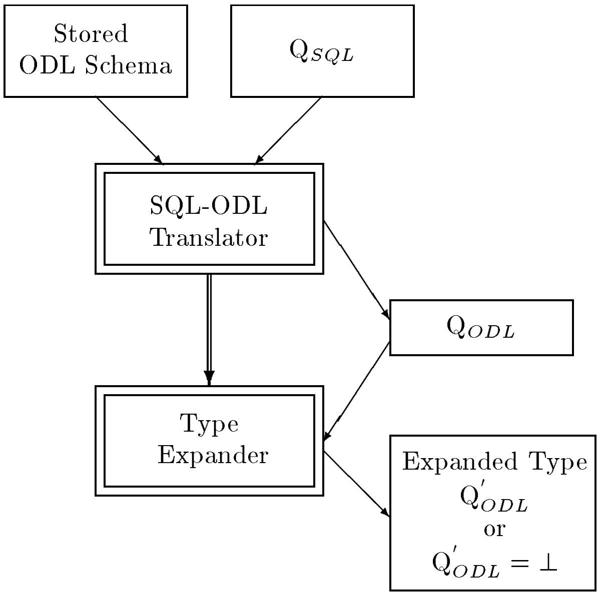


Fig. 3. Architettura funzionale di ODB-QOPTIMIZER

ogni tipo, vengono risolte le congiunzioni e una *classe base* $\mathbf{C}$, con $\sigma(\mathbf{C}) = S$, viene trasformata nella *classe virtuale* $\mathbf{V}$, con $\sigma(\mathbf{V}) = \overline{\mathbf{C}} \sqcap \sigma(S)$, dove $\overline{\mathbf{C}}$ è una *classe base* fittizia con descrizione $\sigma(\overline{\mathbf{C}}) = \triangle\top$.

Nell'algoritmo, a partire dal tipo S, ne viene rilevata la consistenza tramite la trasformazione in forma canonica, vengono calcolate le relazioni di sussunzione e viene generato il primo passo $\Gamma^{i+1}(S)$ (con i = 0) della funzione $\Gamma(S)$. Ora $\Gamma^{i+1}(S)$ diviene il tipo in input per l'iterazione del procedimento descritto che termina quando è stata calcolata l'espansione semantica $EXP(S)$ $(= \tilde{\Gamma}(S))$ oppure viene rilevata l'incoerenza sul tipo $\Gamma^{i+1}(S)$.

Prima di presentare l'algoritmo introduciamo la notazione di GS che denota il Generalization Set, cioè l'insieme dei tipi generalizzazione di un dato tipo, calcolato dall'algoritmo di sussunzione; e $\mathbf{CT}^+(S)$ che denota la chiusura di un tipo S, intendendo l'insieme dei tipi che definiscono ad ogni livello di innestamento la descrizione del tipo stesso.

Algoritmo 1 Espansione Semantica.

- **Acquisizione del tipo** S**:** $((\nu(S_1), GS(S_1)\ , \forall\ S_1 \in \mathbf{CT}^+(S)\ )$
- **Iterazione**: $\forall\ S_1 \in \mathbf{CT}^+(S)$, calcolo dell'espansione semantica di S_1, $\tilde{\Gamma}(S_1)$:

$$\Gamma^i(S_1) = \{ \begin{array}{ll} S_1 \sqcap \sqcap_k\ R_k^c\ \forall R_k: & R_k^a \in GS(S_1), R_k^a \notin GS(S_1) \\ S_1 & \text{altrimenti} \end{array}$$

