Basi di Dati Semistrutturate e XML[]

Carlo Sartiani

Dipartimento di Informatica

Corso Italia 40, Pisa, ITALY

e-mail:sartiani@di.unipi.it

Abstract

Le basi di dati semistrutturate e i documenti XML data-oriented presentano molte somiglianze; la convergenza in atto tra i due ambiti di ricerca ha permesso ai ricercatori sulle basi di dati di applicare ai documenti XML data-oriented quanto già sviluppato per i dati semistrutturati, beneficiando, nel contempo, di alcuni risultati ottenuti dalla comunità XML.

Lo scopo di questo articolo consiste nell'esaminare le principali problematiche relative a basi di dati semistrutturate e documenti XML data-oriented, nonché descrivere le soluzioni sviluppate nelle comunità di ricerca interessate, con particolare riferimento ai modelli dei dati, ai linguaggi di interrogazione e ai meccanismi di schematizzazione logica.

1 Introduzione

Il rapporto tra basi di dati e XML non è univocamente caratterizzabile. Esso si manifesta, infatti, sotto tre diversi aspetti principali, su ognuno dei quali comunità spesso distinte di ricercatori, utenti e sviluppatori concentrano la propria attenzione.

Il primo aspetto è relativo alla gestione di collezioni di documenti XML, ognuno dei quali viene visto come un singolo dato; all'interno di queste collezioni si vogliono eseguire operazioni di ricerca, che possono coinvolgere sia il contenuto dei documenti stessi sia la particolare marcatura impiegata. Sebbene questa applicazione ricada, a nostro avviso, nell'ambito dello *information retrieval* piuttosto che della gestione di basi di dati, è doveroso osservare che molti prodotti commerciali utilizzano, a tale scopo, motori per basi di dati (tipicamente per basi di dati a oggetti).

Il secondo aspetto, che sta divenendo sempre più importante, concerne il *database publishing*, espressione con la quale ci si riferisce alla pubblicazione su Web di informazioni estratte da una base di dati: ad ogni pagina è associata una *query* di definizione, che può essere stata fissata staticamente dal sistema (e.g., le pagine del "Calepino" di Mediobanca)

[] Questo lavoro è stato realizzato con il parziale contributo dei fondi del progetto MURST DATA-X.

oppure essere formulata dall‘utente attraverso opportune *form*. Gli attuali sistemi di database publishing, essendo destinati ad un‘utenza “umana”, inseriscono le informazioni estratte dal database in pagine HTML; l‘impiego di XML e XSL, aumentando il grado di separazione tra presentazione grafica e contenuto delle pagine, potrà agevolare la fruizione dei dati da parte di applicazioni (si pensi, ad esempio, a tutte quelle applicazioni che sfruttano informazioni estratte da pagine HTML mediante l‘uso di *wrapper*).

Il terzo aspetto, che più degli altri ha suscitato l‘interesse presso la comunità dei ricercatori, è relativo all‘uso di XML nell‘ambito delle basi di dati semistrutturate [1]. Una base di dati semistrutturata è caratterizzata da un‘elevata instabilità e/o irregolarità strutturale, che ne rendono la gestione mediante sistemi relazionali e/o *object-oriented* estremamente difficoltosa se non, addirittura, impossibile.

Esempio 1

Supponiamo di dover gestire una collezione di dati sperimentali relativi alla fusione fredda, prodotti da molti gruppi di ricerca indipendenti. Ogni gruppo di ricerca rappresenta le informazioni in modi diversi; l‘integrazione di questi dati in una base di dati centralizzata richiede di trattare opportunamente le ovvie irregolarità strutturali. Ogni gruppo di ricerca, inoltre, può decidere improvvisamente di modificare la struttura dei dati prodotti, il che può imporre la necessità di algoritmi di aggiornamento dello schema molto efficienti.

Tre sono le principali sorgenti di dati semistrutturati. Vi sono, innanzitutto, dati di per sè semistrutturati, quali le evidenze sperimentali (in particolare, i dati biologici [2]) e le collezioni di riferimenti BibTeX. La semistrutturazione dei dati si può presentare, inoltre, nell‘integrazione di dati provenienti da sistemi *legacy*, da basi di dati tradizionali e dal Web; il World Wide Web stesso, infine, può essere considerato un‘importante fonte di dati semistrutturati.

Il rapido imporsi XML come formato standard per lo scambio di dati ha attratto l‘attenzione dei ricercatori sulle basi di dati semistrutturate. Le profonde somiglianze che i documenti XML *data-oriented* presentano con i dati semistrutturati, e.g., autodescrittività dei

dati e flessibilità nella rappresentazione, hanno spinto, quindi, la comunità di ricerca ad adottare XML come formato standard per la rappresentazione di dati semistrutturati.

La convergenza dei due ambiti di ricerca ha avuto due importanti conseguenze: da un lato, è stato possibile applicare alle basi di dati semistrutturate quanto sviluppato dalla comunità XML (con particolare riferimento ai meccanismi di schematizzazione); dall'altro, si è resa possibile l'applicazione ai documenti XML dei risultati ottenuti per i dati semistrutturati (modelli dei dati, linguaggi di interrogazione, ecc...), colmando così alcune lacune della ricerca su XML.

Lo scopo di questo articolo consiste nell'esaminare le principali problematiche relative a basi di dati semistrutturate e documenti XML data-oriented, nonché descrivere le soluzioni sviluppate nelle comunità di ricerca interessate, con particolare riferimento ai modelli dei dati, ai linguaggi di interrogazione e ai meccanismi di schematizzazione logica.

2 Modelli dei Dati

La progettazione di una base di dati, dalla più semplice alla più complessa, impone, come passo preliminare, la scelta di un modello dei dati adeguato alla rappresentazione della realtà di interesse. Nella sua accezione più semplice, un modello dei dati può essere considerato un insieme di meccanismi di astrazione utilizzati per rappresentare le varie forme di conoscenza.

Esempio 2

Nel modello dei dati a oggetti i due meccanismi di astrazione principali sono l'aggregazione e la specializzazione: il primo meccanismo consente di modellare un'entità del mondo reale attraverso un'aggregazione di proprietà, mentre il secondo permette di organizzare gli oggetti in gerarchie.

