

Prototipo per la Creazione di Content Summary

Paolo Ciaccia, Alessandro Linari, Marco Patella

DEIS - Università di Bologna

1 Introduzione

Il progetto WISDOM si è occupato dell'interrogazione di sorgenti di dati eterogenee in ambito peer-to-peer; un problema critico in tale scenario è rappresentato dalla scelta delle sorgenti che devono essere contattate per rispondere ad una interrogazione. Tale scelta è guidata, primariamente, da criteri che tengono in considerazione il *numero* di risultati che la sorgente può restituire e la loro *qualità* rispetto alla query. Lo scopo di questo articolo è di fornire la documentazione del prototipo del Content Summary Manager (CSM) [Las05, Man06], componente software che crea e gestisce i Content Summary (CS), strutture dati che descrivono in termini statistici il contenuto informativo di una sorgente. I Content Summary sono riferiti ai Semantic Mapping (SM) e sono utilizzati dal Cardinality Competence Center per stimare la rilevanza di una sorgente. Nel seguito si farà riferimento ai seguenti concetti di Content Summary e Semantic Mapping:

- **Content Summary.** Un Content Summary è una stima del numero di risultati ottenibili percorrendo un determinato mapping. Un **local CS (LCS)** caratterizza un mapping fra la GVV e una qualsiasi sorgente locale e un **remote CS (RCS)** caratterizza un mapping fra due peer.
- **Semantic Mapping.** Un Semantic Mapping è un unfolding in stile GAV [Hal01] esistente tra classi MOMIS [BBGV03] presenti su due diversi peer.

Il Content Summary Manager (CSM) è il componente software che si occupa della gestione dei Content Summary, assicurandone la creazione a partire dai mapping locali specificati nella Global Virtual View (GVV), il reperimento quando essi sono riferiti a mapping con altri peer e la loro successiva interrogazione (e integrazione) per ottenere una valutazione quantitativa dei rewriting [Las05]. Il "cliente" del CSM è il Cardinality Competence Center (CCC) che, combinando opportunamente i CS relativi ad uno stesso rewriting, è in grado di generare una previsione della distribuzione dei risultati, sulla quale è basata la valutazione del rewriting da parte del query processor [DiC06, Cia06].

2 Il Cardinality Competence Center

Il Cardinality Competence Center (CCC) riceve in input un rewriting espresso in forma interna e lo valuta in base alle preferenze specificate nella query. Dato un insieme di preferenze e una priorità fra di esse, la valutazione di un rewriting consiste nella creazione di una lista (Ranked List) di intervalli di preferenze ordinati in base alla loro priorità. A ciascun intervallo, quindi, è associata la stima del numero di risultati che ricadono in tale intervallo, che viene calcolato interrogando opportunamente il CSM [Man06]. L'interfaccia del CCC, pertanto, è esprimibile come segue:

RankedList getRL(Rewriting r , int $card_min$, double α , double β)

dove r è un'espressione in formato interno su cui sono specificate delle preferenze di priorità e $card_rl$, α e β sono altri parametri che saranno spiegati a breve.

RankedList
-VarPref[1..n]:VariabileRel
-EN[1..m]:Ennupla
-Eval[1..m]:float
+getVar(i:int) : VariabileRel
+getEn(i:int) : Ennupla
+getEval(i:int):float

L'attributo più importante della classe *RankedList* è l'oggetto di tipo **ennupla**, un vettore i cui elementi rappresentano una configurazione di valori per le variabili su cui è stata espressa una preferenza. Gli elementi sono ordinati in base alla priorità di ciascuna configurazione: la prima ennupla contiene i valori di chiave *migliori* per ciascuna variabile target, nella seconda la variabile meno prioritaria assume un valore *meno importante* e così via, diminuendo progressivamente la priorità di ciascuna configurazione nella lista. A ciascuna ennupla, inoltre, è associata la stima del numero di oggetti presenti nella sorgente con la data priorità (calcolata, naturalmente, mediante il relativo CS).

Una *RankedList* è modellata mediante un istogramma equi-depth (approssimato): l'approssimazione deriva dal fatto che ciascuna ennupla contiene *circa* lo stesso numero di oggetti, dipendentemente dalla distribuzione degli oggetti contenuti nella sorgente nello spazio degli attributi. Il parametro $card_min$ rappresenta il numero minimo di oggetti in ciascuna ennupla, β la tolleranza su tale valore (il numero di oggetti della ennupla non deve essere superiore a $\beta * card_min$). Durante il processo di creazione della *RankedList*, il CCC divide lo spazio individuato dalle variabili di preferenza in α regioni e, interrogando i CS forniti dal CSM, stima, per ciascuna di esse, il numero di oggetti che sono contenuti nella sorgente. Se almeno uno dei valori di cardinalità così ottenuti differisce per più di un fattore β dal valore minimo $card_min$, il CCC attiva una opportuna procedura di tuning per modificare, in numero e in ampiezza, le regioni della *RankedList*.