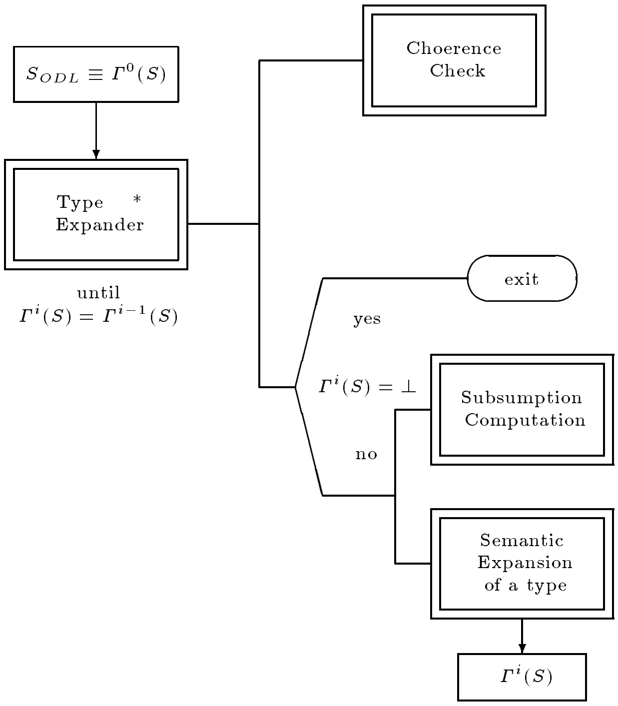


Fig. 4. Architettura funzionale di TYPE EXPANDER

itera finchè $\Gamma^{i+1}(S_1) = \Gamma^i(S_1)$, poni $\tilde{\Gamma}(S_1) = \Gamma^i(S_1)$.

– **Restituzione**: $\{S',\ \sigma(S') = \tilde{\Gamma}(S_1) \mid S_1 \in \mathbf{CT}^+(S)\}$

La complessità di caso peggiore dell'algoritmo è di tipo polinomiale. Dati T il numero dei tipi che descrivono lo schema canonico, R il numero delle regole e N (= T+R) il numero globale dei nomi, il numero massimo di elementi in $\mathbf{CT}^+(S)$ è denotata da N. L'operazione di calcolo di $\Gamma^i(S)$ comporta l'analisi dell'insieme $\mathbf{CT}^+(S)$ (cioè di N nomi) e il controllo sulla condizione di applicabilità delle regole (R^2 confronti). Quindi ogni singola iterazione dell'algoritmo ha una complessità pari a $O(\mathrm{N}{\cdot}\mathrm{R}^2)$. Ricordando poi che una regola può essere applicata al più una volta a ciascun nome dello schema, il numero massimo di iterazioni vale R·N; pertanto la complessità globale è pari a $O(\mathrm{R}^3{\cdot}\mathrm{N}^2)$.

5 Sessione di lavoro con ODB-QOPTIMIZER

Vediamo ora un esempio di ottimizzazione semantica utilizzando la query Q_1 vista in sezione 2. Il prototipo ODB-QOPTIMIZER acquisisce come input la query Q_1 scritta in linguaggio SQL e la converte nel formalismo ODL esprimendola come classe virtuale:

$$\sigma(Q) = \texttt{Storage} \sqcap (\triangle\texttt{.stock.}\forall.\triangle\texttt{.item} : \texttt{Material} \sqcap (\triangle\texttt{.risk}{:}\, 15 \div \infty))$$

Inoltre acquisisce la decrizione dello schema resa in forma canonica da una precedente esecuzione di ODL-Designer. Dopo la trasformazione, il controllo sulle operazioni passa al modulo Type Expander, che, utilizzando l'algoritmo 1, calcola l'espansione semantica dell'interrogazione.

Prima esecuzione dell'espansione semantica ($\Gamma^1(\mathbf{Q}_1)$)

– Type Expander attiva i moduli per il controllo di consistenza e il calcolo della forma canonica dell'interrogazione che risulta:

$$\begin{aligned}
&\nu(Q_1) = \overline{\texttt{Storage}} \sqcap \triangle T_{22}\\
&\quad \nu(T_{22}) = [\texttt{category}\colon \texttt{String}, \texttt{managed-by}\colon \texttt{Manager}, \texttt{stock}\colon T_{21}]\\
&\qquad \nu(T_{21}) = \forall\{T_{20}\}\\
&\qquad\quad \nu(T_{20}) = [\texttt{item}\colon N_3, \texttt{qty}\colon 10 \div 300]\\
&\qquad\qquad \nu(N_3) = \overline{\texttt{Material}} \sqcap \triangle T_{19}\\
&\qquad\qquad\quad \nu(T_{19}) = [\texttt{feature}\colon \forall\{\texttt{String}\}, \texttt{name}\colon \texttt{String},\\
&\qquad\qquad\qquad \texttt{risk}\colon 15 \div \infty]
\end{aligned}$$

