

Una interfaccia generalizzata per l'interrogazione di documenti XML

Oreste Signore – Marco Andreini – Cristian Lucchesi – Silvia Martelli

Ufficio Italiano W3C presso il C.N.R.
Istituto di Scienza e Tecnologie dell'informazione "A. Faedo" (ISTI)
Area della Ricerca di Pisa San Cataldo - Via G. Moruzzi, 1 - 56124 Pisa
Email: oreste@w3.org

Abstract. La diffusione di documenti XML rende necessario poter disporre di strumenti per interrogare collezioni di documenti strutturati. La semantica dei documenti è sia nel testo che nella struttura del documento. In questo lavoro¹ viene presentata un' interfaccia di interrogazione verso collezioni di documenti XML. L' interfaccia si configura automaticamente in base alle caratteristiche dell' utente e alla struttura del documento, rappresentata dal suo XML Schema., che viene annotato esternamente mediante RDF, per consentire la formulazione di query semanticamente corrette.

Le esigenze degli utenti

Il numero di collezioni di documenti XML va crescendo, e la possibilità di interrogarle in modo efficace diventa un obiettivo importante. Considerato che un documento è costituito essenzialmente da tre componenti: *contenuto*, *struttura* e *presentazione*, e che XML ha favorito la separazione degli aspetti di presentazione dagli altri, va tenuto presente che l' utente, pur attribuendo una significativa importanza alla presentazione del documento, è comunque particolarmente interessato a ritrovare i documenti pertinenti ai suoi interessi, e quindi a formulare query precise e significative.

La ricerca per contenuto è sempre stata oggetto di attenzione, e tutti i sistemi di Information Retrieval, così come molti motori di ricerca, risolvono la query ricercando la presenza nel documento di specifiche parole. Tuttavia, sono noti alcuni prolemi intrinseci in questo approccio. La principale controindicazione è dovuta al fatto che ricercare una parola non significa necessariamente ricercare uno specifico concetto. Infatti, le parole sono semanticamente povere, e sono molti i casi di sinonimie e omografie, che hanno un impatto negativo su *precisione* e *richiamo*. Inoltre, spesso alcune parole hanno un significato diverso a seconda del contesto o parte del documento in cui appaiono. A puro titolo di esempio, il nome di una persona ha un significato diverso se appare nella sezione autore o nella sezione bibliografia o ringraziamenti. Da qui l' importanza di poter formulare le query indicando le specifiche sezioni del documento in cui ricercare i termini.

Comunque, per poter formulare query efficaci, individuando con esattezza i concetti da ricercare, è assolutamente necessario che utente e indicizzatore possano *condividere la*

¹ Questo lavoro è stato finanziato dal progetto QUESTION-HOW (Quality Engineering Solutions via Tools, Information and Outreach for the New Highly-enriched Offerings from W3C: Evolving the Web in Europe), contratto IST-2000-28767

stessa base di conoscenza. Questo obiettivo viene normalmente conseguito accedendo a dizionari o thesauri, in modo che l'utente possa verificare l'esattezza dei termini utilizzati e rendersi conto delle relazioni semantiche tra di essi.

Spesso, è disponibile una descrizione della struttura del documento, ma quasi sempre la semantica del campo è affidata alla comprensione del suo nome, e questo è inaccettabile nel contesto del World Wide Web, in cui sono presenti lingue e culture diverse. Inoltre, in presenza di un documento con strutturazione complessa, parte della semantica viene anche dalla struttura, per cui conoscerla e comprenderla diventa essenziale. Giusto a titolo di esempio, si consideri una collezione di documenti che descrivono documenti scientifici. In una struttura del tipo:

```
<paper>
  <title>Metadata and Semantic Web</title>
  <author>author1</author>
  <author>author2</author>
  <author>author3</author>
</paper>
```

il tag `<author>` contiene l'informazione che `author1`, `author2` e `author3` sono gli autori dell'articolo. Tuttavia, il loro ordine può essere il vettore di un'informazione addizionale, cioè che `author1` è il primo autore. L'utente potrebbe essere interessato a ricercare gli articoli scritti da `author1 AND author2`, in cui `author1` è il primo autore, e gli autori sono non più di tre.