3 Architettura del Content Summary Manager

Dato un qualsiasi mapping sm , il content summary $CS(sm)$ è costruito generando una query nel linguaggio della sorgente sottostante. Se la sorgente è locale (si sta creando un LCS), la query viene eseguita direttamente, mentre se l'obiettivo è creare un RCS, i casi sono due:

- Se il peer remoto è provvisto di un CSM, la query è inoltrata direttamente ad esso. Il CSM remoto, in questo caso, crea il relativo CS e lo restituisce al peer che ha iniziato la richiesta;
- Se il peer remoto non ha un CS Manager (sorgente classica), il sistema “stima” il Content Summary generando una serie di query di test e collezionandone i risultati (questa tecnica è chiamata probing [GIS03]).

Pur avendo previsto, a livello architetturale, questa seconda possibilità, d'ora in avanti si supporrà che il peer remoto disponga sempre di un CSM. In Figura 1, l'architettura di alto livello del CSM.

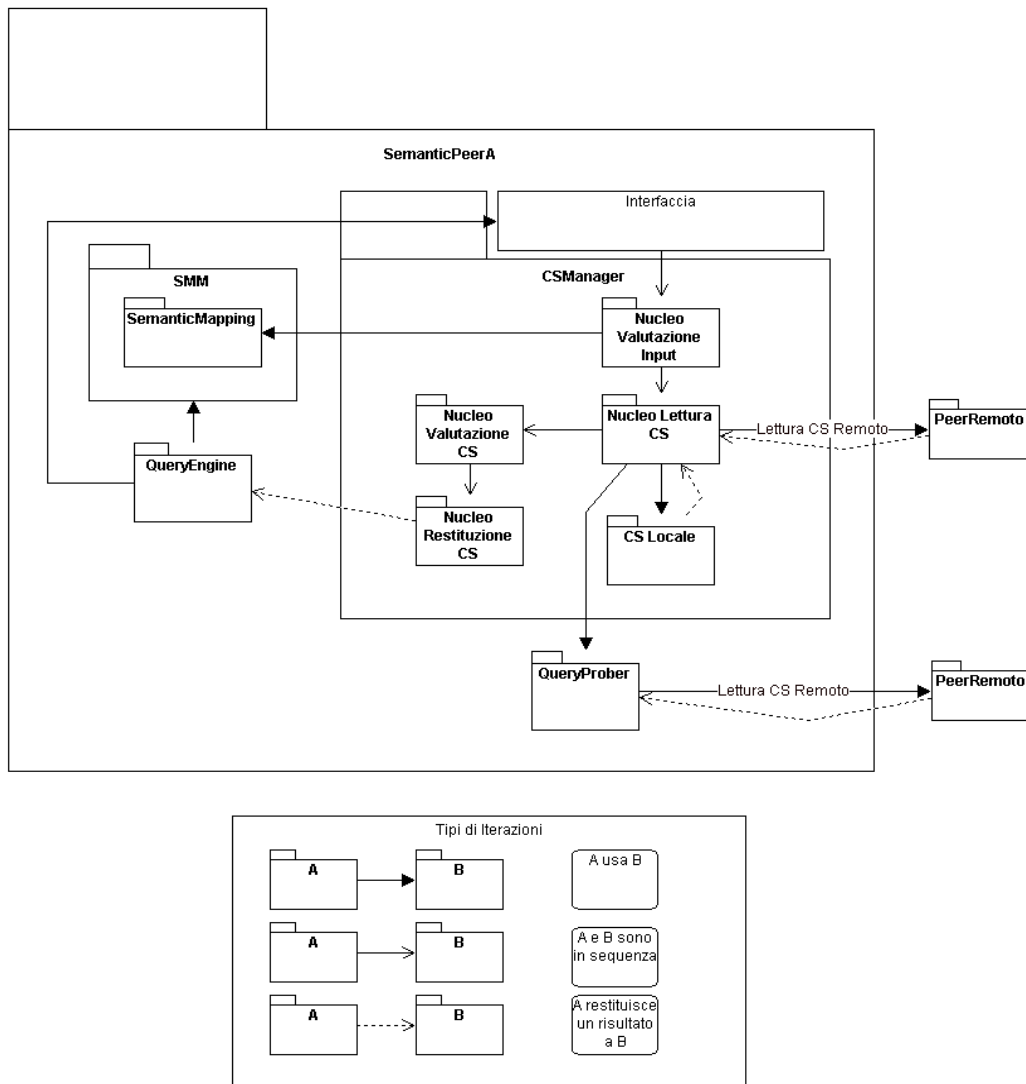


Figura 1: Architettura del Content Summary Manager

Quando una richiesta per un CS arriva al CSM, questa viene inoltrata al *nucleo di valutazione degli input*, che la analizza e la trasforma in una query; successivamente, il *nucleo lettura CS* decide se interrogare una sorgente locale, se inoltrare la richiesta a un peer remoto o se utilizzare una tecnica di probing. Infine, i dati ottenuti dalla precedente interrogazione vengono opportunamente trasformati e restituiti all'utente.