L'operazione seguente è il calcolo delle relazioni di sussunzione tra la forma canonica della query Q e lo schema ODL preesistente. In particolare viene confermata la relazione $\texttt{Storage} \in \texttt{GS}(\mathrm{Q})$ e vengono calcolate ex-novo:
$\mathrm{R}_1^a \in \texttt{GS}(\mathrm{N}_3)$ poichè il dominio di risk in Q_1 è più ristretto di quello in R_1^a.
A questo punto viene costruito l'insieme $\mathbf{CT}^+(\mathrm{Q}_1)$:
$\mathbf{CT}^+(\mathrm{Q}_1) = \{\ \mathrm{Q}_1, \mathrm{T}_{22}, \texttt{Manager}, \mathrm{T}_{21}, \mathrm{T}_3, \mathrm{T}_{20}, \mathrm{N}_3, \mathrm{T}_{19}, \mathrm{T}_1\ \}$
ed, avendo a disposizione gli insiemi $\texttt{GS}(\mathrm{S}), \forall \mathrm{S} \in \mathbf{CT}^+(\mathrm{Q}_1)$

$\texttt{GS}(\mathrm{Q}_1) = \{\texttt{Storage}\}$	$\texttt{GS}(\mathrm{T}_{22}) = \{\mathrm{T}_7\}$	$\texttt{GS}(\texttt{Manager}) = \{\ \}$
$\texttt{GS}(\mathrm{T}_{21}) = \{\mathrm{T}_6, \mathrm{T}_{17}\}$	$\texttt{GS}(\mathrm{T}_3) = \{\ \}$	$\texttt{GS}(\mathrm{T}_{20}) = \{\ \mathrm{T}_5, \mathrm{T}_{16}\}$
$\texttt{GS}(\mathrm{N}_3) = \{\texttt{Material}, \texttt{R1a}\}$	$\texttt{GS}(\mathrm{T}_1) = \{\ \}$	$\texttt{GS}(\mathrm{T}_{19}) = \{\ \mathrm{T}_2, \mathrm{T}_8, \mathrm{T}_9\}$

posso calcolare la funzione $\Gamma^1(\mathrm{Q}_1)$:

- Analisi dei tipi che $\in \mathbf{CT}^+(\mathrm{Q}_1)$, da cui emerge che l'unica regola applicabile è sul tipo N_3.
  $\mathrm{R}_1^a \in \texttt{GS}(\mathrm{N}_3)$; $\mathrm{R}_1^c \notin \texttt{GS}(\mathrm{N}_3)$; viene quindi applicata R_1 a N_3 ottenendo: $\nu(N_3) = \nu(N_3) \sqcap \nu(R_1^c)$
- Type Expander genera la nuova classe virtuale $\mathrm{Q}_1^{'}$ ottenuta dall'applicazione di R_1^c al dominio dell'attributo `item`:

$$\sigma(\mathrm{Q}_1^{'}) = \Gamma^1(\mathrm{Q}_1) = \mathrm{Q}_1 \sqcap \triangle[\ \texttt{stock}\colon \{\texttt{item}\colon \mathrm{R}_1^c\}\ ]$$

Poiché sono state applicate regole occorre attivare un nuovo ciclo di esecuzioni di Type Expander per cercare di applicare ulteriormente regole alla query.

Seconda esecuzione dell'espansione semantica ($\Gamma^2(\mathbf{Q}_1)$)

– Type Expander attiva i moduli per il controllo di consistenza e il calcolo della forma canonica della nuova interrogazione ($\mathrm{Q}_1^{'}$) che risulta:

$$\begin{aligned}
&\nu(Q_1^{'}) = \overline{\texttt{Storage}} \sqcap \triangle T_{26}\\
&\quad \nu(T_{26}) = [\texttt{category}\colon \texttt{String}, \texttt{managed-by}\colon \texttt{Manager}, \texttt{stock}\colon T_{25}]
\end{aligned}$$

$$\nu(T_{25}) = \forall\{T_{24}\}$$
$$\nu(T_{24}) = [\texttt{item}\colon N_4, \texttt{qty}\colon 10 \div 300]$$
$$\nu(N_4) = \overline{\texttt{Material}} \sqcap \overline{\texttt{SMaterial}} \sqcap \triangle T_{23}$$
$$\nu(T_{23}) = [\texttt{feature}\colon \forall\{\texttt{String}\}, \texttt{name}\colon \texttt{String}, \texttt{risk}\colon 15 \div \infty]$$