È forse inutile sottolineare che il tag `<author>` (ma potrebbe anche essere il tag `<a>`, o `<au>` o `<x>`) non ha, di per sé, nessun particolare significato, mentre si assume implicitamente che esso contenga il nome dell'autore dell'articolo, e che l'utente ne comprenda il significato qualunque sia la sua lingua madre. Per una ricerca realmente efficace, l'utente dovrebbe comprendere appieno la semantica del tag, per esempio sapendo che essa è la stessa del tag `<dc:creator>` secondo la definizione della Dublin Core initiative ([DC]). In definitiva, l'utente ha bisogno di una *descrizione semantica di ogni tag* e di comprendere chiaramente la *struttura dei documenti* della collezione da ricercare.

Nel contesto del World Wide Web, accade di frequente che si effettui una ricerca su più database o collezioni di documenti. È ovvio che collezioni diverse di documenti, anche se dello stesso tipo o molto simili, avranno strutture e nomi dei tag diversi. L'identificazione delle equivalenze semantiche in collezioni eterogenee, e quindi la possibilità di ricercare e collegare documenti presenti in queste collezioni, è un obiettivo rilevante nel panorama del Semantic Web ([SemWeb]).

Obiettivi

L'obiettivo essenziale perseguito dall'interfaccia è supportare l'utente nella formulazione di query in cui vengono espresse condizioni sui valori assunti dai singoli elementi tenendo conto anche della *struttura* del documento. L'interfaccia deve consentire di formulare query "*semanticamente corrette*", quindi valide sia per quanto concerne i valori specificati, che per quanto concerne le condizioni imposte sulla struttura.

Per raggiungere questo scopo è necessario che l'utente sia posto nella condizione di:

- comprendere la *semantica degli elementi*;

- immettere i *dati corretti* per formulare le condizioni:
- identificare i *concetti* da ricercare.

Per quanto riguarda il primo aspetto, va ricordato che la semantica del singolo elemento non può essere affidata unicamente al nome del tag, spesso criptico o poco significativo. Nella specifica dello XML schema è possibile utilizzare un campo descrittivo, che però non sempre viene riempito dal progettista.

Per immettere i dati corretti nella specifica delle condizioni elementari è necessario avere conoscenza delle liste di *valori ammissibili* e degli eventuali *vincoli* esistenti.

Liste di valori ammissibili sono specificabili a livello di XML Schema, anche se appare noioso, ridondante e poco flessibile a fronte di possibili variazioni della lista di valori ammessi. Inoltre, non è possibile specificare vincoli nel caso in cui i valori ammissibili per un certo elemento dipendono dal valore assunto da un altro elemento. Per esempio, in una collezione di documenti eterogenei descritti da un unico schema, come nel caso di libri e atti di conferenze, l'elemento contenente il *luogo della conferenza* non può assumere un valore se il tipo di documento è *libro*, e non *atti di conferenza*.

Infine, per alcuni elementi semanticamente pregnanti, come le parole chiave, si rende necessario poter identificare il concetto da ricercare. Sotto questo aspetto, il problema fondamentale è la condivisione della stessa base di conoscenza tra l'indicizzatore e l'utente, e quindi la necessità di poter accedere al thesaurus, o più in generale all'ontologia alla quale fa riferimento lo specifico elemento. Questa esigenza è presente anche nel caso in cui si voglia ricercare in un elemento a testo libero, per il quale potrebbe essere comunque utile indicare uno o più thesauri di riferimento.

Nella formulazione della query, l'utente deve essere supportato anche per aspetti più strettamente legati alle *modalità di interazione*.