3.1 Semantic Mapping

Un Semantic Mapping è una regola di riscrittura fra schemi differenti ed è composto da un *head*, da un *body* e da un insieme di *binding element*. L'**head** è la porzione della GVV locale interessata dal mapping, il **body** è la porzione di schema della sorgente locale o, alternativamente, della GVV del peer remoto con la quale si vuole stabilire un legame e i **binding element** sono gli elementi fra cui c'è corrispondenza fra *head* e *body* e che, quindi, rappresentano la regola di trasformazione. L'*head* è un predicato atomico, mentre il *body* è un'espressione congiuntiva di più predicati. Ogni predicato contiene un insieme di elementi, ciascuno caratterizzato da nome, percorso (o URL), tipo di dato (ad esempio testuale o numerico) e tipologia (ruolo assunto nel mapping).

Un elemento *head* può essere di due tipologie:

- **Binding element**, se ha un corrispondente nel *body* ed è, quindi, coinvolto nel mapping;
- **Indifferente**, se non ha corrispondente nel *body* e, quindi, non è coinvolto nel mapping.

Un elemento nel *body* può essere di quattro tipologie (Figura 2):

- **Binding element**, se ha un corrispondente nell'*head* (coinvolto nel mapping);
- **Indifferente**, se non ha corrispondente nell'*head* e non è coinvolto nel mapping;
- **Join element**, se non ha corrispondente nell'*head* ma esprime un join con un altro elemento nel *body*;
- **Costante**, se deve assumere un valore costante affinché il mapping sia verificato.

Per concludere l'analisi, vale la pena notare che questa soluzione continua ad essere valida sia con il modello di dati relazionale, sia con il modello XML.

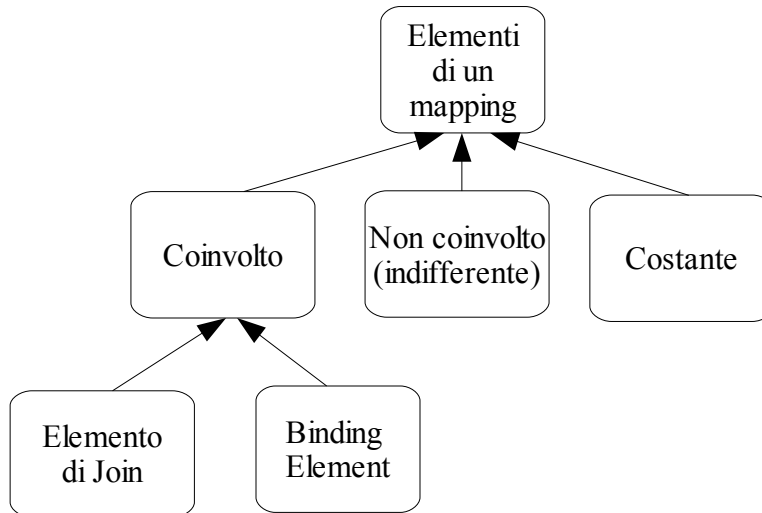


Figura 2: Tipologie di elementi presenti in un Semantic Mapping

3.1.1 Modello di Dati Relazionale

In questo modello i predicati sono *relazioni* e gli elementi sono i loro *attributi*. Le relazioni sono identificate da un nome e sono formate da un *idpeer* (che identifica in maniera univoca il peer in cui si trova la relazione) e da un insieme di attributi. Gli attributi coinvolti nei semantic mapping sono caratterizzati da un nome, da un percorso, da un valore (inerente al mapping), dal tipo di dato che memorizzano e dalle proprietà che li caratterizzano (per esempio, valori costanti o discreti). Per chiarire questi concetti si propone il seguente esempio:

Date le relazioni:
Peer1 → Articoli (Autore, Titolo, ParoleChiavi, Pdf)
Peer2 → Testi (IdAuthor, Title, Abstract, Tipo)
 Autori (Nome, Id, AnnoNascita)
SemanticMapping
Peer1.Articoli(A, T, -, -) ← Peer2.Testi(Id, T, -, "article"),
Peer2.Autori(A, Id, -)

Nell'esempio, la prima relazione è caratterizzata dal nome "*Articoli*", è formata da un idpeer "*Peer1*" (che identifica il peer che contiene la relazione) ed è composta da un insieme di attributi. L'attributo Autore, oltre al nome, è caratterizzato dal percorso "*Peer1/Articoli*" e dal valore "*A*" nel mapping (indica che l'attributo è coinvolto nel mapping, infatti posso trovare lo stesso valore anche nell'attributo Nome nella relazione Autori nel Peer2). L'attributo Autore memorizza elementi di *testo* ed è un attributo *discreto*. Invece l'attributo Anno, descritto dal nome "*Anno*", ha come percorso "*Peer1/Articoli*" e valore "*-*" (che indica un attributo indifferente al mapping). Questo attributo ha dominio *intero* ed è un attributo *discreto*.

