ODB-QOptimizer: un Ottimizzatore Semantico di interrogazioni per OODB*

Jean Paul Ballerini° Domenico Beneventano◇
Sonia Bergamaschi◇ Maurizio Vincini◇

Convegno Nazionale Sistemi di Basi di Dati Evolute (SEBD95), Ravello, Italy. 1996

Sommario

L'ottimizzazione semantica delle interrogazioni utilizza la conoscenza specifica del problema, cioè i vincoli d'integrità, per trasformare un'interrogazione in una *equivalente*, ovvero con la stessa risposta, cui è possibile rispondere in modo più efficiente. Il metodo di ottimizzazione presentato nel lavoro si applica alle interrogazioni di tipo congiuntivo ed è basato su due ingredienti fondamentali. Il primo è *ODL* (*Object Description Logics*) proposto come formalismo comune per esprimere: descrizioni di classi, un insieme rilevante di *vincoli d'integrità* ed interrogazioni. Il secondo ingrediente è costituito dalle tecniche di inferenza introdotte per le *Logiche Descrittive* (*DL*) nell'ambito dell'Intelligenza Artificiale. Tali tecniche vengono utilizzate per valutare le implicazioni logiche espresse dalle regole d'integrità e, quindi, per determinare l'*espansione semantica* di una interrogazione. Infine, viene descritto il prototipo ODB-QOptimizer in grado di effettuare l'ottimizzazione semantica di interrogazioni nell'ambito di sistemi di basi di dati orientati agli oggetti.

1 Introduzione

La nozione di ottimizzazione semantica di query è stata introdotta per le basi di dati relazionali da King [7, 8] e da Hammer e Zdonik [5]. L'idea di base di queste

*Questa ricerca è stata parzialmente finanziata dal progetto M.U.R.S.T. 40% , "Metodi Formali e Strumenti per Basi di Dati Evolute".
° Centro Interdipartimentale di Ricerca in Filosofia del Diritto e Informatica Giuridica, Università di Bologna, Via Galliera 3, I-40121 Bologna, Italia.
◇ Dipartimento di Scienze dell'Ingegneria, Università di Modena, Via G. Campi 213/B, I-41100 Modena, Italia.
e_mail: { ballerin, benevent, sonia, vincini }@sparc20.dsi.unimo.it

proposte è che i vincoli di integrità, espressi per forzare la consistenza di una base di dati, possono essere utilizzati anche per ottimizzare le query fatte dall'utente.

Gli ingredienti principali del nostro approccio per l'ottimizzazione semantica delle query sono i seguenti:

- *la logica descrittiva* ODL *per descrivere schemi di basi di dati generalizzati*

ODL è stato proposto in [3] ed estende l'espressività di linguaggi di logica descrittiva al fine di rappresentare la semantica dei modelli di dati ad oggetti complessi (*CODMs*) recentemente proposti in ambito basi di dati deduttive [1] e basi di dati orientate agli oggetti [9, 2]. ODL introduce 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, e inoltre una distinzione fra *valore* ed *oggetti* con identità e, quindi, fra *tipi valore* e *tipi classe* (chiamati classi). Ciò equivale a dire che i concetti sono partizionati fra quelli che rappresentano insiemi di oggetti e quelli che rappresentano insiemi di valori. Sono, inoltre, direttamente esprimibili i costruttori di tipo, resi spesso disponibili dai CODMs, quali *tuple* ed *insiemi*.

L'estensione di ODL introdotta in questo lavoro ha l'obiettivo di permettere la formulazione dichiarativa di un insieme rilevante di vincoli d'integrità sulla base di dati. Gli schemi di basi di dati reali sono, di fatto, forniti in termini di classi base mentre l'ulteriore conoscenza è espressa attraverso vincoli d'integrità che dovrebbero garantire la consistenza dei dati. In particolare, abbiamo esteso ODL con: *tipi path quantificati* e *regole d'integrità*. La prima estensione è stata introdotta per gestire in modo potente ed agile le strutture nidificate. 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. In particolare, 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 percorso. Uno *schema di basi di dati generalizzato* può, quindi, essere definito come un insieme di dichiarazioni di inclusione fra tipi ODL [4]: cioè permette la formulazione dichiarativa di un altro insieme rilevante di vincoli d'integrità che esprime regole *if then* i cui antecedenti e conseguenti sono tipi 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 particolare, una classe può essere rappresentata tramite una dichiarazione di inclusione dove l'antecedente è il nome della classe e il conseguente la sua descrizione.

- *Query e tecniche di inferenza dei DL*

Un insieme rilevante di query per OODB [6] può essere espresso come un tipo *virtuale*, vista la ricchezza del formalismo ODL. Le tecniche di inferenze sviluppate per i DL in ambito Intelligenza Artificiale sono: il *calcolo della sussuzione*, cioè il calcolo delle relazioni di generalizzazione implicite nella descrizione dei tipi, la rilevazione delle *inconsistenze*, cioè la rilevazione di tipi la cui estensione è vuota per ogni dominio, e la *generazione della forma canonica*, cioè una forma nella quale la descrizione di un tipo viene espressa in modo da rendere confrontabili due tipi sulla base della loro semantica, indipendentemente da eventuali differenze sintattiche. La trasformazione di uno schema in uno schema canonico preserva la semantica dello schema originale, ovvero le relazioni semantiche valide nello schema originale e permette di applicare più facilmente le altre tecniche DL: calcolo della sussunzione e delle inconsistenze. Le tecniche DL possono essere utilizzate per determinare l'*espansione semantica EXP(Q)* di una query Q. Si tratta di una query trasformata che incorpora ogni possibile restrizione che non è presente nella query originale ma che è *logicamente implicata* dalla query e dallo schema globale (classi + tipi + regole d'integrità). Seguendo l'approccio di [10] per l'ottimizzazione semantica, eseguiamo l'espansione semantica dei tipi inclusi ad ogni livello di annidamento nella descrizione della query. L'espansione dei tipi è basata sull'iterazione di questa semplice trasformazione: se un tipo implica l'antecedente di una regola d'integrità allora il conseguente di quella regola può essere intersecato alla descrizione del tipo stesso. Le implicazioni logiche fra i tipi (i tipi da espandere e l'antecedente di una regola) sono determinate attraverso il *calcolo della sussunzione*, che per ODL è stato presentato in [3]. Seguendo questo approccio, l'espansione semantica di una query coincide con la *query più specializzata* fra tutte quelle semanticamente equivalenti. Una query, cioè può essere spostata verso il basso nella gerarchia delle classi e quindi l'insieme degli oggetti interessati alla sua esecuzione può essere ridotto. Durante la trasformazione, determiniamo e sostituiamo nella query, relativamente alla sua gerarchia di aggregazione, le *classi più specializzate*.