Primo fra tutti è l'aspetto del *multilinguismo*, per cui tutti gli elementi descrittivi devono poter essere forniti nella lingua preferita dall'utente. Un aspetto non trascurabile, nel contesto aperto implicitamente sottinteso dal web, è la possibilità di adattarsi alle *diverse culture* dei potenziali utenti. A titolo di esempio, esistono tradizioni diverse per esprimere le date, e non sempre riesce naturale utilizzare il formato adottato nella specifica collezione.

Un altro elemento distintivo è il livello di competenza dell'utente. Un utente che accede regolarmente ad una collezione, ne conosce bene l'organizzazione e il significato degli elementi, per cui potrebbe preferire una interfaccia meno verbosa, soprattutto nel caso in cui acceda al servizio tramite un dispositivo lento.

È ben noto che sarebbe possibile interrogare la collezione utilizzando delle opportune *espressioni XPath*, ma è anche evidente che non si tratta di una sintassi particolarmente adatta ad un utente finale.

Una esigenza molto sentita è quella di elaborare i documenti restituiti, sia per la tradizionale enfattizzazione dei termini ricercati o per la formattazione del documento, che per arricchire il testo con hyperlink, annotazioni, elementi aggiuntivi derivanti da una analisi del documento.

L'architettura

Dal punto di vista meramente architettureale, due condizioni sono state ritenute particolarmente stringenti e significative:

- *non modificare* la collezione di documenti e il suo schema

- mantenere la coerenza con i principi informatori del web, individuando quindi una architettura *aperta e distribuita*.

Possiamo distinguere tre livelli:

- *amministratore* della collezione,
- motore di ricerca e *document server*,
- *utente*.

L' amministratore della collezione ha il compito di *gestire* la collezione di documenti, utilizzando il motore di indicizzazione che preferisce, *definire*, ove questo non esistesse, lo XMLSchema della collezione, *annotare esternamente lo schema*, utilizzando RDF. I metadati descritti nell' annotazione RDF possono essere importati nel sistema, e memorizzati nei suoi oggetti interni. Una componente del sistema permette di aggiungere i metadati agli oggetti del sistema, che li può quindi esportare sotto forma di annotazione RDF. In questo modo si può creare l' annotazione RDF senza dover utilizzare un editor ad hoc.

L' utente può *navigare sulla struttura* del documento e formulare la query. Per ogni elemento, il sistema conosce e può mostrare all' utente la *semantica* dell' elemento e i *vincoli* esistenti.

La query viene preparata in un *formato generico*, che può successivamente essere trasformato nel formato previsto dal search engine sottostante.

```
<books>
  <book number="1">
    <metadata>
      <author>Oreste Signore</author>
      <title>The evolving Web</title>
    </metadata>
    <content>
      <introduction>
        <author>Oreste Signore</author>
      </introduction>
      <part number="1">
        <chapter>
          <author>Chris Green</author>
          <author>Oreste Signore</author>
          <author>John White</author>
          <title>About the future of the Web</title>
        </chapter>
      </part>
      <part number="2">
        <chapter>
          <author>John Smith</author>
          <author>Chris Green</author>
          <author>Jim Brown</author>
          <title>The future of technologies</title>
        </chapter>
      </part>
    </content>
  </book>
</books>
```