3.1.2 Modello di Dati XML

In questo modello un predicato è rappresentato da una espressione XML: questa è identificata da un nome e caratterizzata da un idpeer (che identifica in maniera univoca il peer in cui si trova la relazione) e da un *AlberoXML* composto da *nodiXML*. Ogni *nodoXML* è formato da un nome, da un percorso, da un valore nel mapping, da un valore (distintivo del nodo), dal tipo di dato che memorizza, dalle proprietà che lo caratterizzano e da un insieme di attributi. Gli attributi hanno una struttura del tutto analoga a quella già vista per il caso relazionale. Nel modello XML devono essere considerati *elementi* sia i *nodiXML* che gli attributi. Infatti, un mapping può essere stabilito anche fra essi. Se ad esempio consideriamo i seguenti mapping,

SemanticMapping
Attributo *Peer1/Articoli/Autore/Nome* “A” → Nodo *Peer2/Autori/Nome* “A”
Nodo *Peer2/Testi/Tipo* → “article” (costante)
Nodo *Peer2/Testi/IdAuthor* “Id” → *Nodo Peer2/Autori/Id* “Id” (join)

ad essi può essere associata la seguente rappresentazione ad albero di Figura 3:

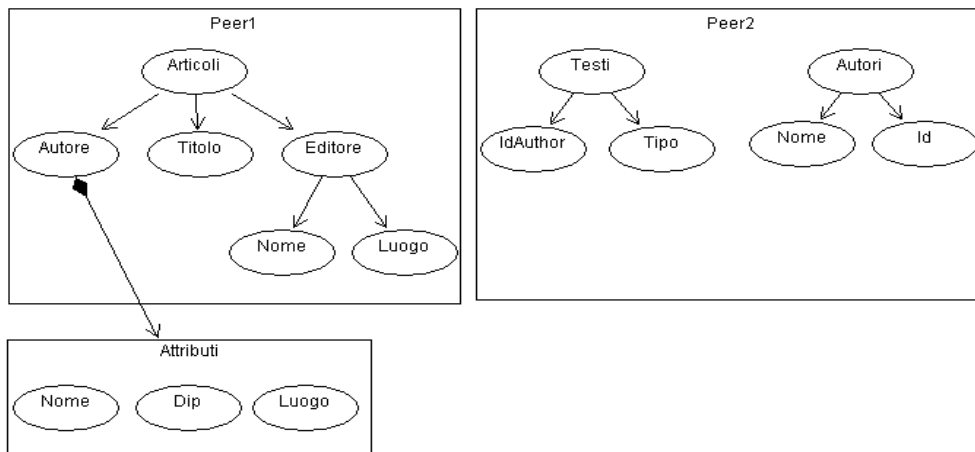


Figura 3: Esempio di Semantic Mapping rappresentato ad albero

In Figura 3, la prima *espressioneXML* è caratterizzata dal nome “Articoli” è formata da un idpeer “Peer1” (che identifica il peer che contiene la relazione) ed è composta da un *alberoXML* di nodi. L’attributo Nome (coinvolto nel mapping) è caratterizzato dal nome “Nome”, dal percorso “*Peer1/Articoli/Autore*” e dal valore “A” nel mapping (indica che l’attributo è coinvolto nel mapping, infatti è possibile trovare lo stesso valore anche nel Nodo “*Peer2/Autori/Nome*”). L’attributo Autore

memorizza elementi di *testo* ed è un attributo *discreto*. Invece il nodo Tipo è caratterizzato dal nome “Tipo”, da un percorso “Peer2/Testi” e dal valore “article” nel mapping (che indica che il nodo è una costante). Questo nodo, inoltre, ha dominio *testo*, ha valore *discreti* e non ha nessun attributo.

3.2 Content Summary

I Content Summary studiati nell’ambito del progetto WISDOM ricadono in tre tipologie:

- **CS di Selezione pura.** Questo tipo di CS rappresenta un istogramma sui valori dell’attributo *A* ed è utilizzato dal CCC per stimare il numero di oggetti della sorgente aventi un particolare valore di attributo. In pratica, *riempiono* gli intervalli individuati dalla *RankedList*. Facendo riferimento al modello relazionale, i CS di selezione sono generati dalla seguente query:

```
SELECT A, COUNT (*)
FROM R
GROUP BY A
(più eventuali clausole WHERE)
```

- **CS di conteggio.** Il CS di conteggio restituisce il numero di valori distinti dell’attributo (o dell’insieme di attributi) su cui è costruito. Il CSM può costruire istogrammi multidimensionali sia su una singola che su diverse classi, tuttavia non sempre questa tecnica viene applicata, perché inefficiente o poco praticabile: si pensi, ad esempio, ad un join che coinvolge più sorgenti. In questi casi, il CSM può *simulare* il CS multidimensionale applicando un modello di join basato, appunto, sulla conoscenza del numero di *valori distinti* degli attributi coinvolti [Las05]. Si noti che questo tipo di CS viene utilizzato anche nel CCC, nel caso un join coinvolga più mapping [Man06]. Nel modello relazionale, i CS di conteggio sono generati dalla seguente query:

```
SELECT COUNT(DISTINCT B)
FROM R
(più eventuali clausole WHERE)
```

- **CS di Selezione mista.** I CS di selezione mista costruiscono un istogramma che, per ciascun valore dell’attributo *A*, riporta in numero di valori distinti associati a *B*. Questo tipo di CS, di uso meno comune, possono essere utilizzati sia per stimare la selettività di un join sull’attributo *B* quando è specificato un vincolo su *A*, sia per rispondere a query più complesse come, ad esempio, “Dimmi quanti autori (B) hanno pubblicato negli ultimi 5 anni (A)”. La corrispondente query SQL è:

```

SELECT A, COUNT (DISTINCT B)
FROM R
GROUP BY A
(più eventuali clausole WHERE)

```

Come è stato appena suggerito, i Content Summary possono essere rappresentati mediante istogrammi. Vale la pena sottolineare che, se un mapping coinvolge più di un binding element, il CS risultante può essere rappresentato da un unico istogramma multidimensionale o da una qualsiasi combinazione di istogrammi a dimensionalità ridotta.