1.1 Esempio: il dominio Magazzino

Per illustrare il nostro metodo, consideriamo il seguente esempio relativo alla gestione di un magazzino (vedi tabella 1). I materiali (`Material`) sono descritti da un nome, un rischio e da una caratteristica. 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. I manager hanno un nome, un salario superiore ai 40K dollari e un livello compreso tra 1 e 15. I `TManager` sono quei dirigenti che hanno un livello compreso tra 8 e 12. Infine abbiamo i magazzini "speciali" (`SStorage`) e i materiali "speciali" (`SMaterial`). 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 come regole *if then rule* in tabella 1) per garantire la consistenza della base di dati.

class	Material	=	[name: String, risk: Int, feature : {String}]
class	SMaterial	=	**isa** Material
class	Storage	=	[managed-by : Manager, category : String, stock : {item : Material, qty : $10 \div 300$]}]
class	SStorage	=	**isa** Storage
class	Manager	=	[name: String, salary: $40K \div \infty$, level : $1 \div 15$]
class	TManager	=	**isa** Manager **and** [level : $8 \div 12$]

Vincoli di integrità:

R_1	**if** Manager **and** (level : $5 \div 10$)	**then**	(salary : $40K \div 60K$)
R_2	**if** Material **and** (risk ≥ 10)	**then**	SMaterial
R_3	**if** Storage **and** (category = "$B4''$)	**then**	(managed-by : TManager)
R_4	**if** Storage **and** (stock.$\forall$.item = SMaterial)	**then**	SStorage
R_5	**if** Storage **and** (stock.$\exists$.qty : $10 \div 50$)	**then**	(category = "$B4''$)

Tabella 1: Lo schema del Magazzino in una sintassi OODB–like

Ad esempio la regola R_4 dice che tutti i magazzini che contengono materiali tutti appartenenti alla classe `SMaterial` appartengono alla classe degli `SStorage`.

A partire dallo schema con regole vogliamo ora mostrare come opera il metodo di ottimizzazione semantica delle query; il primo esempio riguarda la rilevazione di una query inconsistente, la quale permette di fornire una risposta senza accedere al database. Q_1 : "Seleziona i dirigenti che hanno un livello superiore a 20".
Una query è un *tipo virtuale*, poiché la sua descrizione esprime condizioni necessarie e sufficienti. Nella sintassi OODB-like, la Q_1 può essere espressa come una classe vituale nel seguente modo:

virtual class Q_1 = **isa** Manager **and** [level : $20 \div \infty$]

Il metodo da noi proposto permette di rilevare che l'intervallo per il livello definito nella query (`level` : $20 \div \infty$) è disgiunto da quello definito nella classe `Manager` (`level` : $1 \div 15$) e quindi che Q_1 è inconsistente poichè nessun dominio di oggetti può popolarla.

Il secondo esempio illustra l'effettiva ottimizzazione calcolata tramite la tecnica dell'espansione semantica che riclassifica la query introducendo la conoscenza contenuta nei vincoli di integrità. Q_2 : "Seleziona i magazzini che hanno categoria "B4", sono diretti da un manager con livello inferiore a 10 e contengono materiali che hanno tutti rischio maggiore o uguale a 15".

$$\begin{array}{llll} \underline{\textbf{virtual class}} & \mathrm{Q}_2 & = & \underline{\textbf{isa}}\ \texttt{Storage}\ \ \underline{\textbf{and}}\ [\ \texttt{category=}\ \textit{"B4"}, \\ & & & \quad \texttt{managed-by}:\ \underline{\textbf{isa}}\ \texttt{Manager}\ \underline{\textbf{and}}\ (\texttt{level}\ \leq 10), \\ & & & \quad \texttt{stock.}\forall\texttt{.item}:\ \underline{\textbf{isa}}\ \texttt{Material}\ \underline{\textbf{and}}\ (\texttt{risk} \geq 15)\ ] \end{array}$$

Poiché la query è consistente cerchiamo di applicare i vincoli di integrità: a Q_2 possiamo applicare R_2 ed R_3 dando origine alla nuova query:

$$\begin{array}{llll} \underline{\textbf{virtual class}} & \mathrm{Q}_2^{'} & = & \underline{\textbf{isa}}\ \texttt{Storage}\ \ \underline{\textbf{and}}\ [\ \texttt{category=}\ \textit{"B4"}, \\ & & & \quad \texttt{managed-by}:\ \underline{\textbf{isa}}\ \texttt{TManager}\ \underline{\textbf{and}}\ (\texttt{level}\ 8 \div 10), \\ & & & \quad \texttt{stock.}\forall\texttt{.item}:\ \underline{\textbf{isa}}\ \texttt{SMaterial}\ \underline{\textbf{and}}\ (\texttt{risk} \geq 15)\ ] \end{array}$$

Questa applicazione ci permette di sostituire le due classi (`Manager` e `Material`) presenti nella gerarchia di aggregazione della query con due sottoclassi (`TManager` e `SMaterial`) e quindi di ottenere già una ottimizzazione di $\mathrm{Q}_2^{'}$. Iterando il procedimento di controllo sull'applicabilità dei vincoli di integrità alla $\mathrm{Q}_2^{'}$ possiamo applicare R_1 ed R_4 dando origine alla:

$$\begin{array}{llll} \underline{\textbf{virtual class}} & \mathrm{Q}_2^{''} & = & \underline{\textbf{isa}}\ \texttt{SStorage}\ \ \underline{\textbf{and}}\ [\ \texttt{category=}\ \textit{"B4"}, \\ & & & \quad \texttt{managed-by}:\ \underline{\textbf{isa}}\ \texttt{TManager}\ \underline{\textbf{and}}\ (\texttt{level}\ 8 \div 10) \\ & & & \qquad\qquad\qquad \underline{\textbf{and}}\ (\texttt{salary}\ : 40K \div 60K), \\ & & & \quad \texttt{stock.}\forall\texttt{.item}:\ \underline{\textbf{isa}}\ \texttt{SMaterial}\ \underline{\textbf{and}}\ (\texttt{risk} \geq 15)\ ] \end{array}$$

A $\mathrm{Q}_2^{''}$ non è possibile applicare ulteriori vincoli di integrità; ciò significa che abbiamo trovato l'espansione semantica della query che rappresenta la classe più specializzata tra quelle equivalenti alla query di partenza. Si può notare come la query risulti ottimizzata in quanto essa è ora una specializzazione di `Storage` (sostituita dalla classe `SStorage`) e sono state specializzate classi appartenenti alla sua gerarchia di aggregazione.

2 ODL esteso con i vincoli di integrità

In questa sezione il formalismo ODL introdotto in [3] viene esteso per rappresentare i vincoli di integrità.

2.1 Schema e Istanza del Database

Sia **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 **B** l'insieme di designatori di tipi atomici, con $\mathbf{B} = \{$`Int`, `String`, `Bool`, `Real`, i_1–j_1, i_2–j_2, ..., d_1, d_2, ...$\}$, 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 $\mathbf{D}$, $\mathcal{V}(\mathcal{O})$ di tutti i *valori su* $\mathcal{O}$ costruito secondo le seguenti regole (assumendo $p \geq 0$ e $a_1, \ldots, a_p$ attributi distinti in $\mathbf{A}$):

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

Gli identificatori di oggetti sono valori assegnati da una *funzione totale* δ da $\mathcal{O}$ a $\mathcal{V}(\mathcal{O})$. 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. $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$[1] indica l'insieme di tutte le *descrizioni di tipo finite* (S, S', …), dette brevemente *tipi*, su di un dato $\mathbf{A}, \mathbf{B}, \mathbf{N}$, ottenuto in accordo con la seguente regole sintattica:

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

$\top$ denota il *tipo universale*; [ ] 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. Il tipo $\sigma(N)$ esprime sia le proprietà locali del nome di tipo N sia le relazioni di *isa* tra N e altri nomi di tipo. Infatti, diremo che un nome di tipo N *eredita* da un altro nome di tipo N', denotato con $N \prec\!\!\mid N'$, se $\sigma(N) = N' \sqcap S$. Si richiede che la relazione di ereditarietà sia priva di cicli, i.e., la chiusura transitiva di $\prec\!\!\mid$, denotata $\prec$, sia un ordine parziale stretto; in tal caso si parla di schema *ben formato*.