In un'accezione più estesa un modello dei dati può essere visto come un insieme di meccanismi di astrazione e, laddove presenti, di meccanismi di tipizzazione.

Anche per le basi di dati semistrutturate, come per quelle tradizionali, il primo problema che deve essere affrontato è la scelta di un modello dei dati che sia in grado di trattare adeguatamente le caratteristiche peculiari dei dati in esame, i.e., l'irregolarità e/o l'instabilità strutturali. La letteratura sui dati semistrutturati propone molti modelli dei dati, che possono essere suddivisi in tre categorie.

Modelli dei dati lightweight I modelli dei dati cosiddetti *lightweight* rappresentano una base di dati attraverso un grafo diretto etichettato. Il più diffuso e noto modello lightweight è OEM [3]. In OEM una base di dati è una collezione non ordinata di oggetti, ognuno dotato di un proprio oid unico. Un oggetto può essere atomico o complesso: nel primo caso il suo valore è un valore di base, mentre nel secondo caso esso consiste di un insieme non ordinato di coppie (etichetta,oggetto), dove l'etichetta descrive il ruolo semantico giocato dal relativo *sotto-oggetto*.

In OEM, quindi, una base di dati viene rappresentata attraverso un grafo diretto, i cui nodi sono associati agli oggetti e i cui archi sono etichettati con gli attributi degli oggetti complessi. Nelle Figure 1 e 2 sono mostrati una semplice base di dati OEM e la sua rappresentazione a grafo.

```
BibRoot:&o1
        {article:&o2
                {author:&o21            Ghelli
                 title:&o22             Divergence of F< Type Checking
                 year:&o23              1995
                }
        },
        {techreport:&o3
                {author:&o31            Albano
                 author:&o21            Ghelli
                 author:&o32            Bergamini
                 author:&o33            Orsini
                 title:&o34             An Object Data Model with Roles
                 year:&o35              1993
                }
        },
        {techreport:&o4
                {author:&o41            Cardelli
                 title:&o42             Typeful Programming
                 year:&o43              1989
                }
        }
```

Figura 1: Una base di dati OEM

Come è possibile osservare dalle figure, OEM permette ad un oggetto di avere molteplici attributi con lo stessa etichetta, offrendo così la possibilità di rappresentare collezioni eterogenee.

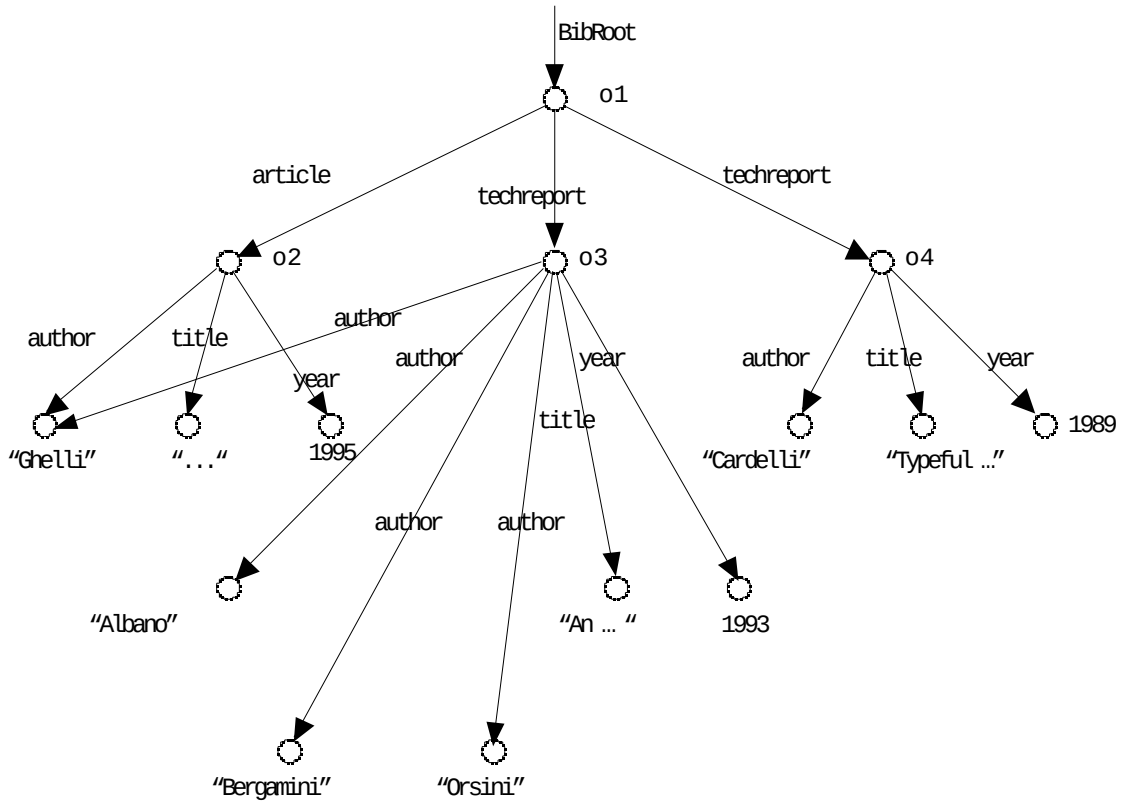

Figura 2: Rappresentazione a grafo di una base di dati OEM

Esistono numerose varianti di OEM (e.g., [4] e [5]), che si differenziano da OEM principalmente per l'etichettatura dei nodi e/o degli archi.

I modelli dei dati lightweight sono contraddistinti da un elevato grado di flessibilità nella modellazione dei dati, che consente di rappresentare agevolmente qualsiasi sorgente di dati. Essi, inoltre, permettono un'agevole integrazione di sorgenti di dati eterogenee, rivelandosi particolarmente adatti all'impiego in sistemi di integrazione (e.g., [6] e [7]).

Modelli dei dati heavyweight I modelli dei dati cosiddetti *heavyweight* sfruttano i meccanismi di astrazione del modello a oggetti e ne arricchiscono i meccanismi di tipizzazione con tipi unione e/o tipi ricorsivi.

Nel modello dei dati del linguaggio POQL [8] il modello ODMG viene esteso con tipi unione etichettati; questi tipi aumentano il livello di flessibilità di ODMG, consentendo di rappresentare più agevolmente le irregolarità strutturali dei dati semistrutturati.