4 Progetto del Content Summary Manager

In questa sezione sono presentati i Semantic Mapping e i Content Summary in quanto elementi più significativi del CSM.

4.1 Progetto dei Semantic Mapping

Un oggetto *SemanticMapping* è composto da un *head*, da un *body* e da un insieme di *binding element*. L'*head* è una singola espressione, mentre il *body* è un insieme di espressioni. In Figura 4 si riporta il diagramma UML per il progetto di un Semantic Mapping.

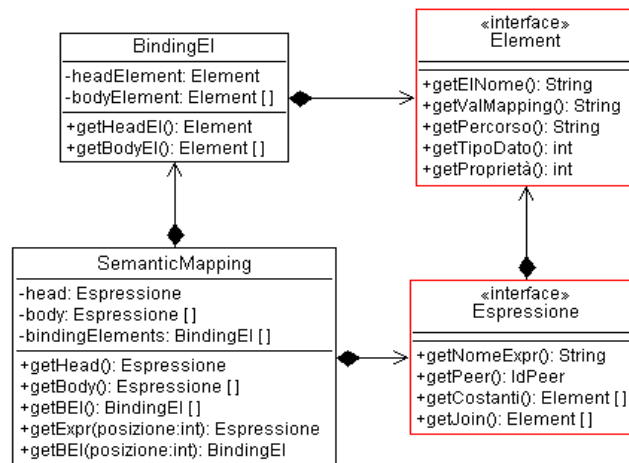


Figura 4: Interfaccia di un Semantic Mapping

Fra i selettori, *getExpr(int posizione)* restituisce l'espressione e *getBEI (int posizione)* il binding element passato in input. Poiché un bindingElement è una relazione fra più elementi, un oggetto *BindingElement* è composto da *headElement* e da un insieme di *bodyElement*. Infine, affinché la definizione di Semantic Mapping sia indipendente dall'implementazione, gli oggetti *Element* e *Espressione* sono stati definiti come interfacce. In particolare, l'interfaccia *Element* è caratterizzata da

cinque metodi che restituiscono, rispettivamente, il nome dell'elemento, la sua tipologia rispetto al mapping, il percorso, il dominio dei dati (discreto o continuo) e il tipo di dato (intero, decimale, etc).

Nel **modello relazionale**, l'interfaccia *Espressione* è realizzata dalla classe *Relazione* e *Element* dalla classe *Attributo*:

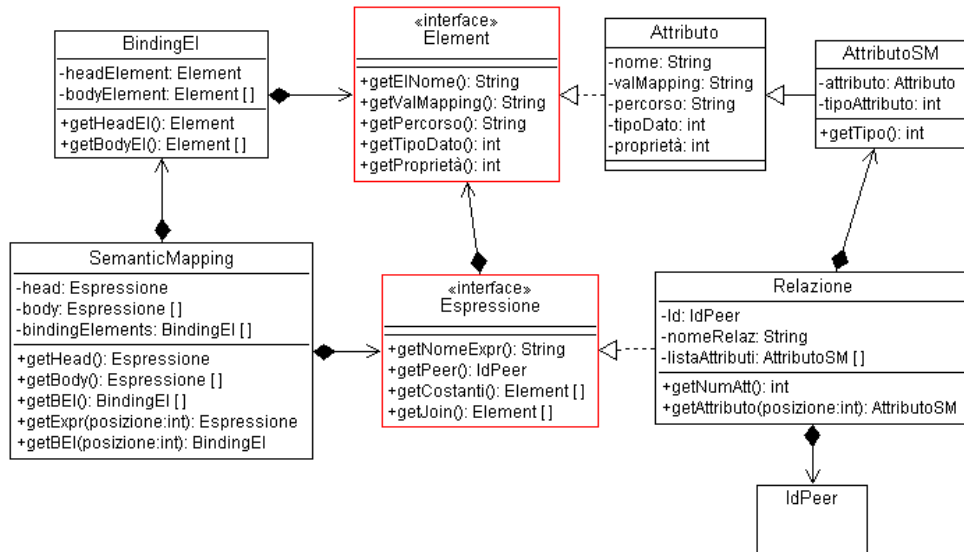


Figura 5: Semantic Mapping nel modello relazionale

Una *Relazione* ha un nome, un *IdPeer* e un array di *AttributoSM* (cioè da particolari tipi di *Attributi*). Gli *Attributi*, a loro volta, hanno un nome, un percorso, un tipo di dominio e un tipo di dato (sotto forma di selettore numerico). La classe *AttributoSM*, specializzazione della classe *Attributo*, aggiunge un selettore numerico per individuare la tipologia di attributo nel mapping.