Dato un dato sistema di tipi $\mathbf{S}$, una *regole di integrità* R è espressa nella forma $R = S^a \rightarrow S^c$, 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 tramite lo schema con regole mostrato in tabella 2.

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

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

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

$$\mathbf{R} \begin{cases} R_1: & \texttt{Manager} \sqcap (\triangle.\texttt{level}: 5 \div 10) \rightarrow (\triangle.\texttt{salary}: 40K \div 60K) \\ R_2: & \texttt{Material} \sqcap (\triangle.\texttt{risk}: 10 \div \infty) \rightarrow \texttt{SMaterial} \\ R_3: & \texttt{Storage} \sqcap (\triangle.\texttt{category}: \text{``}B4\text{''}) \rightarrow \triangle[\texttt{managed-by}: \texttt{TManager}] \\ R_4: & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\forall.\texttt{item}: \texttt{SMaterial}) \rightarrow \texttt{SStorage} \\ R_5: & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{qty}: 10 \div 50) \rightarrow (\triangle.\texttt{category}: \text{``}A2\text{''}) \end{cases}$$

Tabella 2: Schema con Regole di Integrità in linguaggio ODL

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

$$\begin{aligned} \mathcal{I}[[a_1: S_1, \ldots, a_p: S_p]] &= \Big\{[a_1: v_1, \ldots, a_q: v_q] \Big| p \leq q, v_i \in \mathcal{I}[S_i], 1 \leq i \leq p, \\ &\qquad v_j \in \mathcal{V}(\mathcal{O}), p+1 \leq j \leq q\Big\} \\ \mathcal{I}[\forall\{S\}] &= \Big\{\{v_1, \ldots, v_p\} \Big| v_i \in \mathcal{I}[S], 1 \leq i \leq p\Big\} \\ \mathcal{I}[\exists\{S\}] &= \Big\{\{v_1, \ldots, v_p\} \Big| \exists i, 1 \leq i \leq p, v_i \in \mathcal{I}[S]\Big\} \\ \mathcal{I}[\triangle S] &= \Big\{o \in \mathcal{O} \Big| \delta(o) \in \mathcal{I}[S]\Big\} \\ \mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S'] \end{aligned}$$

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

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

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

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

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

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

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 [3]; 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 (tipi aciclici).

2.2 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:*

$$S \sqsubseteq_{\mathrm{R}} S' \quad sse \quad \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ per tutte le istanze legali } \mathcal{I} \text{ di } (\sigma, \mathbf{R}).$$

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* [2, 9]. Questa relazione può essere calcolata attraverso una comparazione sintattica sui tipi; per il nostro modello tale l'algoritmo è stato presentato in [3].

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* $EXP(S) \sqsubseteq S'$*.*

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 [3].

Dato un tipo S, una regola R e un cammino p diciamo che R è *applicabile* a S con cammino p se $S \sqsubseteq (p : S^a)$. Ad esempio, se consideriamo $S =$ `Manager` $\sqcap$ $(\triangle.$`level`$: 5 \div 8)$ la regola R_1 di tabella 2 è applicabile a S con cammino ϵ. La stessa regola è applicabile con il cammino $\triangle$.`managed-by` al tipo `Storage` $\sqcap \triangle[$`managed-by`: `Manager` $\sqcap$ $(\triangle.$`level`$: 8 \div 10)]$.

È facile verificare che, per ogni $S \in \mathbf{S}$ e per ogni $R \in \mathbf{R}$, se R è applicabile a S con cammino p, allora $S \sqcap (p: S^c) \simeq_{\mathrm{R}} S$. Questa trasformazione del tipo S nel tipo $S \sqcap (p: S^c)$ è alla 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 $(\sigma, \mathbf{R})$, definiamo la funzione totale $\Gamma : \mathbf{S} \longrightarrow \mathbf{S}$, come segue:

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

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 ($S \not\sqsubseteq (p: S^c)$).

Si può dimostrare che, per ogni $S \in \mathbf{S}$, EXP(S) è effettivamente calcolabile tramite $\tilde{\Gamma}(S)$. Nella prossima sezione definiremo quindi un algoritmo efficiente per il calcolo della funzione $\tilde{\Gamma}$. Tale algoritmo si basa sulle tecniche DL applicate al formalismo ODL: trasformazione in forma canonica, controllo di consistenza e calcolo della relazione di sussunzione, brevemente presentate nel seguito.

2.3 Algoritmi

Il principale obiettivo della forma canonica è quello di semplificare sia il controllo di consistenza che il calcolo di sussunzione.

$\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$ denota l'insieme delle *descrizioni canoniche di tipo* ($\check{S}$, $\check{S}'$, ...) su $\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}}$, con $\tilde{\mathbf{N}} = \tilde{\mathbf{C}} \cup \tilde{\mathbf{V}} \cup \tilde{\mathbf{T}}$, che sono costruite rispettando la seguente regola sintattica[3] (con $k \geq 0$, $i \geq 0$ e $\tilde{S} \in \{\top\} \cup \mathbf{B} \cup \tilde{\mathbf{V}} \cup \tilde{\mathbf{T}}$ e $\tilde{C}_i \in \tilde{\mathbf{C}}$.)

$$\breve{S} \quad \to \quad \tilde{S} \mid \forall\{\tilde{S}\} \mid \exists\{\tilde{S}\} \mid [a_1 : \tilde{S}_1, \ldots, a_k : \tilde{S}_k] \mid \textstyle\bigsqcap_i \tilde{C}_i \sqcap \triangle\tilde{S}.$$

Uno *schema canonico* ν è uno schema su $\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$.

Definizione 4 (Schema Canonico Equivalente) *Dato uno schema σ su $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ si dice che uno schema canonico ν su $\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$ con $\mathbf{N} \subseteq \tilde{\mathbf{N}}$ è* equivalente *a σ se solo se per ogni istanza legale $\mathcal{I}$ di σ esiste un'istanza legale $\mathcal{I}'$ di ν e* vice versa *tale che $\mathcal{I}[N] = \mathcal{I}'[N], \forall N \in \mathbf{N}$.*

Da questa definizione segue che le relazioni semantiche in due schemi equivalenti sono le stesse: di conseguenza, il calcolo della sussunzione e il controllo di coerenza in uno schema arbitrario possono essere effettuati considerando lo schema canonico equivalente.