Proposte successive impiegano, in luogo di tipi unione etichettati, tipi unione non etichettati, che, come mostrato in [9], si rivelano più idonei a rappresentare dati fortemente irregolari.

I modelli dei dati heavyweight, pur essendo molto più ricchi dei modelli a grafo, non si sono, tuttavia, rivelati particolarmente adatti alla rappresentazione di dati semistrutturati: il livello di flessibilità che essi offrono, infatti, spesso non è sufficiente a trattare dati fortemente irregolari e a consentire un'agevole integrazione di sorgenti eterogenee.

Modelli dei dati midweight I modelli dei dati cosiddetti *midweight* cercano di coniugare le caratteristiche di flessibilità dei modelli lightweight con la potenza dei meccanismi di astrazione dei modelli a oggetti. Ciò avviene arricchendo i meccanismi di tipizzazione del modello a oggetti con un particolare tipo Unstructured, il quale viene associato ad ogni valore non facilmente rappresentabile con i meccanismi tradizionali. Il tipo Unstructured non esprime vincoli sui valori che esso descrive; a differenza, tuttavia, di un tipo Any, esso indica a un motore di query, che sfrutti tale modello, che i dati da esso rappresentati sono semistrutturati e che, quindi, devono essere interrogati ed esplorati con le tecniche proprie dei sistemi basati su modelli lightweight.

Gli esempi più significativi di modelli midweight sono presenti in [10] e in [11]. Nella prima proposta un modello dei dati a oggetti con tipi unione e tipi ricorsivi viene esteso con un tipo spring, che svolge le funzione del citato tipo Unstructured; nella seconda proposta, il modello dei dati di O_2 viene arricchito con un tipo OEM, permettendo, così, di utilizzare questa variante di O_2 sia per gestire tradizionali basi di dati O_2 che dati semistrutturati.

Il modello dei dati OEM è ormai considerato lo standard de facto per la rappresentazione di dati semistrutturati e la maggior parte dei sistemi di gestione di SSD si basa su di esso. Ciò è stato in larga parte determinato dalle caratteristiche di semplicità del modello e di flessibilità nella rappresentazione; l'impiego di OEM, tuttavia, ha comportato alcune conseguenze negative, prima fra tutte, come vedremo in seguito, la mancanza di schemi.

Il modello dei dati OEM può essere facilmente esteso alla rappresentazione di documenti XML data-oriented. Un'estensione molto semplice è quella descritta in [12] e impiegata nel linguaggio di interrogazione XML-QL. In questo modello un documento XML è rappresentato attraverso un *grafo XML*, descritto nel modo seguente.

Definizione

Un grafo XML è un grafo diretto etichettato *G* nel quale:

- gli archi sono etichettati con i *tag* degli elementi;
- i nodi sono associati al contenuto degli elementi ed etichettati con coppie (attributo,valore); ogni nodo, inoltre, è dotato di un oid unico;
- le foglie sono etichettate con valori di base;
- un nodo particolare svolge la funzione di radice ed è associato al *document element* del documento corrispondente.

Gli attributi di tipo ID, IDREF e IDREFS vengono gestiti in modo particolare: gli attributi di tipo ID sono rappresentati come oid definiti esplicitamente, mentre attributi di tipo IDREF e IDREFS vengono rappresentati attraverso archi dagli elementi referenti ai nodi associati al contenuto degli elementi riferiti; tali archi sono etichettati con il nome dell'attributo IDREF o IDREFS.

Esempio 3

Consideriamo un documento XML relativo a sistemi operativi e produttori software, mostrato nella Figura 3. Esso può essere rappresentato dal grafo XML mostrato nella Figura 4.

```
<database>
        <softwareVendors>
                <company        ID="o001">
                        <name> Apple Computers Inc. </name>
                        <headquarters> Cupertino </headquarters>
                        ...
                </company>

                <company        ID="o777">
                        <name> Microsoft Inc. </name>
                        <headquarters> Redmond </headquarters>
                        <comment> Evil Empire </comment>
                        ...
                </company>

                ...
        </softwareVendors>

        <operatingSystems>
                <OS Y2KBugFree="yes" releasedBy="o001">
                        <name> MacOS </name>
                        <currentVersion         lastRevisionDate="03012000">
                                9.1
                        </currentVersion>
                        ...
                </OS>

                <OS Y2KBugFree="no" releasedBy="o777">
                        <name> WIndowsNT </name>
                        <currentVersion         lastRevisionDate="11011999">
                                4 SP6
                        </currentVersion>
                        ...
                </OS>

                ...
        </operatingSystems>
</database>
```

Figura 3: Un documento XML

Un‘altra interessante variante di OEM è il modello XML Infoset [13]. Il modello XML Infoset può essere considerato un modello lightweight con nodi etichettati; esso si differenzia dal precedente modello per lo spostamento dell‘etichettatura dagli archi ai nodi, per la maggior quantità di informazione che esso esprime e per l‘assenza di una nozione di oid.