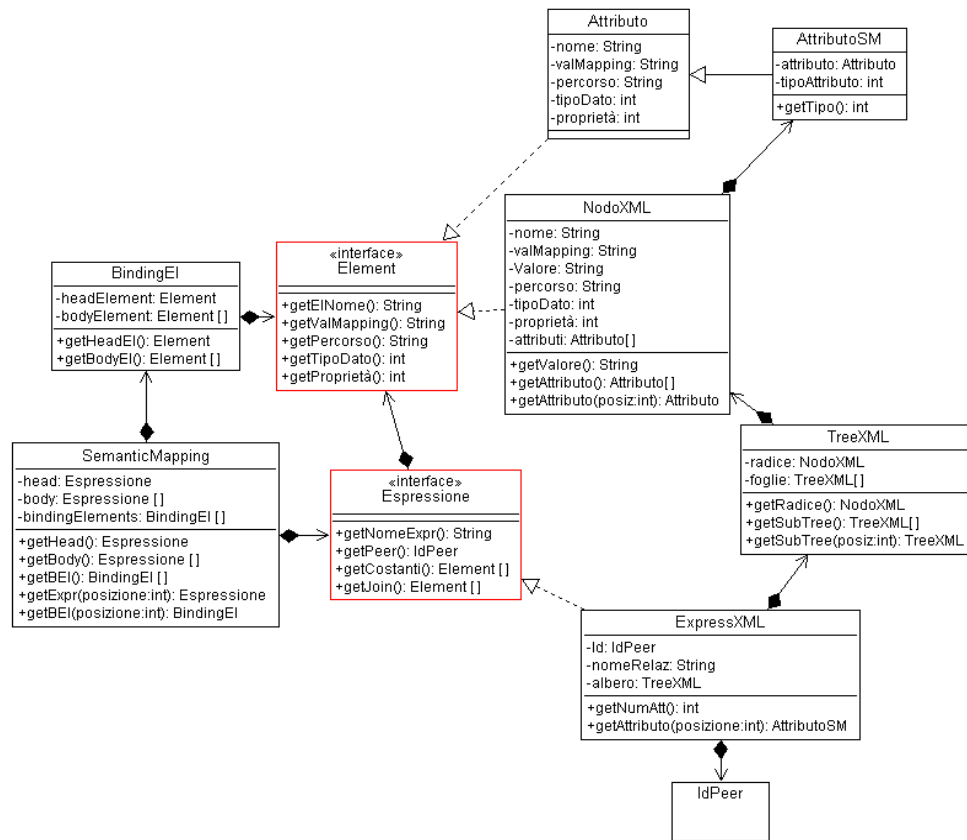


Figura 6: Semantic Mapping nel modello di dati XML

Facendo riferimento al **modello di dati XML** (vedi Figura 6), l'interfaccia *Espressione* è implementata dalla classe *ExpressXML* e *Element* dalle classi *Attributo* e *Nodo* (si ricorda che il mapping può essere espresso su entrambi). Un' *ExprXML* è formata da un *AlberoXML*. Questo, a sua volta, è composto da un nodo radice e da un array di *AlberoXML*. Un *Nodo* è caratterizzato da un valore (su cui eventualmente fare join) e da un array di attributi. La classe *Attributo*, infine, è analoga al caso relazionale.

4.2 Progetto dei Content Summary

L'interfaccia principale del Content Summary Manager è:

ContentSummary getValutazione (Interrogazione[] i1)

che, dato un Array di oggetti Interrogazione, restituisce il corrispondente Content Summary.

Nel diagramma delle classi in Figura 7, si può notare che l'interfaccia definita come *IContentSummary* è stata implementata mediante la classe *ContentSummary* che, a sua volta, è stata modellata come un Array di *Summary*. L'interfaccia *Summary* può adattarsi a molteplici situazioni reali, in particolare è possibile gestire contemporaneamente istogrammi di varie dimensionalità.

L'interfaccia *Summary* definisce i selettori *getNome()* (nome dell'oggetto) e *getDimensione()*, che ne restituisce la dimensionalità e prevede il metodo *getValutazione(Interrogazione i1)*, che restituisce il Summary valutato e *restituzioneVal()* che fornisce un valore intero in output, nel caso la struttura dati abbia dimensionalità pari a 0.

Infine, gli oggetti *Istogramma* sono caratterizzati da un nome, un array di *Element* (individua quali elementi sono descritti), un array di valori per ogni *Element*, un selettore numerico per la tipologia del dominio di ciascun *Element*, un array (di dimensione pari alla lunghezza dell'array degli *Element*) che indica le stime numeriche per ogni *Valore*.