Ogni schema può essere effettivamente trasformato in uno schema canonico equivalente. I punti principali sono i seguenti: ogni classe base C è trasformata in una classe virtuale (espressa come la congiunzione della descrizione di C, $\sigma(C)$, e di un nuovo nome di classe base $\tilde{C}$, detto *atomo fittizio*, appartenente a $\tilde{\mathbf{C}}$); la descrizione di ogni nome di tipo N è trasformata sostituendo i nomi dai quali N eredita con le rispettive descrizioni, espandendo i nomi di tipi valore e rinominando i nuovi tipi che si ottengono risolvendo le intersezioni. Quindi, all'inizio si ha $\tilde{\mathbf{V}} = \mathbf{C} \cup \mathbf{V}$ e $\tilde{\mathbf{T}} = \mathbf{T}$ e questi insiemi crescono man mano che si effettuano le trasformazioni. Nella prossima sezione sono mostrati esempi di queste trasformazioni.

Per calcolare la sussunzione, si introduce una relazione $\leq$ tra i nomi di tipi ($\leq \subseteq \tilde{\mathbf{N}} \times \tilde{\mathbf{N}}$) e si definisce una relazione di sussunzione basata su $\leq$, denotata con $\breve{S} \sqsubseteq_{\leq} \breve{S}'$, come segue:

$$\begin{array}{rcl}
B \sqsubseteq_{\leq} B' & \text{sse} & B \sqcap B' = B \\
\forall\{N\} \sqsubseteq_{\leq} \forall\{N'\} & \text{sse} & N \leq N' \\
\bigsqcap_i \exists\{N_i\} \sqcap \forall\{N\} \sqsubseteq_{\leq} \forall\{N'\} & \text{sse} & N \leq N' \\
\bigsqcap_i \exists\{N_i\} \sqcap \forall\{N\} \sqsubseteq_{\leq} \bigsqcap_j \exists\{N'_j\} \sqcap \forall\{N'\} & \text{sse} & N \leq N' \wedge \forall j \exists i \colon N_i \leq N'_j \\
[\ldots, a_i : N_i, \ldots] \sqsubseteq_{\leq} [\ldots, a'_j : N'_j, \ldots] & \text{sse} & \forall j \exists i \colon (a_i = a'_j \wedge N_i \leq N'_j) \\
\bigsqcap_i \tilde{C}_i \sqcap \triangle N \sqsubseteq_{\leq} \bigsqcap_j \tilde{C}'_j \sqcap \triangle N' & \text{sse} & N \leq N' \wedge \forall j \exists i \colon \tilde{C}_i = \tilde{C}'_j.
\end{array}$$

Algoritmo 1 (Sussunzione) *Sia $\leq^0 = \tilde{\mathbf{N}} \times \tilde{\mathbf{N}}$ e definisci $\leq^{i+1}$ nel seguente modo:*

$$\leq^{i+1} = \left\{ (N, N') \in \tilde{\mathbf{N}} \times \tilde{\mathbf{N}} \,\middle|\, \nu(N) \sqsubseteq_{\leq^i} \nu(N') \vee N = \bot \vee N' = \top \right\}.$$

Calcola $\leq^0, \ldots, \leq^i, \ldots$ finchè $\leq^i = \leq^{i+1}$ e poni $\lesssim_\nu = \leq^i$.

[3] Si assume che per $i = 0$ la regola $\bigsqcap_i \tilde{C}_i \sqcap \triangle\tilde{S}$ produca il tipo $\triangle\tilde{S}$. Inoltre, per semplicità, i nomi di tipo non vengono segnati, cioè gli elementi di $\tilde{\mathbf{N}}$ vengono indicati ancora con N.

Poichè $\tilde{\mathbf{N}}$ è finito, esiste sempre un numero naturale n tale che $\leq^n = \leq^{n+1}$; il massimo valore di n è $|\tilde{\mathbf{N}}|^2$, quindi l'algoritmo di sussunzione è polinomiale in ν. L'algoritmo è corretto e completo: per ogni $N, N' \in \tilde{\mathbf{N}}$: $N \sqsubseteq N'$ sse $N \lesssim_\nu N'$.

A questo punto possiamo introdurre l'algoritmo per il calcolo dell'espansione semantica, per la descrizione del quale servono le seguenti definizioni. Dati N e $N' \in \mathbf{N}$, diciamo che N *dipende direttamente* da N' sse N' è contenuto nella descrizione canonica di $N, \nu(N)$. La chiusura transitiva, denotata con $\mathbf{CT}^+(N)$ è definita come segue:

Definizione 5 (Chiusura di un tipo) *Dato lo schema canonico* $\tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}})$*:*
$\mathbf{CT}^0(N) = \{N\} \cup \{N_1 \in \tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}}) \setminus (\mathbf{B} \cup \tilde{\mathbf{C}}) \mid N \textit{ dipende direttamente da } N_1\}$
$\mathbf{CT}^{i+1}(N) = \mathbf{CT}^i(N) \cup \{N_1 \in \tilde{\mathbf{S}}(\mathbf{A}, \mathbf{B}, \tilde{\mathbf{N}}) \setminus (\mathbf{B} \cup \tilde{\mathbf{C}}) \mid N_2 \in \mathbf{CT}^i(N) \wedge N_2$ dipende direttamente da $N_1\}$
$\mathbf{CT}^+(N) = \mathbf{CT}^{\bar{i}}(N)$, *dove* $\bar{i}$ *è il piú piccolo intero tale che* $\mathbf{CT}^{\bar{i}}(N) = \mathbf{CT}^{\bar{i}+1}(N)$.

Algoritmo 2 (Espansione Semantica)

- **Acquisizione del tipo** N**:** $((\nu(N_1),\ GS(N_1)^4),\ \forall\ N_1 \in \mathbf{CT}^+(N))$
- **Iterazione***:* $\forall\ N_1 \in \mathbf{CT}^+(N)$, *calcolo dell'espansione semantica di* N_1, $\tilde{\Gamma}(N_1)$*:*

$$\Gamma^i(N_1) \quad = \quad \begin{cases} N_1 \sqcap \bigsqcap_k\ R_k^c & \forall R_k :\ R_k^a \in GS(N_1), R_k^c \notin GS(N_1) \\ N_1 & \textit{altrimenti} \end{cases}$$

itera finchè $\Gamma^{i+1}(N_1) = \Gamma^i(N_1)$, *poni* $\tilde{\Gamma}(N_1) = \Gamma^i(N_1)$.

- **Restituzione***:* $(N',\ \sigma(N') = \tilde{\Gamma}(N))$

Complessità: il calcolo della funzione $\Gamma^i(N_1)$ ha complessità pari a $|\tilde{\mathbf{N}}|$ + $|\tilde{\mathbf{N}}_R^2|$ (con $|\tilde{\mathbf{N}}_R|$ numero di regole dello schema), poichè occorre ispezionare dapprima GS(N_1) che contiene al più $|\tilde{\mathbf{N}}|$ elementi e poi controllare per ciascun antecedente (al più $|\tilde{\mathbf{N}}_R|$) la condizione di applicabilità (al più $|\tilde{\mathbf{N}}_R|$ confronti). Il numero di iterazioni compiute su $\Gamma^i(N_1)$ sono pari, nel caso peggiore, al numero di regole $|\tilde{\mathbf{N}}_R|$, poichè una regola può essere applicata una sola volta sullo stesso *path*; mentre il ciclo esterno sugli elementi di $\mathbf{CT}^+(N)$ viene ripetuto al più $|\tilde{\mathbf{N}}|$ volte. Ne consegue una complessità di caso peggiore di tipo polinomiale pari a $|\tilde{\mathbf{N}}^2|\ |\tilde{\mathbf{N}}_R|$ + $|\tilde{\mathbf{N}}|\ |\tilde{\mathbf{N}}_R^3|$.