Figura 1 - Un esempio di documento XML

La query

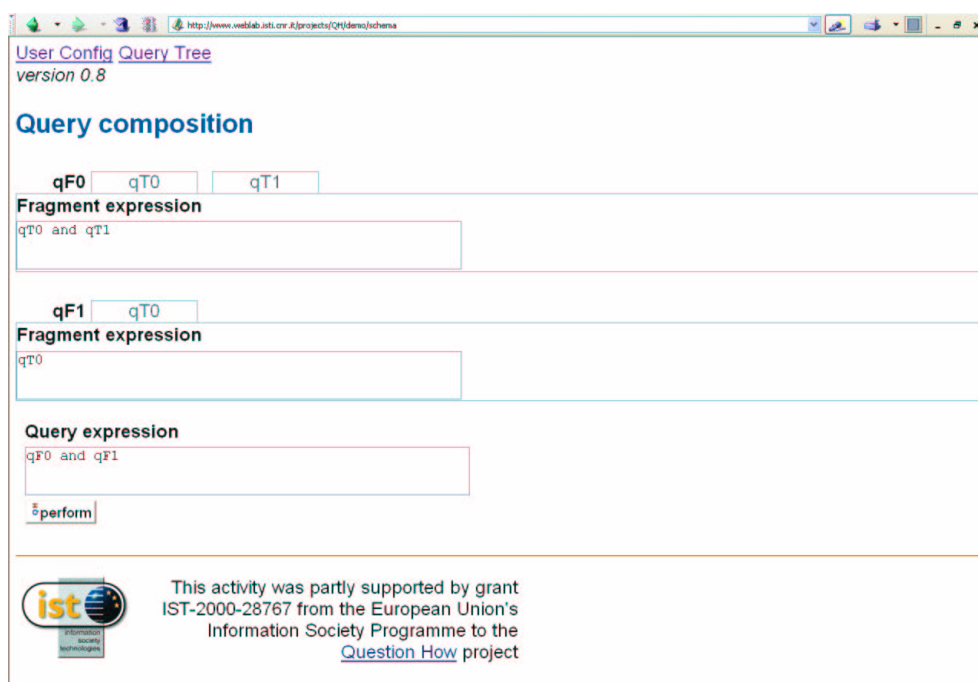
La query viene composta interattivamente navigando sulla struttura del documento e interagendo con una QueryForm predisposta sulla base delle informazioni deducibili dallo schema e dalla informazioni aggiuntive specificate nella annotazione RDF.

Come è ben noto fin dal tempo dei sistemi gerarchici, esiste un problema di normalizzazione della query, ovvero di individuazione del padre comune agli elementi sui quali si impongono delle condizioni. In altri termini, quando si specificano condizioni su rami diversi dell' albero, bisogna specificare qual è la radice del sottoalbero rispetto alla quale devono essere verificate le condizioni. Per esempio, interrogando una collezione di documenti contenente record del tipo riportato in Figura 1, una query che imponesse le condizioni

author="Oreste Signore" AND title="The future of technologies"

deve specificare la radice ([chapter](#) o [content](#)) del sottoalbero nel quale le due condizioni devono essere valide.

L' individuazione della radice del sottoalbero viene legata naturalmente, dal punto di vista utente, ad una navigazione nella struttura, durante la quale vengono specificate le condizioni sui singoli elementi.



The screenshot shows a web browser window with the URL <http://www.weblab.isti.cnr.it/projects/QT/demos/schema>. The page title is "User Config Query Tree" and the version is "version 0.8". The main heading is "Query composition".

The interface is divided into three main sections:

- Fragment expression:** This section contains two input fields. The first is labeled "qF0" and contains the text "qT0 and qT1". The second is labeled "qT0" and contains the text "qT0".
- Query expression:** This section contains one input field labeled "qF0" and "qF1" which contains the text "qF0 and qF1".

At the bottom of the form, there is a "perform" button. Below the form, there is a logo for "ist" (Information Society Technologies) and a text block: "This activity was partly supported by grant IST-2000-28767 from the European Union's Information Society Programme to the [Question How](#) project".

Figura 2 - Fase di editing della query (lingua inglese)

In base a queste considerazioni, e anche avendo ben presente che la composizione di una query non banale, con vari livelli di parentesi, risulta sempre complessa e farraginoso, si è ritenuto opportuno definire la query come la combinazione di vari elementi.

Una **query** è la composizione booleana (con livelli di parentesi) di uno o più *queryFragment*. Per default, la query viene preparata mettendo in AND tutti i *queryFragment*. L'utente può editare la query, modificando la combinazione dei *queryFragment* e introducendo un numero arbitrario di operatori booleani e di livelli di parentesi (Figura 2).