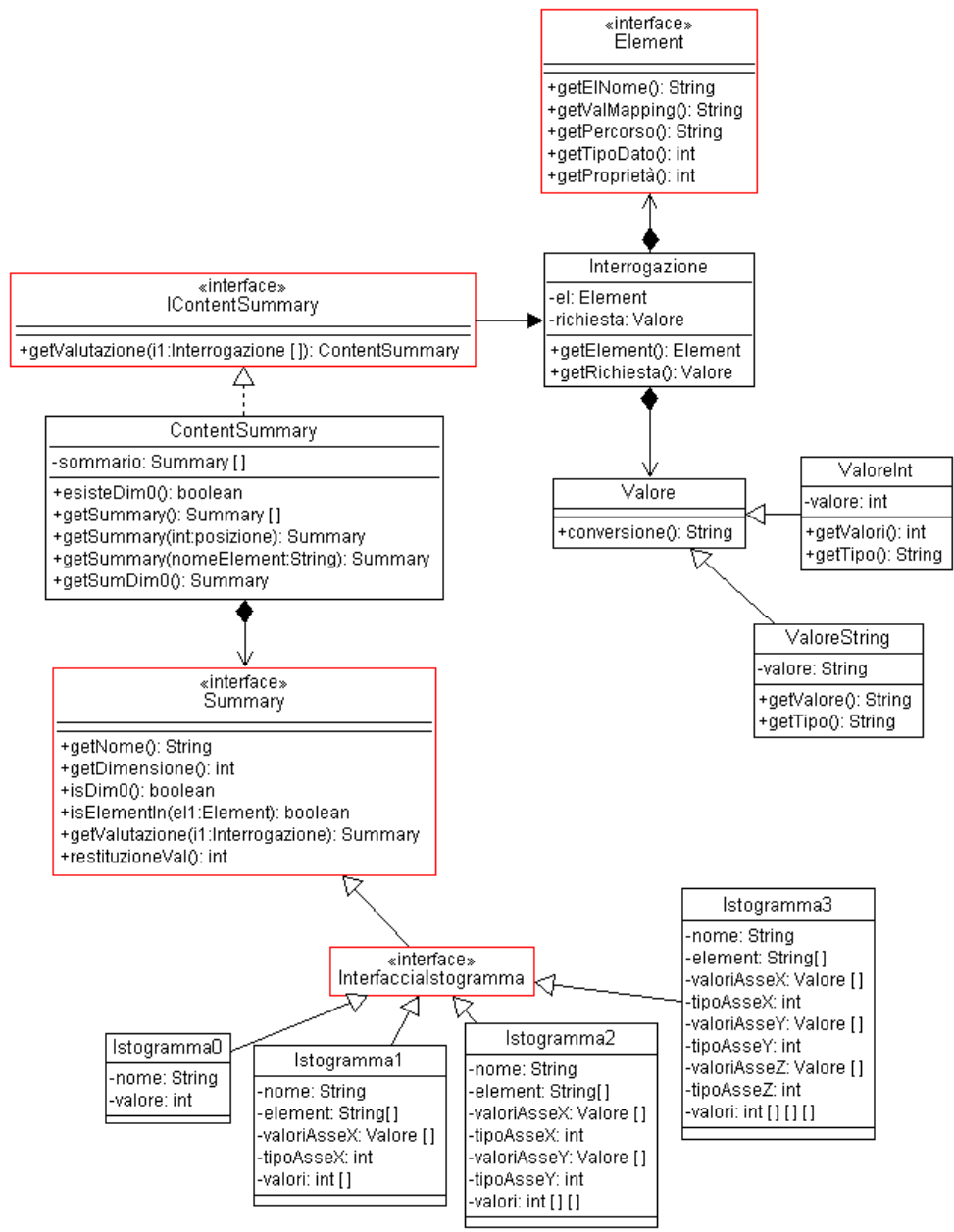


Figura 7: Content Summary, interfaccia e implementazione

Quando il query processor richiede un CS al CSM deve essere in grado di scegliere la tipologia di CS desiderata (selezione pura, conteggio e selezione mista): in Figura 7 è stata introdotta una classe *CSType*, formata da un selettore numerico (per il tipo di CS) e da un campo *Element*, obbligatorio per i CS di conteggio e di selezione mista, che indica l'attributo su cui effettuare il conteggio (nel modello relazionale è l'attributo nel campo COUNT).

4.3 Creazione del Content Summary

Il query processor richiede un CS facendo riferimento all'head di un semantic mapping, mentre il CS viene costruito a a partire dal body, rispetto al quale sono espresse le sorgenti. Il compito del CS Manager, dunque, è anche quello di riformulare la richiesta in modo da ottenere una query che sia espressa rispetto allo schema locale. Facendo riferimento al modello di dati relazionale, si possono scrivere le seguenti regole di “traduzione”:

SELECT	→	Element (Attributo nel caso relazionale)
COUNT	→	è presente un DISTINCT con un Element se si ha CS di conteggio, o un CS di selezione mista (desumibile dal CSType), o “*”, altrimenti
FROM	→	espressione del Body
GROUP BY	→	Element (supposto unico dalle supposizioni precedenti)
WHERE	→	gli Element costanti si tramutano in clausole WHERE e, se il valore di un attributo è comune a più relazioni (sempre nel body), si traduce in un join (ovviamente se gli attributi sono su peer diversi sarà compito del CS Manager spezzare la query).