Vengono calcolate le nuove relazioni di sussunzione tra la forma canonica della query Q e lo schema ODL preesistente.
A questo punto viene costruito l'insieme $\mathbf{CT}^{+}(Q_1^{'})$:
$\mathbf{CT}^{+}(Q_1^{'}) = \{\ Q_1^{'}, T_{26}, \texttt{Manager}, T_{25}, T_3, T_{24}, N_4, T_{23}, T_1\ \}$
ed, avendo a disposizione gli insiemi $\texttt{GS}(S), \forall S \in \mathbf{CT}^{+}(Q_1^{'})$

$\texttt{GS}(Q_1^{'}) = \{\texttt{Storage}, \texttt{R2a}, Q_1\}$ $\qquad$ $\texttt{GS}(\texttt{Manager}) = \{\ \}$
$\texttt{GS}(T_{26}) = \{T_7, T_{12}, T_{22}\}$ $\qquad$ $\texttt{GS}(T_{25}) = \{T_6, T_{11}, T_{17}, T_{21}\}$
$\texttt{GS}(T_3) = \{\ \}$ $\qquad$ $\texttt{GS}(T_{24}) = \{\ T_5, T_{10}, T_{16}, T_{20}\}$
$\texttt{GS}(N_4) = \{\texttt{Material}, \texttt{SMaterial}, \texttt{R1a}, \texttt{R1c}, N_1, N_3\}$
$\texttt{GS}(T_{23}) = \{\ T_2, T_8, T_9, T_{19}\}$ $\qquad$ $\texttt{GS}(T_1) = \{\ \}$

posso calcolare la funzione $\Gamma^2(Q_1)$:

- Analisi dei tipi che $\in \mathbf{CT}^{+}(Q_1^{'})$, da cui emerge che l'unica regola applicabile è :
  $R_2^a \in \texttt{GS}(Q_1^{'})$; $R_2^c \notin \texttt{GS}(Q_1^{'})$; viene quindi applicata R_1 a $Q_1^{'}$ ottenendo: $\nu(Q_1^{'}) = \nu(Q_1^{'}) \sqcap \nu(R_2^c)$
- Type Expander genera la nuova classe virtuale $Q_1^{''}$ ottenuta dall'applicazione di R_2^c :

$$\sigma(Q_1^{''}) = \Gamma^2(Q_1) = Q_1^{'} \sqcap R_2^c$$

Poiché sono state applicate regole occorre attivare un nuovo ciclo di esecuzioni di Type Expander per cercare di applicare ulteriormente regole alla query.

Terza esecuzione dell'espansione semantica ($\Gamma^3(\mathbf{Q_1})$)

– Type Expander attiva i moduli per il controllo di consistenza e il calcolo della forma canonica della nuova interrogazione ($Q_1^{''}$) che risulta:

$$\nu(Q_1^{''}) = \overline{\texttt{Storage}} \sqcap \overline{\texttt{Storage}} \sqcap \triangle T_{29}$$
$$\nu(T_{29}) = [\texttt{category}\colon \texttt{String}, \texttt{managed-by}\colon \texttt{Manager}, \texttt{stock}\colon T_{28}]$$
$$\nu(T_{28}) = \forall\{T_{27}\}$$
$$\nu(T_{27}) = [\texttt{item}\colon N_5, \texttt{qty}\colon 10 \div 300]$$
$$\nu(N_5) = \overline{\texttt{Material}} \sqcap \overline{\texttt{SMaterial}} \sqcap \triangle T_{19}$$

Vengono calcolate le nuove relazioni di sussunzione tra la forma canonica della query Q e lo schema ODL preesistente.
A questo punto viene costruito l'insieme $\mathbf{CT}^{+}(Q_1^{''})$:
$\mathbf{CT}^{+}(Q_1^{''}) = \{\ Q_1^{''}, T_{29}, \texttt{Manager}, T_{28}, T_3, T_{27}, N_5, T_{19}, T_1\ \}$
ed, avendo a disposizione gli insiemi $\texttt{GS}(S), \forall S \in \mathbf{CT}^{+}(Q_1^{'})$