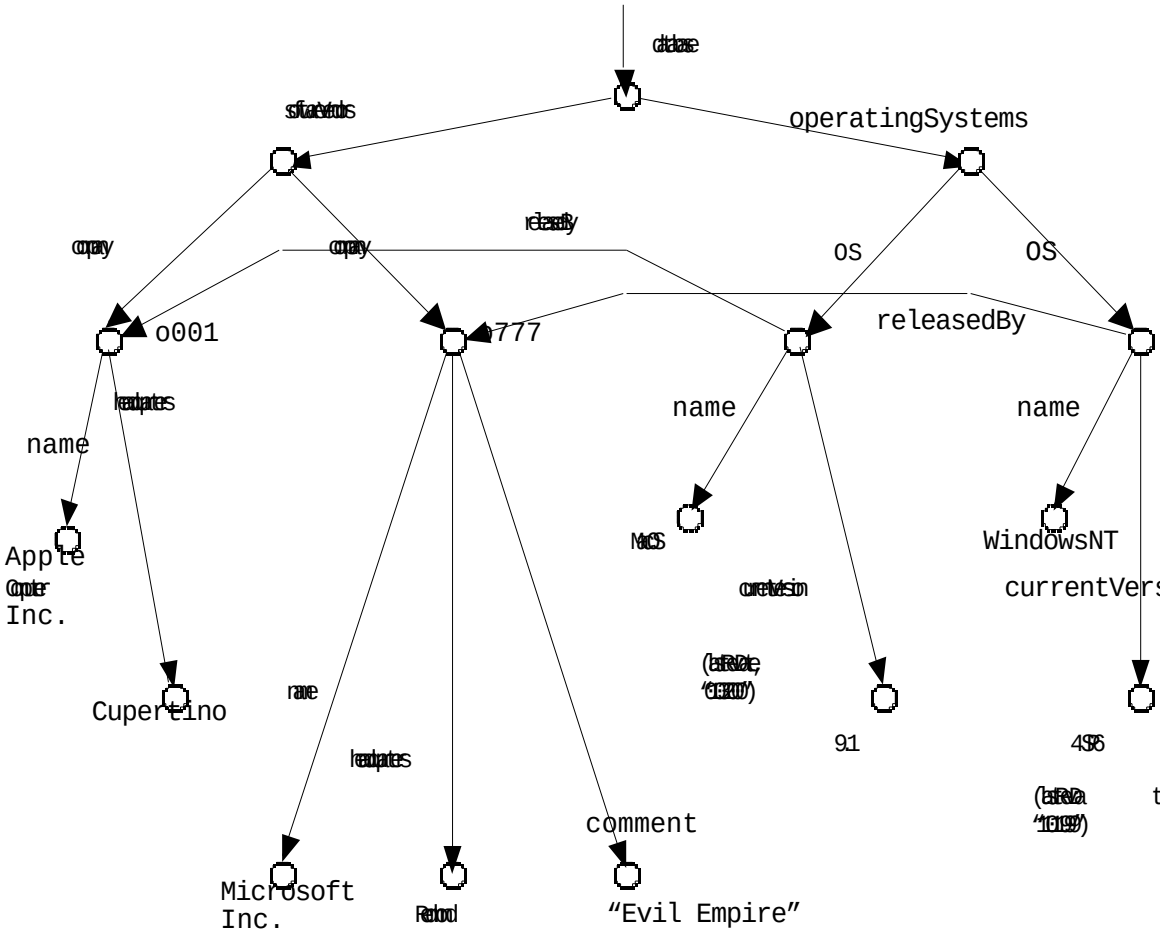

Figura 4: Un grafo XML

Sebbene il modello Infoset non risulti del tutto adeguato, per l‘eccessiva quantità di informazioni descritte, alla rappresentazione di documenti XML data-oriented, riteniamo che, come indicato anche in [14], una sua variante più astratta possa essere proficuamente impiegata anche per i documenti data-oriented.

3 Linguaggi di Interrogazione

Qualunque sia la base di dati in esame, un suo proficuo impiego richiede l’adozione di un idoneo linguaggio di interrogazione.

È possibile individuare alcune caratteristiche irrinunciabili che un linguaggio di interrogazione per basi di dati semistrutturate deve possedere. In primo luogo, il linguaggio

deve essere dichiarativo, deve, cioè, consentire all'utente di specificare quali dati vuole recuperare senza costringerlo a descrivere anche un piano di accesso ai dati stessi.

In secondo luogo, il linguaggio deve essere in grado di esprimere sia operazioni di *interrogazione* vere e proprie (quali, ad esempio, la selezione e la proiezione dell'algebra relazionale) sia operazioni di *trasformazione* sulla base di dati, che possano essere sfruttate, ad esempio, per l'integrazione di una molteplicità di sorgenti di dati.

In terzo luogo, le query devono essere *composizionali*: ciò significa che il risultato di una query deve essere esso stesso una base di dati, utilizzabile come input a query successive; questa caratteristica, unita alla capacità di esprimere trasformazioni, garantisce la possibilità di impiegare il linguaggio per la definizione di viste o di basi di dati integrate (secondo l'approccio *Global as View*), che possano essere ulteriormente interrogate dagli utenti.

Il linguaggio, infine, deve avere un potere espressivo paragonabile a quello di SQL, se non superiore.

Sono stati sviluppati molti query language per dati semistrutturati. Fra i linguaggi più interessanti possiamo segnalare StruQL, Lorel e YATL.

StruQL [15], realizzato nell'ambito del progetto STRUDEL, è un linguaggio dichiarativo basato sul modello OEM. Esso è in grado di esprimere sia operazioni di interrogazione che di trasformazione di un grafo OEM; il risultato di una query, inoltre, è sempre un grafo OEM e, quindi, il criterio di composizionalità risulta soddisfatto (si veda la Figura 5 per un esempio di query).

Il potere espressivo di StruQL è molto elevato ed è dimostratamente equivalente a *FO+TC*: è possibile, cioè, esprimere le query della logica del primo ordine e calcolare la chiusura transitiva di una relazione binaria.

StruQL, infine, è contraddistinto da una sintassi e una semantica relativamente semplici, il che non è frequente nell'ambito dei linguaggi per dati semistrutturati.

```
WHERE OS{x}, Company{y},
        x --> "Y2KBugFree" --> "Yes",
        x --> "releasedBy" --> y,
        y --> "name" --> n
COLLECT
        Result(),
        Result() --> "name" --> n
```

Figura 5: Una query StruQL

Lorel [16], sviluppato nell'ambito del sistema LORE, è un query language dichiarativo per l'interrogazione di dati semistrutturati e documenti XML. Come accade per StruQL, anche Lorel è in grado di esprimere trasformazioni e soddisfa il criterio di composizionalità.

Il potere espressivo del linguaggio è paragonabile a quello di StruQL, sebbene non sia possibile calcolare la chiusura transitiva di una relazione binaria.

A livello sintattico, Lorel estende la sintassi di SQL e OQL, come è possibile constatare dalla Figura 6; per quanto riguarda la semantica, il linguaggio è contraddistinto da una semantica estremamente complessa e spesso ad hoc, necessaria per gestire le operazioni di *type coercion* proprie di Lorel.

```
select xml(result: {name : n})
from database.softwareVendors x, x.company y, y.name n
        database.operatingSystems z,
        z.OS u, u.releasedBy y
where u.Y2KBugFree="Yes"
```

Figura 6: Una query Lorel

Il linguaggio YATL [17] è stato sviluppato all'interno del sistema di integrazione YAT [18]. Originariamente impiegato unicamente per la definizione delle operazioni di integrazione e conversione di dati, YATL è stato recentemente esteso in modo da permetterne lo sfruttamento anche come query language.