[4] GS denota Generalization Set, cioè l'insieme dei tipi generalizzazione di un dato tipo, calcolato dall'algoritmo di sussunzione.

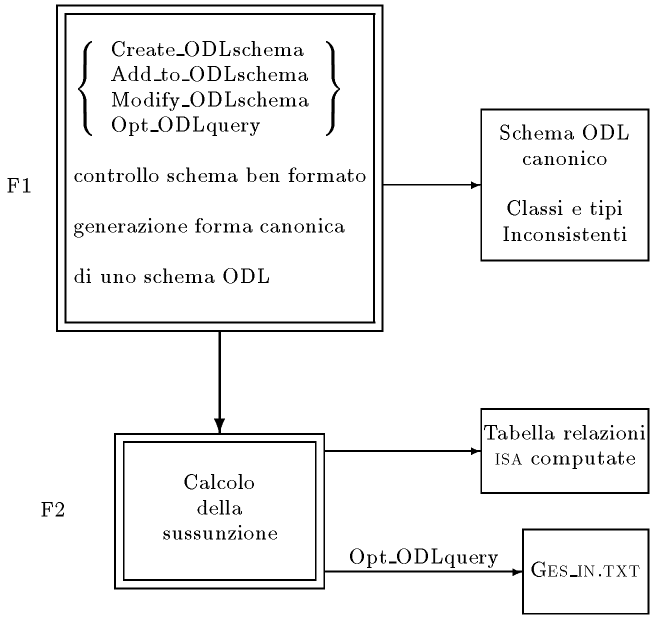

Figura 1: Architettura funzionale di ODL-DESIGNER

3 ODB-QOPTIMIZER: il tool

ODB-QOPTIMIZER è un software per l'ottimizzazione semantica delle query che, partendo da una generica query espressa in linguaggio ODL, ne calcola l'espansione semantica. ODB-QOPTIMIZER si compone di due moduli software: il prototipo ODL-DESIGNER ed il prototipo GES entrambi sviluppati su piattaforma hardware SUN SPARCstation 20, sistema operativo SOLARIS 2.3. ODL-DESIGNER realizza la trasformazione di uno schema in forma *canonica*, controlla la consistenza dei tipi e delle classi e calcola le relazioni di sussunzione tra tipi e classi. Una prima versione di ODL-DESIGNER è stata sviluppata in linguaggio di programmazione Prolog in ambiente SICStus prolog versione 2.1 nell'ambito del progetto finalizzato LOGIDATA$^+$. ODL-DESIGNER è poi stato esteso come verrà descritto in seguito per effettuare l'ottimizzazione semantica di query. GES è un componente software, sviluppato presso il Dipartimento di Scienze dell'Ingegneria (DSI) dell'Università di Modena, che realizza l'espansione semantica di un tipo. L'ambiente di sviluppo è il linguaggio di programmazione C, grazie alla portabilità del codice. Poiché GES e ODL-DESIGNER lavorano in ambienti differenti, abbiamo definito due file di interfaccia (GES_IN.TXT e GES_OUT.PL) per lo scambio di dati. Una nuova versione di ODL-DESIGNER in C è in corso di sviluppo presso il DSI.

In figura 1 viene descritta l'architettura di ODL-DESIGNER. Il programma è diviso in due blocchi funzionali principali:
il primo, denotato con F1, permette la creazione e la trasformazione di uno schema

ODL attraverso l'interfaccia utente, controlla la correttezza dello schema da un punto di vista sintattico e semantico, controlla l'aciclicità delle relazioni ISA esplicite e dei tipi valore e, nel caso in cui lo schema risulti corretto, genera la forma canonica dello schema ODL. F1 è stato esteso rispetto alla prima prototipazione per supportare schemi con regole *if then*. Il secondo, denotato con F2, partendo da uno schema ODL canonico, calcola le relazioni di *sussunzione*, cioè tutte le relazioni ISA, comprese quelle implicite. F2 è stato esteso rispetto alla prima prototipazione, in quanto vi è stata aggiunta la funzionalità Opt_ODLquery per ottimizzare le query. Le modalità di esecuzione di F1 sono quattro:

Create_ODLschema Lo schema inserito dall'utente, o attraverso l'interfaccia o indicando il file che lo contiene, viene analizzato da un punto di vista sintattico e semantico. Se supera positivamente questo controllo viene verificata l'assenza di cicli nel grafo delle relazioni ISA generato durante l'analisi dello schema inserito e l'assenza di tipi valore ciclici. Se dopo tale controllo lo schema risulta essere ben formato si procede alla generazione dello schema canonico sul quale vengono rilevate le eventuali classi o tipi inconsistenti. In questa fase è prevista una interazione con l'utente per ogni eventuale errore rilevato.

Add_to_ODLschema La procedura aggiunge una o più descrizioni di tipi ad uno schema preesistente (quindi anche una query che è una classe virtuale). Coincide con Create_ODLschema fino all'analisi sintattica e semantica. A questo punto, infatti, per verificare l'assenza di cicli ISA e di tipi valore ciclici, devono essere considerati sia i grafi di gerarchia generati dall'esecuzione corrente sia quelli generati durante la creazione dello schema cui si stanno aggiungendo classi e tipi. Successivamente, se lo schema globale risulta essere ben formato, il programma procede alla generazione della forma canonica per quanto riguarda le classi e i tipi di nuovo inserimento, prendendo in input anche lo schema canonico preesistente e determina così le classi e i tipi inconsistenti nonché lo schema canonico globale.
Questo tipo di procedimento è consentito dal fatto che si presume una crescita strettamente monotona della conoscenza che implica, quindi, il mantenimento della consistenza dello schema preesistente.

Modify_ODLschema La procedura parte da uno schema preesistente ma, in questo caso, l'input prevede la sostituzione della descrizione di uno o più classi o tipi esistenti con una nuova descrizione. Di conseguenza, al termine dell'input occorre procedere con l'analisi sintattica e semantica dello schema globale, cioè comprendente classi e tipi non modificati e classi e tipi nuovi o modificati, per poi procedere, come nel caso di creazione, con il controllo sull'aciclicità dei tipi e delle relazioni ISA, dello schema globale, e così via. A differenza della modalità Add_to_ODLschema, qui non si fa la presunzione di crescita monotona della conoscenza e ciò implica la necessità di ricalcolare la consistenza di tutto lo schema e non solamente della parte inserita con l'ultima esecuzione del programma.