$\texttt{GS}(Q_1^{''}) = \{\texttt{Storage}, \texttt{SStorage}, \texttt{R2a}, \texttt{R2c}, Q_1, Q_1^{'}\}$ $\qquad$ $\texttt{GS}(\texttt{Manager}) = \{\ \}$
$\texttt{GS}(T_{29}) = \{T_7, T_{12}, T_{22}, T_{26}\}$ $\qquad$ $\texttt{GS}(T_3) = \{\ \}$
$\texttt{GS}(T_{28}) = \{T_6, T_{11}, T_{17}, T_{21}, T_{25}\}$

$\texttt{GS}(\mathrm{T}_{27}) = \{\ \mathrm{T}_5, \mathrm{T}_{10}, \mathrm{T}_{16}, \mathrm{T}_{20}, \mathrm{T}_{24}\}$
$\texttt{GS}(\mathrm{N}_5) = \{\texttt{Material}, \texttt{SMaterial}, \texttt{R1a}, \texttt{R1c}, \mathrm{N}_1, \mathrm{N}_3, \mathrm{N}_4\}$
$\texttt{GS}(\mathrm{T}_{19}) = \{\ \mathrm{T}_2, \mathrm{T}_8, \mathrm{T}_9\}$ $\texttt{GS}(\mathrm{T}_1) = \{\ \}$
posso calcolare la funzione $\Gamma^3(\mathrm{Q}_1)$:

- Analisi dei tipi di $\mathbf{CT}^+(\mathrm{Q}_1^{'})$, da cui emerge che nessuna regola è applicabile.
- Poichè non possiamo applicare alcuna regola $\Gamma^3(\mathrm{Q}_1) = \Gamma^2(\mathrm{Q}_1)$, da cui segue che abbiamo ottenuto l'espansione semantica dell'interrogazione, che in forma SQL puo' essere espressa come:
  select * from `SStorage`
  where for all `stock` **:** `item` $=$ `SMaterial` **and** `^risk` ≥ 15 **;**

6 Confronti con altri lavori

Normalmente, nei sistemi per la rappresentazione della conoscenza basati sulle logiche descrittive [2], è prevista una componente che, utilizzando la sussunzione e le classi virtuali, effettua l'acquisizione e il controllo della gerarchia delle classi.

In vari lavori [17, 18, 19, 1], questa idea è stata estesa anche agli schemi per basi di dati. Inoltre, nei sistemi CANDIDE [18] e CLASSIC [19] le interrogazioni sono espresse nello stesso linguaggio utilizzato per descrivere la base di conoscenza. In questo modo, le tecniche di inferenza possono essere applicate anche alle interrogazioni. D'altra parte, l'idea dell'ottimizzazione dell'interrogazioni basata sul calcolo della sussunzione tra una interrogazione e una classe virtuale dello schema è stata importata anche nell'ambito OODB [12]; gli autori individuano un formalismo, per rappresentare parte dello schema e delle interrogazioni, per il quale il problema della sussunzione può essere deciso in tempo polinomiale. Però in questi sistemi non vengono normalmente trattate regole di integrità e quindi la sussunzione (e, di conseguenza, l'acquisizione dello schema delle classi e l'ottimizzazione semantica delle interrogazioni) è significativa solo in presenza di classi virtuali. Il nostro metodo estende l'ambito di questi lavori in quanto permette di ottenere, definendo uno schema con regole, nuove superclassi che sono sia classi virtuali che classi base.