YATL si basa su un modello dei dati ad albero, descritto in [17], non perfettamente adattabile alla classificazione introdotta nella Sezione 2.

YATL è un linguaggio dichiarativo, capace di esprimere trasformazioni e soddisfacente il criterio di composizionalità. Il suo potere espressivo è leggermente inferiore a quello di StruQL e Lorel, non essendo possibile formulare, per precise decisioni di progetto, query con *regular path expression*.

Le espressioni regolari di cammino costituiscono un elemento caratterizzante la maggioranza dei linguaggi di interrogazione per dati semistrutturati. Nella sua forma più semplice un'espressione regolare di cammino è una sequenza di etichette $l_1.l_2. \ldots .l_n$; la sua valutazione consiste nel percorrere, a partire dalla radice del grafo, ogni cammino le cui etichette concidano con $l_1, l_2, \ldots, l_n$ e nel restituire l'insieme di nodi raggiungibile attraverso tale cammino. Se consideriamo, ad esempio, la semplice basi di dati introdotta nella Figura 3, la valutazione dell'espressione database.softwareVendors.company restituisce due nodi (o, meglio, oid) corrispondenti a Apple e Microsoft.

A partire da questa forma estremamente semplice, le espressioni di cammino possono venire arricchite attraverso l'uso di *wild card*, quali _ e *, che consentono di descrivere, rispettivamente, qualsiasi etichetta e una sequenza di zero o più occorrenze di una stessa etichetta. La grammatica completa delle espressioni regolari di cammino è riportata nella Figura 7.

```
e ::= l |   ε | _ | e '.' e | '(' e ')' | e '|' e | e '*' | e '+'  | e'\'
```

Figura 7: Grammatica delle espressioni regolari di cammino

Per quanto riguarda i linguaggi di interrogazione per documenti XML data-oriented, rimangono validi i desiderata espressi per i linguaggi per dati semistrutturati; ad essi, tuttavia, deve essere aggiunto un quinto requisito, che consiste nel richiedere che il motore di query sia in grado di sfruttare, se presenti, le informazioni strutturali contenute nell'eventuale DTD associata ad ogni sorgente XML; come vedremo nella sezione successiva, la conoscenza di tali informazioni può consentire una più efficiente valutazione delle query.

L'interrogazione di documenti XML data-oriented può essere effettuata attraverso query language per SSD opportunamente estesi, quali Lorel e YATL, oppure mediante linguaggi pensati espressamente per XML. Tra questi ultimi i linguaggi più interessanti sono sicuramente XML-QL e XQL.

XML-QL [12] è stato sviluppato dal medesimo gruppo di ricerca che ha prodotto StruQL. I due linguaggi sono fortemente connessi tra loro: XML-QL, in effetti, può essere considerato un'evoluzione e un adattamento di StruQL a XML.

XML-QL si basa sull'estensione del modello dei dati OEM descritta nella precedente sezione; esso è in grado di esprimere sia operazioni di query che operazioni di trasformazione; il risultato di un'interrogazione, inoltre, è sempre un documento XML e, di conseguenza, viene soddisfatto il criterio di composizionalità.

Nella Figura 8 è mostrata una semplice query XML-QL sul database sistemi/produttori presentato precedentemente. Questa query restituisce un documento XML contenente il nome di ogni compagnia che produce almeno un sistema operativo non soggetto al *millennium bug*.

```
WHERE <database>
            <operatingSystems>
                  <OS Y2KBugFree="Yes" releasedBy=$i> $o </>
            </>,
            <softwareVendors>
                  <company ID=$i>
                        <name> $n </>
                  </>
            </>
      </> IN "www.a.b.c/opsys.xml"
CONSTRUCT
      <result>
            <name> $n </>
      </>
```

Figura 8: Una query XML-QL

Il potere espressivo di XML-QL è molto elevato e paragonabile a quello di StruQL (si può dimostrare che esso è pari a *FO+TC*).

Attualmente il motore di query di XML-QL non è in grado di sfruttare le eventuali informazioni strutturali contenute nelle DTD; gli autori di XML-QL, comunque, hanno in programma la modifica del *query engine* in modo che esso sia in grado di utilizzare tali informazioni ai fini dell'ottimizzazione.

XML-QL, infine, è contraddistinto da una sintassi e una semantica estremamente chiare e semplici, che ne fanno, a nostro giudizio, il miglior query language per documenti XML data-oriented.

XQL [19] è un linguaggio di interrogazione dichiarativo per documenti XML *general-purpose*. Esso estende il linguaggio di *pattern* di XSL [20], sfruttando i meccanismi di ricorsione strutturale (l'operatore //) già impiegati in altri linguaggi (UnQL [4], ad esempio).

XQL è in grado di esprimere limitate trasformazioni su grafi, non essendo possibile effettuare il *flattening* dei documenti, condizione indispensabile per poter eseguire ristrutturazioni *arbitrarie* della struttura dei grafi stessi (si veda [21]).

XQL soddisfa il criterio di composizionalità, offrendo un interessante, sebbene confuso, meccanismo di *serializzazione* dei risultati di una query.

Il potere espressivo di XQL risulta inferiore a quello di XML-QL, non essendo presenti nel linguaggio *variabili di tag*, indispensabili per l'interrogazione di documenti la cui struttura non sia completamente nota. Inoltre, sebbene sia possibile formulare query di giunzione, come mostrato nella Figura 9, la semantica stessa della giunzione non è ben definita, non essendo del tutto chiaro se tale operazione consista in una vera giunzione o, piuttosto, in una *semigiunzione*.