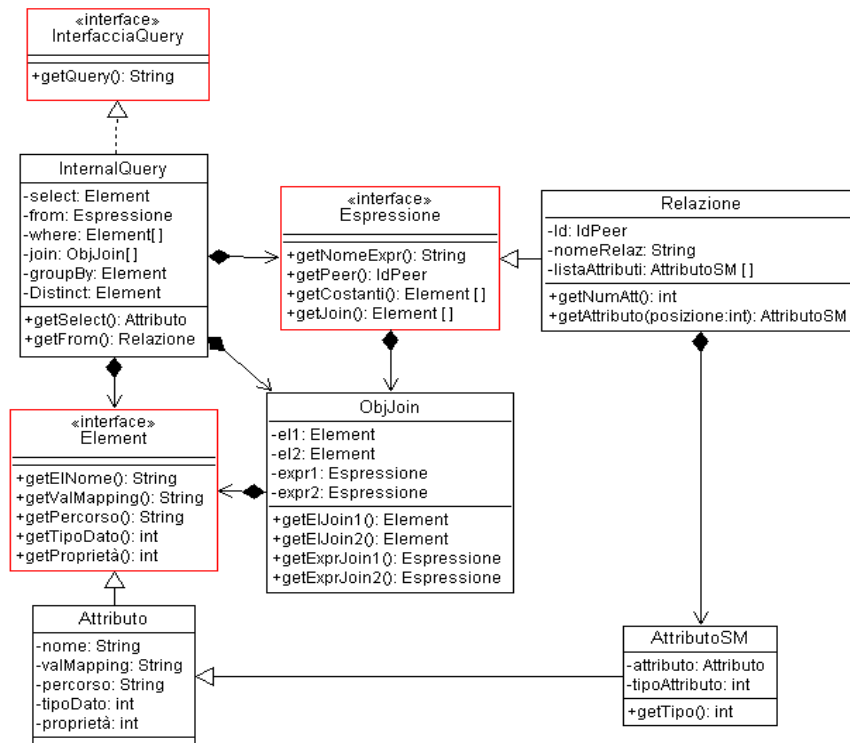


Figura 8: Creazione di un CS nel modello di dati relazionale

In Figura 8, la classe che implementa la query locale è chiamata, per comodità, *InternalQuery*: è composta da un *Element* (clausola SELECT), da un' *Espressione* (clausola FROM), da un insieme di *Element* costanti (clausole WHERE), da un insieme di *ObjJoin* (clausole di join) e da altri due *Element* (per il GROUP BY e per il DISTINCT). Le clausole WHERE e i join sono separati perché memorizzano al loro interno informazioni differenti. Questi ultimi, infatti, contengono informazioni sugli *Element* coinvolti e sulle *Espressioni* che li contengono.

4.4 Interfacce del Content Summary Manager

L'interfaccia *creaCS*,

ContentSummary creaCS (SemanticMapping sm, Element[] elem, CStype type)

viene utilizzata per creare un CS riferito al semantic mapping *sm*. Il CS deve essere di tipo *type* e deve essere costruito sugli attributi specificati in *elem[]*. Infine, affinché due peer possano comunicare direttamente, è definita la seguente interfaccia,

ContentSummary creaCS (InternalQuery q1)

invocata da un peer remoto passando, come argomento, la richiesta espressa rispetto allo schema locale.

Bibliografia

- [BCLP05] I. Bartolini, P. Ciaccia, A. Linari, M. Patella. *Critical analysis of query processing techniques for heterogeneous environments*. Rapporto Tecnico D3.R2, WISDOM (Web Intelligent Search based on DOMain ontologies) - Progetto MIUR, 2005. Disponibile all'URL <http://dbgroup.unimo.it/wisdom/>
- [Cia06] P. Ciaccia et al. *Definition of the ontology-based query language and query processing techniques*. Rapporto Tecnico D3.R3, WISDOM (Web Intelligent Search based on DOMain ontologies) - Progetto MIUR, 2005. Disponibile all'URL <http://dbgroup.unimo.it/wisdom/>
- [BBGV03] D. Beneventano, S. Bergamaschi, F. Guerra, M. Vincini. *Synthesizing an Integrated Ontology*. In IEEE Internet Computing Magazine, Vol. 7, Num. 5, 2003.
- [DiC06] M. Di Carlo. *Un modello per la rappresentazione in forma interna e la riscrittura di interrogazioni distribuite*. Tesi di laurea specialistica presso l'Università degli Studi di Bologna, facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 2006.
- [GIS03] L. Gravano, P. G. Ipeirotis, M. Sahamadi. *QProber: A system for automatic classification of hidden-web databases*. In ACM TOIS, Vol. 21, Num. 1. 2003.
- [Hal01] A. Y. Halevy. *Answering queries using views: A survey*. In VLDB Journal, Vol. 10, Num. 4, 2001.
- [Las05] G. Lasagni. *Gestione dei content summary nel progetto WISDOM*. Tesi di laurea presso l'Università degli Studi di Bologna, facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 2005.
- [Man06] G. Mandarini. *Un framework per la gestione di statistiche per query con preferenze*. Tesi di laurea presso l'Università degli Studi di Bologna, facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, 2006.