Numerosi lavori relativi all'ottimizzazione semantica sono presenti in letteratura. In particolare, il metodo dell'espansione semantica, attraverso il quale vengono aggiunte all'interrogazione, in modo esaustivo, tutte le restrizioni possibili, è stato utilizzato anche in altri lavori [14, 20]. In [14] gli autori descrivono un metodo di ottimizzazione semantica, riferito al modello relazionale, basato sui grafi che sfrutta due tipi di vincoli di integrità: vincolo di sottoinsieme e vincolo di implicazione. In particolare, in questo lavoro viene chiamata espansione semantica quella fase della trasformazione che ha l'obiettivo di incorporare ogni restrizione o join che non era presente nell'interrogazione originale. Le principali fasi successive all'espansione semantica sono l'eliminazione di relazioni e l'eliminazione di join. In [20] è presentato un efficiente algoritmo di ottimizzazione semantica delle interrogazioni per un OODB che considera anche vincoli di integrità. L'approccio è simile all'espansione semantica in quanto tutte le trasformazioni sono applicate all'interrogazione; il compito di scegliere le trasformazioni utili è ritardato fino a quando tutte le trasformazioni sono state considerate. In questo modo le trasformazioni che sono state applicate non precludono trasformazioni future e l'ordine delle trasformazioni è irrilevante, pertanto gli autori affermano che l'algoritmo è di complessità polinomiale. L'algoritmo e' riferito ad un particolare genere di schema e di vincoli di integrità (in forma di clausole di Horn) espressi con un modello che non e' ad oggetti complessi. Del modello non viene data la semantica e non viene presentata nessuna verifica di correttezza delle trasformazioni semantiche considerate.

Nell'ambito OODB, alcuni lavori propongono l'uso della relazione di ereditarietà tra classi per effettuare l'ottimizzazione semantica delle interrogazioni. In [21] è presentata una metodologia per l'elaborazione delle interrogazioni orientate ad oggetti. Al fine di ottimizzare le interrogazioni vengono introdotte regole di trasformazione distinte in *algebriche* e *semantiche*. Le trasformazioni algebriche si limitano a riscrivere una interrogazione, creando espressioni equivalenti basate sulla sostituzione testuale e il "pattern matching". Le trasformazioni semantiche trasformano l'interrogazione sulla base della definizione delle classi e della gerarchia di ereditarietà.

In [22] viene investigata l'ottimizzazione di una classe di interrogazioni *congiuntive* per OODB. Per tali interrogazioni viene studiato e caratterizzato il problema del *contenimento* tra interrogazioni e della *minimizzazione* dello spazio di ricerca per le variabili coinvolte nell'interrogazione. Le informazioni utilizzate sono quelle "classiche" espresse in uno schema: il tipo degli attributi di una classe, la relazione di specializzazione e l'assunzione che un'oggetto sia istanziato in una ed una sola classe. In entrambi i lavori appena citati la classe delle interrogazioni e la categoria di problemi presi in esame sono più ampie di quelle considerate nel presente articolo. D'altra parte, oltre al fatto che il modello di riferimento di questi lavori non è ad oggetti complessi, dal punto di vista proprio dell'ottimizzazione semantica non sono stati considerati vincoli di integrità espressi come regole che risultano molto utili nella trasformazione di una interrogazione. Inoltre, l'espansione semantica e, più in generale, tutte le trasformazioni proposte con il nostro metodo sono *corrette*, cioè producono espressioni semanticamente equivalenti. Tale proprietà è formalmente garantita dalla precisa semantica sia del modello dei dati che della relazione di sussunzione. Il problema di una formale verifica della correttezza delle trasformazioni semantiche è stato affrontato solo in pochi lavori, tra i quali [23]. In questo lavoro ogni trasformazione è considerata singolarmente e viene dimostrata la sua correttezza.

Lavori più recenti di elevata generalità sia per il modello di dati che per il linguaggio di interrogazione sono stati proposti per basi di conoscenza orientate ad oggetti [24, 25]. In tali lavori viene utilizzato un linguaggio logico del primo ordine sia per la descrizione degli oggetti, in termini di classi, leggi deduttive e vincoli di integrità, sia per il linguaggio di interrogazione. Le opportunità di ottimizzazione derivano essenzialmente dalla natura deduttiva della base di conoscenza.

In [24] viene descritta una tecnica di trasformazione semantica che utilizza regole di riscrittura per trasformare una interrogazione; inoltre vengono presentati alcuni criteri per ordinare i fattori nella interrogazione trasformata.