XQL, in conclusione, non risulta del tutto adeguato per l'interrogazione di documenti XML data-oriented. Cionondimeno, occorre osservare che alcune caratteristiche di XQL, quali la possibilità di formulare query compatte, inseribili in URL, e la capacità di esprimere condizioni sui testi e di accedere al testo contenuto in elementi fra loro vicini attraverso il metodo text(), appaiono molto interessanti ai fini dell'interrogazione di documenti XML general-purpose.

```
\database\softwareVendors/company[$i=ID]/name
        [/database/operatingSystems/OS[@releasedBy=$i][@Y2KBugFree="Yes"]]
```

Figura 9: Una query XQL

Per concludere la trattazione sui linguaggi di interrogazione, occorre osservare che, sebbene molto lavoro sia stato fatto, rimangono ancora aperti alcuni problemi significativi, legati all'assenza di schemi: in primo luogo, nessuno dei linguaggi finora proposti ha una nozione di correttezza di query che non sia la mera correttezza sintattica; in secondo luogo, l'ottimizzazione di query contenenti espressioni regolari e, in particolare, operatori di chiusura, rimane insoddisfacente. Questi due aspetti saranno esaminati con maggior dettaglio nella sezione successiva.

4 Meccanismi di Schematizzazione

Nelle precedenti sezioni abbiamo osservato come i sistemi per la gestione di dati semistrutturati siano di soliti privi di uno schema logico. Lo schema logico di una base di dati, ricordiamo, descrive l'organizzazione logica dei dati.

L'assenza di schema logico trova origine fondamentalmente nell'adozione di modelli dei dati lightweight; questi modelli dei dati presentano un elevato grado di flessibilità nella modellazione di dati semistrutturati e nell'integrazione di sorgenti eterogenee, che mal si conciliano con la rigidità imposta da uno schema.

Naturalmente, l'assenza di schemi logici ha alcune conseguenze negative. In primo luogo, un'efficiente valutazione di query con espressioni regolari di cammino e, in particolare, con operatori di chiusura non è possibile in mancanza di schemi. Sebbene gli attuali sistemi di gestione di dati semistrutturati presentino ottimizzatori *cost-based* simili a quelli dei sistemi relazionali o a oggetti, la valutazione di *path expression* avviene, nella maggioranza dei casi, attraverso un approccio *naive*, che consiste nel costruire l'automa equivalente e nel visitare conseguentemente il grafo; un tale approccio può richiedere, nel caso di query come quella mostrata nella Figura 10, l'esplorazione dell'intera base di dati (il che, ovviamente, è troppo costoso).

```
WHERE <*>
                <comment> $c </>
        </> IN "www.a.b.c/opsys"
CONSTRUCT <comment>
                $c
            </>
```

Figura 10: Una query con operatori di chiusura

In secondo luogo, l'assenza di uno schema non ha permesso la definizione di una nozione di correttezza di query. Nei sistemi relazionali o a oggetti, la macchina logica verifica la correttezza di ogni query sullo schema, accertandosi che i nomi di relazioni, classi, attributi e campi siano utilizzati in modo corretto; essendo lo schema molto più piccolo dei dati, questa verifica viene effettuata molto efficientemente.

Nei sistemi per SSD, invece, non essendo presente uno schema, non è possibile accertare la correttezza delle query rispetto ad esso. La soluzione alternativa di verificare la correttezza direttamente sui dati non è praticabile, poiché una qualunque modifica ai dati potrebbe inficiare la *soundness* di una query ritenuta precedentemente corretta (il che potrebbe avere antipatiche conseguenze nel caso di una query incapsulata in un'applicazione).

In terzo luogo, l'assenza di uno schema rende estremamente difficoltosa la formulazione stessa delle interrogazioni: mentre gli utenti "umani" possono comunque accedere alla base di dati attraverso un processo di *navigazione*, le applicazioni si vedono preclusa qualsiasi forma di interazione con la base di dati, non essendo possibile per il programmatore incapsulare in esse le opportune query.

La mancanza di uno schema logico, infine, può avere pesanti conseguenze negative sull'efficienza di memorizzazione dei dati su memoria permanente; senza la conoscenza dello schema logico o di informazioni strutturali sui dati, infatti, non è possibile individuare adeguate strategie di memorizzazione, che offrano buone proprietà di *clustering*.

Esistono numerose proposte di meccanismi per la descrizione della struttura dei dati semistrutturati. I meccanismi proposti hanno caratteristiche peculiari, che li distinguono da quelli impiegati per basi di dati strutturate. Innanzitutto, mentre nei sistemi tradizionali lo schema logico viene costruito prima di popolare la base di dati, nei sistemi per SSD lo schema viene di solito inferito a partire da basi di dati già esistenti. In secondo luogo, mentre gli schemi logici per sistemi tradizionali sono relativamente stabili nel tempo, gli schemi per SSD sono suscettibili di frequenti aggiornamenti, per rispecchiare i cambiamenti strutturali nei dati.

Fra le proposte di meccanismi di schematizzazione per SSD quelle che rivestono il maggior interesse sono i Graph Schema [22] e i DataGuide [23].

Un Graph Schema è un grafo diretto etichettato i cui archi vengono etichettati con formule unarie di una teoria decidibile del primo ordine (si veda la Figura 11 per un Graph Schema che descrive la basi di dati sistemi/produttori). La relazione di conformità tra un base di dati e uno schema è calcolabile in tempo $O(mnt)$ (dove m e n sono il numero complessivo di archi e

nodi dei due grafi e *t* è il tempo necessario a verificare la validità di una formula unaria della teoria) attraverso un algoritmo di simulazione. Esiste, inoltre, una nozione di sussunzione fra schemi, simile ad una relazione di *subtyping*, definita in termini di simulazione e calcolabile anch'essa in PTIME.

I Graph Schema possono essere impiegati per l'ottimizzazione di query, adottando le tecniche di riscrittura di espressioni regolari di cammino descritte in [24] e [25].