Opt_ODLquery Viene elaborata una query introdotta come se si fosse in modalità Add_to_ODLschema. Alla funzionalità F2 che calcola la relazione di sussunzione di un tipo rispetto allo schema è stata aggiunta una funzianalità per l'ottimizzzazione della query: viene generato il file GES_IN.TXT che contiene la descrizione della query e dei tipi che la compongono ad ogni livello di innestamento, l'insieme delle generalizzazioni di tali tipi comprensivi degli antecedenti delle regole ed i tipi che definiscono i conseguenti di ciascuna regola il cui antecedente è generalizzazione di un qualche tipo a qualsiasi livello di innestamento.

Figura 2: Architettura funzionale di GES

Il modulo GES è descritto in figura 2, dalla quale vediamo che l'input è il file GES_IN.TXT generato da ODL-DESIGNER. GES realizza l'espansione semantica di una query applicando l'algoritmo presentato nella sezione precedente ad ogni tipo $\mathrm{N}_1 \in \mathbf{CT}^+(\mathrm{Q})$. L'output di GES è il file GES_OUT.PL contenente la descrizione, nel formalismo ODL, dell'espansione i-esima della query Q_i.

In figura 3 è illustrata l'architettura complessiva di ODB-QOPTIMIZER. Inserita una query Q per mezzo dell'interfaccia utente e attivato ODB-QOPTIMIZER, viene attivato una prima volta ODL-DESIGNER, in modalità Opt_ODLquery, che fornisce in uscita il rilevamento dell'inconsistenza della query Q o, nel caso sia consistente, il file GES_IN.TXT. Se la query Q è consistente, ODB-QOPTIMIZER attiva il componente GES il quale acquisisce il file GES_IN.TXT come proprio file di input. GES espande la query $(\Gamma^i(\mathrm{Q}))$ intersecando tutti i conseguenti delle regole applicabili ed inviando un messaggio a ODB-QOPTIMIZER riguardo al risultato della sua esecuzione. Si distinguono due casi principali: viene applicata almeno una regola (quindi presumibilmente l'espansione semantica non è stata ancora completata) oppure nessuna regola è stata applicata (l'espansione semantica è stata completata). Nel primo caso, ODB-QOPTIMIZER attiva nuovamente ODL-DESIGNER in modalità Opt_ODLquery inviandogli il file di uscita di GES (GES_OUT.PL) che contiene l'espansione calcolata $\Gamma^i(\mathrm{Q})$ come una nuova query, chiamata Q_i. Nel secondo caso, invece, l'iterazione termina, avendo ottenuto come risultato l'espansione semantica della query $\tilde{\Gamma}(\mathrm{Q})$.

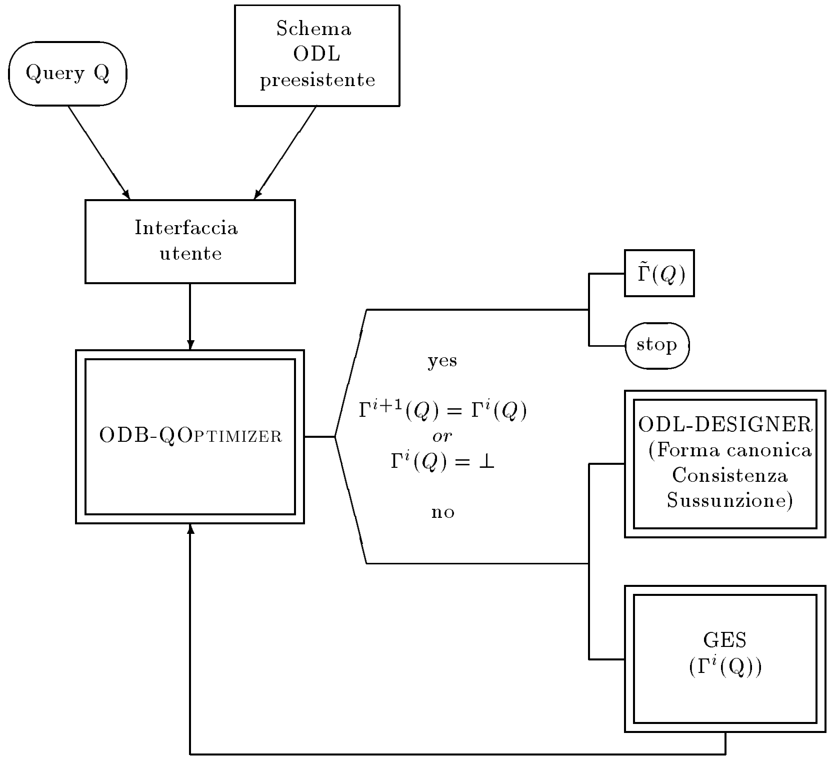

Figura 3: Architettura funzionale di ODB-QOptimizer

4 Sessione di lavoro con ODB-QOptimizer

Vediamo ora un esempio di ottimizzazione semantica utilizzando la query Q_2 vista in sezione 1.1. Faremo riferimento alla forma canonica dei tipi coinvolti nella trasformazione: in particolare in tabella 3 viene riportata la descrizione delle regole in forma canonica. Q_2 è espressa nel formalismo ODL come una classe virtuale:

$$\begin{aligned}\sigma(Q) \;=\; & \texttt{Storage} \sqcap (\triangle.\texttt{category}: \text{“}B4\text{”}) \\ & \sqcap (\triangle.\texttt{managed-by}: \texttt{Manager} \;\sqcap (\triangle.\texttt{level}: \infty \div 10)) \\ & \sqcap (\triangle.\texttt{stock}.\forall.\triangle.\texttt{item}: \texttt{Material} \sqcap (\triangle.\texttt{risk}: 15 \div \infty))\end{aligned}$$

Prima esecuzione dell'espansione semantica (Γ^0(Q))

– ODB-QOptimizer attiva ODL-Designer, in modalità Opt_ODLquery.

ODL-Designer esegue le funzioni F1 e F2 descritte in sezione 3 determinando i valori da inserire nel file Ges_in.txt per l'esecuzione di GES. L'esecuzione di F1 porta alla forma canonica di Q:

$$
\begin{array}{l}
\nu(\texttt{R1a}) = \overline{\texttt{Manager}} \sqcap \triangle \mathrm{T}_8 \\
\qquad \nu(\mathrm{T}_8) = [\texttt{level}\colon 5 \div 10, \texttt{name}\colon \texttt{String}, \texttt{salary}\colon 40K \div \infty\ ] \\
\nu(\texttt{R2a}) = \overline{\texttt{Material}} \sqcap \triangle \mathrm{T}_{10} \\
\qquad \nu(\mathrm{T}_{10}) = [\ \texttt{feature}\colon \forall\{\texttt{String}\}, \texttt{name}\colon \texttt{String}, \texttt{risk}\colon 10 \div \infty\ ] \\
\nu(\texttt{R3a}) = \overline{\texttt{Storage}} \sqcap \triangle \mathrm{T}_{11} \\
\qquad \nu(\mathrm{T}_{11}) = [\ \texttt{category}\colon \text{“}\mathit{B4}\text{”}, \texttt{stock}\colon T_6, \texttt{managed-by}\colon \texttt{Manager}\ ] \\
\qquad\qquad \nu(\mathrm{T}_6) = \forall\{\mathrm{T}_5\} \\
\qquad\qquad\qquad \nu(\mathrm{T}_5) = [\texttt{item}\colon \texttt{Material}, \texttt{qty}\colon 10 \div 300\ ] \\
\nu(\texttt{R4a}) = \overline{\texttt{Storage}} \sqcap \triangle \mathrm{T}_{15} \\
\qquad \nu(\mathrm{T}_{15}) = [\texttt{category}\colon \texttt{String}, \texttt{stock}\colon T_{14}, \texttt{managed-by}\colon \texttt{Manager}] \\
\qquad\qquad \nu(\mathrm{T}_{14}) = \forall\{\mathrm{T}_{13}\} \\
\qquad\qquad\qquad \nu(\mathrm{T}_{13}) = [\texttt{item}\colon \texttt{SMaterial}, \texttt{qty}\colon 10 \div 300\ ] \\
\nu(\texttt{R5a}) = \overline{\texttt{Storage}} \sqcap \triangle \mathrm{T}_{18} \\
\qquad \nu(\mathrm{T}_{18}) = [\texttt{category}\colon \texttt{String}, \texttt{stock}\colon T_{17}, \texttt{managed-by}\colon \texttt{Manager}\ ] \\
\qquad\qquad \nu(\mathrm{T}_{17}) = \exists\{\mathrm{T}_{16}\} \\
\qquad\qquad\qquad \nu(\mathrm{T}_{16}) = [\texttt{item}\colon \texttt{Material}, \texttt{qty}\colon 10 \div 50]
\end{array}
$$

Tabella 3: Forma canonica degli antecedenti delle regole

$$
\begin{array}{l}
\nu(Q) = \overline{\texttt{Storage}} \sqcap \triangle T_{24} \\
\qquad \nu(T_{24}) = [\texttt{category}\colon \text{“}\mathit{B4}\text{”}, \texttt{managed-by}\colon N_3, \texttt{stock}\colon T_{22}] \\
\qquad\qquad \nu(T_{22}) = \forall\{T_{21}\} \\
\qquad\qquad\qquad \nu(T_{21}) = [\texttt{item}\colon N_2, \texttt{qty}\colon 10 \div 300] \\
\qquad\qquad\qquad\qquad \nu(N_2) = \overline{\texttt{Material}} \sqcap \triangle T_{20} \\
\qquad\qquad\qquad\qquad\qquad \nu(T_{20}) = [\texttt{feature}\colon \forall\{\texttt{String}\}, \texttt{name}\colon \texttt{String}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \texttt{risk}\colon 15 \div \infty] \\
\qquad\qquad \nu(N_3) = \overline{\texttt{Manager}} \sqcap \triangle T_{23} \\
\qquad\qquad\qquad \nu(T_{23}) = [\texttt{level}\colon 1 \div 10, \texttt{name}\colon \texttt{String}, \texttt{salary}\colon 40K \div \infty]
\end{array}
$$

L'esecuzione della funzione F2 porta al calcolo delle relazioni di sussunzione tra la forma canonica della query Q e lo schema ODL preesistente. In particolare viene confermata la relazione `Storage` $\in$ `GS`(Q) e vengono calcolate ex-novo da ODL-DESIGNER:

$\mathrm{R}_3^a \in \texttt{GS}(\mathrm{Q})$ poichè il dominio di level in Q $(1 \div 10)$ è più ristretto di quello in R_3^a (pari a quello in `Manager` : $1 \div 15$).

$\mathrm{R}_2^a \in \texttt{GS}(\mathrm{N}_2)$ poichè il dominio di risk in Q è più ristretto di quello in R_2^a (vedi T_{10} e T_{20}).

ODL-DESIGNER non rileva alcun tipo inconsistente e restituisce il controllo a ODB-QOPTIMIZER assieme al file GES_IN.TXT che contiene:

- la forma canonica ν(Q)

- le generalizzazioni di Q e dei tipi che $\in \mathbf{CT}^{+}(\mathrm{Q})$:

$$\begin{array}{lll}
\mathtt{GS}(Q) = \{\mathtt{Storage},\ \mathtt{R3a}\} & \mathtt{GS}(T_{24}) = \{T_{11},\ T_7\} & \mathtt{GS}(T_{22}) = \{T_6\} \\
\mathtt{GS}(N_2) = \{\mathtt{Material},\ \mathtt{R2a}\} & \mathtt{GS}(T_{20}) = \{T_2,\ T_{10}\} & \mathtt{GS}(T_{21}) = \{T_5\} \\
\mathtt{GS}(T_1) = \{\ \} & \mathtt{GS}(N_3) = \{\mathtt{Manager}\} & \mathtt{GS}(T_{23}) = \{T_3\}
\end{array}$$

- i conseguenti delle regole coinvolte (R_2^a e R_3^a) nelle generalizzazioni, espresse nella forma canonica:

$$\begin{array}{l}
\nu(\mathtt{R2c}) = \overline{\mathtt{Material}} \sqcap \overline{\mathtt{SMaterial}} \sqcap \triangle T_2 \\
\qquad \nu(T_2) = [\mathtt{feature}\colon \forall\{\mathtt{String}\}, \mathtt{name}\colon \mathtt{String}, \mathtt{risk}\colon \mathtt{Int}] \\
\nu(\mathtt{R3c}) = \overline{\mathtt{Storage}} \sqcap \triangle T_{12} \\
\qquad \nu(T_{12}) = [\mathtt{category}\colon \mathtt{String}, \mathtt{stock}\colon T_6, \mathtt{managed\text{-}by}\colon N_1] \\
\qquad\qquad \nu(T_6) = \forall\{T_5\} \\
\qquad\qquad\qquad \nu(T_5) = [\mathtt{item}\colon \mathtt{Material}, \mathtt{qty}\colon 10 \div 300] \\
\nu(N_1) = \overline{\mathtt{Manager}} \sqcap \overline{\mathtt{TManager}} \sqcap \triangle T_4 \\
\nu(T_4) = [\mathtt{level}\colon 8 \div 12, \mathtt{name}\colon \mathtt{String}, \mathtt{salary}\colon 40K \div \infty]
\end{array}$$

– ODB-QOptimizer attiva GES che esegue le seguenti operazioni:

- Analisi dei tipi che $\in \mathbf{CT}^{+}(\mathrm{Q})$, da cui emerge che l'unica regola applicabile è sul tipo N_2.

  $R_2^a \in GS(N_2)$; $R_2^c \notin GS(N_2)$; viene quindi applicata R_2 a N_2 ottenendo: $\nu(N_2) = \nu(N_2) \sqcap \nu(R_2^c)$

- Analisi delle generalizzazioni di Q (GS(Q)):

  $R_3^a \in GS(Q)$; $R_3^c \notin GS(Q)$; viene quindi applicata R_3 a Q ottenendo: $\nu(Q) = \nu(Q) \sqcap \nu(R_3^c)$

GES crea una nuova classe virtuale $\mathrm{Q}_1 = \Gamma^0(\mathrm{Q})$ la cui descrizione corrisponde alla nuova $\nu(\mathrm{Q})$ ottenuta in base a tutte le trasformazioni effettuate per i tipi contenuti e ne scrive la descrizione nel file GES_OUT.PL secondo la sintassi ODL-Designer. Durante l'esecuzione di GES su video vengono inviati messaggi che segnalano le regole applicate. In questa prima esecuzione compare su video:

```
Esecuzione di GES
Applico la regola r2
Applico la regola r3
```

– Il controllo ritorna a ODB-QOptimizer che verifica lo stato di ritorno di GES: poiché sono state applicate regole il valore di stato restituito è "continua". Occorre attivare un nuovo ciclo di esecuzioni di ODL-Designer e GES per cercare di espandere ulteriormente la query.