In [25] i principi di astrazione di un modello orientato ad oggetti sono espressi come assiomi in una teoria del primo ordine. Tali assiomi vengono utilizzati quindi per l'ottimizzazione semantica; ad esempio, l'assioma che esprime il tipo di un attributo può essere utilizzato per eliminare, in una interrogazione, un predicato di appartenenza ad una classe. Le interrogazioni sono considerate classi virtuali, la cui semantica è definita tramite regole deduttive. In questo modo una interrogazione può essere classificata nella gerarchia delle classi ottenendo un'ulteriore ottimizzazione nel caso in cui l'interrogazione è una sottoclasse di una *vista materializzata*.

7 Conclusioni e lavoro futuro

In questo lavoro è stato presentato sia l'ambito teorico che l'implementazione di un sistema intelligente per l'acquisizione e la verifica di schemi e per l'ottimizzazione semantica di interrogazioni nelle OODB. È stato utilizzato ed esteso il formalismo ODL per esprimere nello stesso formalismo descrizioni di classi, vincoli d'integrità ed interrogazioni. In questo modo sia il controllo dello schema che l'ottimizzazione delle interrogazioni vengono essenzialmente effettuate con lo stesso metodo, basato sul calcolo dell'espansione semantica, che utilizza il calcolo della sussunzione introdotto per le Logiche Descrittive nell'ambito dell'Intelligenza Artificiale.

Il lavoro futuro riguarderà entrambe le componenti di ODB-Reasoner. In particolare, è in fase di realizzazione il componente del sistema che si occupa della memorizzazione persistente degli schemi e delle interrogazioni utilizzate da ODL-Designer e ODB-QOptimizer. Tale componente, realizzato in C++, utilizza il sistema EOS [26] per la memorizzazione persistente di oggetti sviluppato presso gli AT&T Bell Labs.

Per quanto riguarda l'ottimizzazione semantica verrà sviluppato sia l'ambito teorico che quello implementativo per effettuare eliminazione di fattori ridondanti dall'interrogazione.

Referimenti Bibliografici

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. W.A. Woods and J.G. Schmolze. The kl-one family. In F.W. Lehman, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Pubblished as a Special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2-9.
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. 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.
7. R.G.G. Cattell. *The Object Database Standard: ODMG-93, Release 1.1.* Morgan Kaufmann Publishers, Inc., 1994.
8. J. J. King. *Query optimization by semantic reasoning.* PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.
9. 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.
10. M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.
11. W. Kim. A model of queries for object-oriented database systems. In *Int. Conf. on Very Large Databases*, Amsterdam, Holland, August 1989.
12. M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.
13. L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.
14. 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.
15. M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimization. *ACM Transactions on Database Systems*, 17(4):563–600, December 1992.
16. D. Beneventano and S. Bergamaschi. Incoherence and subsumption for recursive views and queries in object-oriented data models. *Data & Knowledge Engineering*, 1996. To appear.
17. S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.
18. H.W. Beck, S.K. Gala, and S.B. Navathe. Classification as a query processing technique in the CANDIDE data model. In *5th Int. Conf. on Data Engineering*, pages 572–581, Los Angeles, CA, 1989.
19. A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.
20. H. H. Pang, H. Lu, and B.C. Ooi. An efficient semantic query optimization algorithm. In *Int. Conf. on Data Engineering*, pages 326–335, 1991.
21. D. D. Straube and T. Özsu. Queries and query processing in object-oriented database systems. *ACM Trans. on Information Systems*, 8(4):387–430, October 1990.
22. Edward P.F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Principles of Database Systems*, pages 202–11. ACM, 1992.
23. E. Bertino and D. Musto. Query optimization by using knowledge about data semantics. *Data and Knowledge Engineering*, 9(2):121–155, December 1992.
24. P. C. Sheu, R.L. Kashyap, and S. Yoo. Query optimization in object-oriented knowledge bases. *Data & Knowledge Engineering*, 3:285–302, 1989.

25. M. Jeusfeld and M. Staudt. Query optimization in deductive object bases. In Freytag, Maier, and Vossen, editors, *Query Processing for Advanced Database System*. Morgan Kaufmann Publishers, Inc., June 1993.
26. A. Biliris and A. Panagos. *EOS Users's Guide Realease 2.2*. At&T Bell Laboratoires, Murray Hill, NY, 1995.