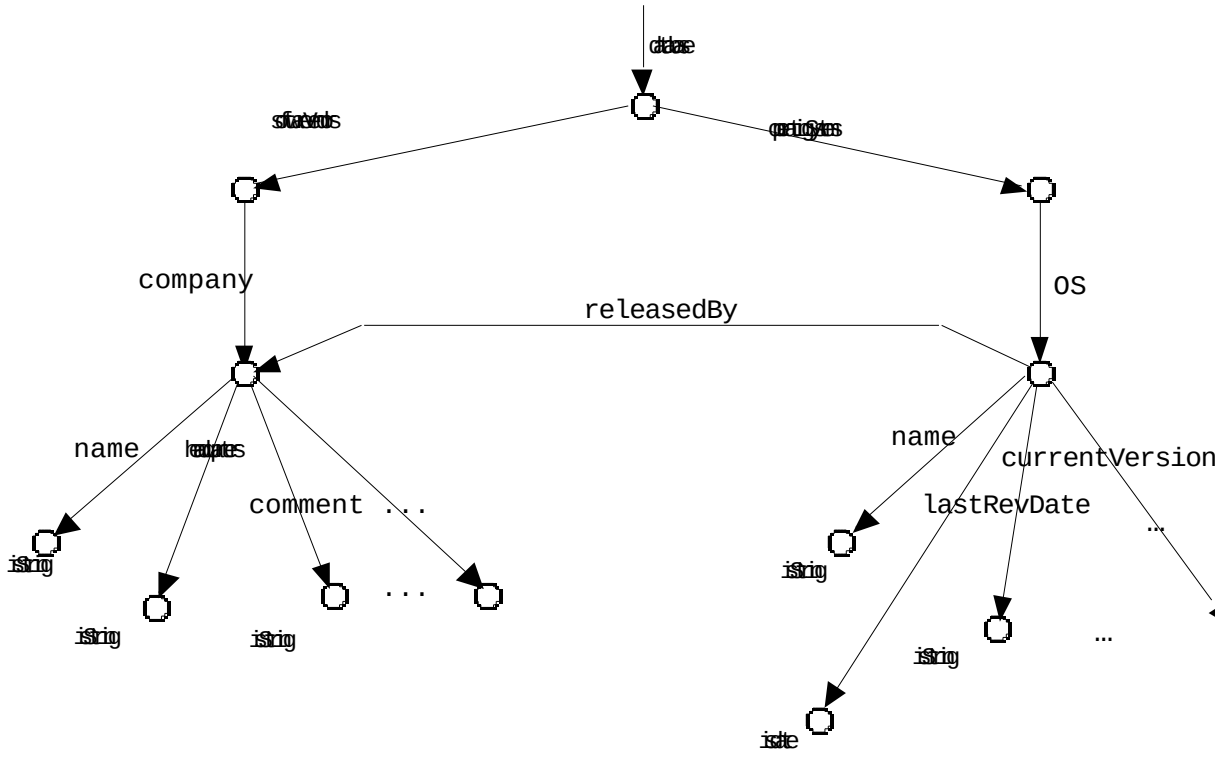

Figura 11: Un Graph Schema

Un DataGuide [23] [26] è un grafo OEM i cui archi sono etichettati con costanti e ai cui nodi sono associate informazioni statistiche relative alla base di dati. I DataGuide sono utilizzati all'interno del sistema LORE; essi vengono automaticamente inferiti e aggiornati dal sistema, senza richiedere intervento alcuno da parte del progettista.

I DataGuide vengono proficuamente impiegati nell'ottimizzazione di espressioni regolari di cammino [27] e nella costruzione di un'interfaccia grafica per la navigazione nel database.

I meccanismi di schematizzazione per dati semistrutturati possono essere agevolmente adattati alla rappresentazione di documenti XML data-oriented, a patto che venga impiegato un idoneo modello dei dati.

La schematizzazione di documenti data-oriented può essere effettuata anche attraverso meccanismi specifici per documenti XML. Lo standard XML, innanzi tutto, definisce una forma di "schema" per i documenti XML, quali sono le DTD. Le DTD non risultano adeguate alla rappresentazione di basi di dati per tre ordini di motivi: in primo luogo, esse descrivono la struttura sintattica dei documenti piuttosto che l'organizzazione logica dei dati in essi contenuti; in secondo luogo, esse impongono un ordinamento sui dati presenti nei documenti, il che può creare seri problemi in fase di ottimizzazione delle query; le DTD, infine, non tipizzano i riferimenti né impongono su essi alcun tipo di vincolo.

Oltre alle DTD esistono numerose proposte di schemi per documenti XML. Per quanto concerne il nostro specifico campo di applicazione, la proposta che ci apparepiù interessante è SOX2 [28]. SOX2 è stato ideato da CommerceOne per i propri strumenti di commercio elettronico; uno schema SOX descrive un documento attraverso un insieme di definizioni di *element type* e di *attribute type*. Gli schemi SOX si differenziano dalle DTD essenzialmente per la tipizzazione dei riferimenti e per un maggiore attenzione all'organizzazione logica dei dati. Nella Figura 12 è presentato uno schema SOX per il nostro database sistemi/produttori.

L'adozione delle DTD o degli schemi SOX per la schematizzazione di documenti data-oriented impone di affrontare alcune problematiche. In primo luogo, occorre adattare le tecniche di ottimizzazione di espressioni regolari di cammino sviluppate per altri formalismi: se la tecnica di *query pruning* appare facilmente adattabile, le tecniche più complesse di riscrittura di *path expression* a tempo di compilazione potrebbero creare alcune difficoltà. In secondo luogo, appare necessario sviluppare idonee strategie di gestione e aggiornamento di tali schemi, che consentano di adeguare gli schemi stessi alle modifiche strutturali della base di dati, senza nel contempo richiedere il blocco del sistema.

È nostra opinione che la soluzione di queste problematiche richieda la trasposizione di questi meccanismi di schematizzazione in formalismi quali la teoria dei tipi o teoria dei linguaggi formali, secondo lo spirito di alcuni progetti di ricerca già esistenti [29] [30].

5 Conclusioni

Molti problemi rimangono aperti nella ricerca su basi di dati semistrutturate e documenti XML data oriented. Il più importante di essi è certamente la definizione di idonei meccanismi di schematizzazione logica dei dati, attraverso i quali poter dare soluzione ai problemi di ottimizzazione e di controllo di correttezza delle query. Un altro interessante problema aperto riguarda l'individuazione di adeguate strategie di memorizzazione dei dati su memoria permanente: questo tema di ricerca ci appare, comunque, strettamente connesso con quello dei meccanismi di schematizzazione.

Per quanto concerne la modellazione dei dati, le soluzioni presenti in letteratura sono ritenute unanimamente soddisfacenti, così come vi è un ampio consenso sulle caratteristiche dei linguaggi di interrogazione (prova ne è il fatto che i principali linguaggi finora proposti, con l'eccezione di XQL, sono abbastanza simili fra loro).