Seconda esecuzione dell'espansione semantica (Γ^1(Q))

– ODB-QOptimizer attiva ODL-Designer in modalità Opt_ODLquery.

Viene letto il file Ges_out.pl inserendo quindi anche la classe Q_1 nello schema, controllata la consistenza di Q_1 rispetto allo schema e generata la forma canonica di Q_1. Infine vengono calcolate le relazioni di sussunzione tra la forma canonica della query Q_1 e lo schema ODL preesistente determinando:
(Q_1, R_4^a, R_3^c) $\in \mathtt{GS}(\mathrm{Q}_1)$ e $\mathrm{R}_1^a \in \mathtt{GS}(\mathrm{N}_7)$

ODL-Designer non rileva alcun tipo inconsistente e restituisce il controllo a ODB-QOptimizer assieme al file Ges_in.txt che contiene:

- la forma canonica $\nu(\mathrm{Q}_1)$.
- le generalizzazioni di Q_1 e dei tipi che $\in \mathbf{CT}^+$(Q):

  $\mathtt{GS}(Q_1) = \{\mathtt{Storage},\ Q,\ \mathtt{R3a},\ \mathtt{R4a},\ \mathtt{R3c}\}$ $\mathtt{GS}(T_{34}) = \{T_{11},\ T_7,\ T_{24},\ T_{12},\ T_{15}\}$
  $\mathtt{GS}(N_5) = \{\mathtt{Material},\ \mathtt{SMaterial},\ \mathtt{R2a},\ \mathtt{R2c},\ N_4,\ N_2\}$ $\mathtt{GS}(T_{28}) = \{T_2,\ T_{10}\}$
  $\mathtt{GS}(N_7) = \{\mathtt{Manager},\ \mathtt{TManager},\ N_1,\ N_3,\ N_6,\ \mathtt{R1a}\}$ $\mathtt{GS}(T_{30}) = \{T_5,\ T_{13},\ T_{21}\}$
  $\mathtt{GS}(T_{31}) = \{T_6,\ T_{14},\ T_{22}\}$ $\mathtt{GS}(T_{33}) = \{T_3,\ T_4,\ T_8,\ T_{23}\}$ $\mathtt{GS}(T_{26}) = \{\ \}$

- i conseguenti delle regole coinvolte nelle generalizzazioni espresse nella forma canonica, $\nu(\mathtt{R1c})$ e $\nu(\mathtt{R4c})$.

– ODB-QOptimizer attiva GES che compie le seguenti operazioni:

- Analisi delle generalizzazioni dei tipi $\in \mathbf{CT}^+$(Q): l'unica regola applicabile è relativa ad N_7:

  $R_1^a \in GS(N_7)$; $R_1^c \notin GS(N_7)$; posso, quindi, applicare R_1 a N_7: $\nu(N_7) = \nu(N_7) \sqcap \nu(R_1^c)$

- Analisi delle generalizzazioni di Q_1 (GS(Q_1)):

  $R_3^a, R_3^c, R_4^a \in GS(Q_1)$; $R_4^c \notin GS(Q_1)$; posso, quindi, applicare R_4 a Q (poichè la R_3 è già applicata): $\nu(Q_1) = \nu(Q_1) \sqcap \nu(R_4^c)$

Si ottiene quindi $\mathrm{Q}_2 = \Gamma^1(\mathrm{Q})$.
– Il controllo è restituito a ODB-QOptimizer che verifica la terminazione: poiché il valore di stato restituito restituito da GES è "continua", occorre attivare un nuovo ciclo di esecuzioni di ODL-Designer e GES per cercare di espandere ulteriormente

la query.

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

ODL-Designer calcola la forma canonica di Q_2 e le relazioni di sussunzione: ($\texttt{SStorage}$, R_1^c, Q_1) $\in GS(Q_2)$
Viene attivato GES che calcola la funzione $\Gamma^2(Q)$ (controllando l'applicabilità delle regole): poiché nessuna regola è applicabile GES restituisce quindi il controllo ad ODB-QOptimizer ritornando il valore "termina". L'espansione semantica della query Q è completata ($\tilde{\Gamma}(Q) = \Gamma^2(Q)$) e su video compare la query ottimizzata:

$$
\begin{array}{lcl}
\underline{\textbf{virtual class}} \;\; Q_2'' & = & \underline{\textbf{isa}}\ \texttt{SStorage}\ \underline{\textbf{and}}\ [\ \texttt{category=}\ \textit{"B4"}, \\
 & & \quad \texttt{managed-by}: \underline{\textbf{isa}}\ \texttt{TManager}\ \underline{\textbf{and}}\ (\texttt{level}\ 8 \div 10) \\
 & & \qquad\qquad\qquad\quad \underline{\textbf{and}}\ (\texttt{salary}\ : 40K \div 60K), \\
 & & \quad \texttt{stock.}\forall\texttt{.item}: \underline{\textbf{isa}}\ \texttt{SMaterial}\ \underline{\textbf{and}}\ (\texttt{risk} \geq 15)\ ]
\end{array}
$$

5 Conclusioni e lavoro futuro

In questo lavoro è stato presentato un metodo di ottimizzazione semantica delle interrogazioni in ambito OODB. Questo metodo è basato sul formalismo ODL che presenta e descrive lo schema, i vincoli d'integrità e le interrogazioni, ed è basato sulle tecniche di inferenza introdotte nelle Logiche Descrittive in ambito Intelligenza Artificiale. È stato, inoltre, presentato il prototipo ODB-QOptimizer, che implementa il metodo di ottimizzazione semantica di interrogazioni proposto.

Riferimenti bibliografici

[1] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[2] P. Atzeni, editor. *LOGIDATA+: Deductive Databases with Complex Objects.* Springer-Verlag: LNCS n. 701, Heidelberg - Germany, 1993.

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

[4] F.M. Donini, A. Schaerf, and M. Buchheit. Decidable reasoning in terminological knowledge representation systems. In *13th International Joint Conference on Artificial Inteligence*, Chambery - France, September 1993.

[5] M.M. Hammer and S. B. Zdonik. Knowledge based query processing. In *6th Int. Conf. on Very Large Databases*, pages 137–147, 1980.

[6] W. Kim. A model of queries for object-oriented database systems. In *Int. Conf. on Very Large Databases*, Amsterdam, Holland, August 1989.

[7] J. J. King. *Query optimization by semantic reasoning.* PhD thesis, Dept. of Computer Science, Stanford University, Palo Alto, 1981.

[8] 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.

[9] 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.

[10] 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.