La ricerca attuale si sta indirizzando verso varie problematiche. Alcuni gruppi di ricerca, quali il Verso Database Research Group di INRIA, stanno affrontando il problema della memorizzazione dei dati (progetto Xyleme); altri gruppi si stanno dedicando all'estensione e il miglioramento di sistemi prototipali esistenti, con l'obiettivo, ad esempio, di sfruttare informazioni strutturali ai fini dell'ottimizzazione.

Il Gruppo di Ricerca sulle Basi di Dati del Dipartimento di Informatica dell'Università di Pisa, infine, sta cercando di definire un linguaggio di interrogazione tipizzato, dotato di una nozione di correttezza di query e di buone proprietà di ottimizzazione.

```
<schema uri="urn:opsys:sampleSchema" />
   <elementtype name="database" >
      <model> <sequence>

            <elementtype name="softwareVendors">
               <model> <sequence occurs="*">

                  <elementtype name="company">
                     <attdef name="ID" datatype="int" />
                     <model> <sequence>

                        <elementtype name="name">
                           <model> <string/> </model>
                        </elementtype>

                        <elementtype name="comment">
                           <model occurs="?"> <string/> </model>
                        </elementtype>

                     </sequence> </model>
                  </elementtype>

               </sequence> </model>
            </elementtype>

            <elementtype name="operatingSystems"/>
               <model> <sequence occurs="*">

                  <elementtype name="OS">
                     <attdef name="Y2KBugFree" datatype="string"/>
                     <model>
                      <elementtype name="name">
                       <model> <string/> </model>
                      </elementtype>

                      <elementtype name="currentVersion">
                      <attdef name="lastRevisionDate" datatype="date"/>
                       <model> <string/> </model>
                      </elementtype>

                      <elementtype name="releasedBy" type="company"/>
                     </model>
                  </elementtype>

               </sequence> </model>
            </elementtype>

      </sequence> </model>
   </elementtype>
</schema>
```

Figura 12: Uno schema SOX2

Riferimenti

1. Buneman, P. *Semistructured Data*. In *Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 1997. Tucson, Arizona: ACM Press.

2. Stein, L.D. and J. Thierry-Mieg**,** *AceDB: A Genome Database Management System.* Computing in Science and Engineering, 1999. **1**(3).

3. Papakonstantinou, Y., J. Widom, and H.G. Molina**,** *Object Exchange Across Heterogeneous Information Sources.* Proceedings of IEEE Int. Conference on Data Engineering, Birmingham, England, 1996 .

4. Buneman, P., Davidson, S., and Suciu, D. *Programming Constructs for Unstructured Data*. in *Proceedings of 5th International Workshop on Database Programming Languages*. 1995. Gubbio, Italy.

5. Fernandez, M., *et al., Catching the Boat with Strudel: Experiences with a Web-Site Management System.* SIGMOD Record (ACM Special Interest Group on Management of Data), 1998. **27**(2): p. 414--??

6. Chawathe, S., *et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources*. In *16th Meeting of the Information Processing Society of Japan.* 1994. Tokyo, Japan.

7. Ives, Z.G., *et al. An Adaptive Query Execution System for Data Integration*. In *ACM SIGMOD Conference on Management of Data*. 1999. Philadelphia, PA.

8. Christophides, V., *et al. From Structured Documents to Novel Query Facilities*. In *1994 ACM SIGMOD International Conference on Management of Data*. 1994. Minneapolis, Minnesota: ACM Press.

9. Buneman, P. and Pierce, B., *Union Types for Semistructured Data*, 1999, Dept. of CIS, University of Pennsylvania.

10. Bertino, E., *et al. An Approach to Classify Semi-Structured Objects*. In *Thirteenth European Conference on Object-Oriented Programming.* 1999. Lisboa (Portugal): Lecture Notes on Computer Science.

11. Abiteboul, S., *et al., Ozone: Integrating Structured and Semistructured Data*, 1998, Stanford University Database Group.

12. Deutsch, A., *et al.*, *XML-QL: A Query Language for XML*, 1998, World Wide Web Consortium.

13. Cowan, J. and D. Megginson, *XML Information Set*, 1999, World Wide Web Consortium.

14. Fankhauser, P., M. Marchiori, and J. Robie, *XML Query Requirements*, 2000, World Wide Web Consortium.

15. Fernandez, M., *et al.*, *A Query Language for a Web-Site Management System.* SIGMOD Record (ACM Special Interest Group on Management of Data), 1997. **26**(3): p. 4--??

16. Abiteboul, S., *et al.*, *The Lorel Query Language for Semistuctured Data.* Journal of Digital Libraries, 1(1), 1997: p. 68-88.

17. Cluet, S. and J. Siméon, *Data Integration based on data conversion and restructuring*, 1997, INRIA, Verso database group.

18. Christophides, V., S. Cluet, and J. Siméon. *Semistructured and Structured Integration Reconciled*, 1998.

19. Robie, J., *XQL (XML Query Language)*, August 1999.

20. Adler, S., *et al.*, *Extensible Stylesheet Language (XSL) Version 1.0*, 2000.

21. Fernandez, M., J. Siméon, and P. Wadler, *XML Query Languages: Experiences and Exemplars*, 1999.

22. Buneman, P., *et al.*, *Adding structure to unstructured data.* Lecture Notes in Computer Science, 1997. **1186**: p. 336--??

23. Goldman, R. and J. Widom. *DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases.* In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece.* 1997: Morgan Kaufmann.

24. Fernandez, M. and D. Suciu. *Optimizing Regular Path Expressions Using Graph Schemas*. In *Fourteenth International Conference on Data Engineering*. 1998. Orlando, Florida, USA: IEEE Computer Society.

25. Fernandez, M. and D. Suciu, *Optimizing regular path expressions using graph schemas (full version)*, 1997, AT&T Research Labs.

26. Goldman, R. and J. Widom. *Approximate DataGuides*. In *Proceedings of the second International Workshop WebDB '99, Pennsylvania*. 1999.

27. McHugh, J. and J. Widom, *Compile-Time Path Expansion in Lore*, 1998, Stanford University Database Group.

28. Davidson, A., *et al.*, *Schema for Object-Oriented XML 2.0*, 1999, World Wide Web Consortium.

29. Hosoya, H. and B.C. Pierce, *XDuce: A Typed XML Processing Language*, 2000, Department of CIS - University of Pennsylvania.

30. Milo, T., D. Suciu, and V. Vianu, *Typechecking for XML Transformers.*