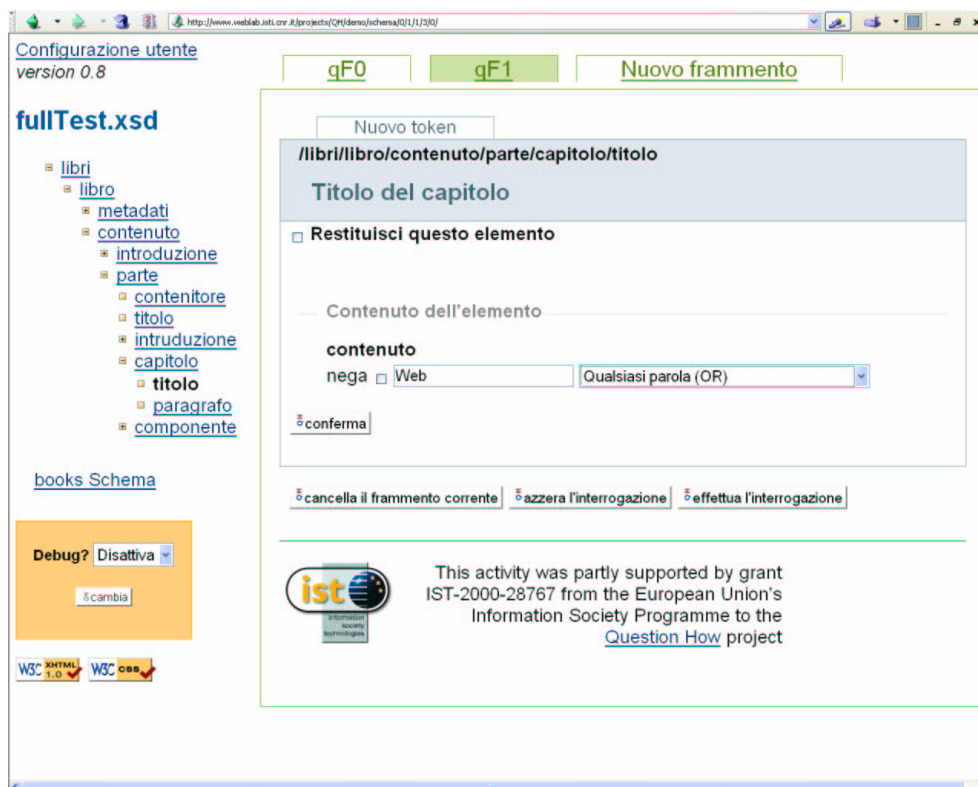


Figura 3 - Formulazione di un queryToken (lingua italiana)

Un **queryFragment** è la composizione booleana (con livelli di parentesi) di *queryToken*. Il *queryFragment* viene costruito automaticamente come conseguenza di una navigazione sul *documentTree*, durante la quale vengono specificate le condizioni sui vari elementi. Per default, il sistema assume che i *queryToken* siano connessi con l'operatore booleano AND, ma l'utente può modificare il *queryFragment* (identificato come qF0, qF1, ...) inserendo livelli di parentesi e combinando in vario modo i *queryToken*. Ogni volta che l'utente specifica "Nuovo frammento" (o "New fragment") viene iniziata una nuova navigazione sul document tree, e creato un nuovo *queryFragment*.

Il **queryToken** (identificato come qT0, qT1, ... all'interno del singolo qF) è il componente elementare della query, e viene creato mediante un'interazione con il *queryForm* (Figura 3). Nel caso in cui si vogliono specificare condizioni su più istanze dell'elemento, è necessario iniziare una nuova interazione sullo stesso elemento. Nell'esempio di Figura 1, per ricercare i libri scritti da due autori occorrerebbe specificare due *queryToken*: `<author>="X" AND <author>="Y"`, mentre, se l'elemento `<author>` contenesse tutti

gli autori del libro, occorrerebbe specificare: `<author>= ("X" AND "Y")`. Quale sia la modalità da utilizzare dipende quindi dalla struttura del documento, che può essere resa nota all'utente sfruttando la metainformazione codificata nell'opportuno elemento della annotazione RDF (longDescription).

L'annotazione RDF

L'amministratore della collezione di documenti, che si suppone abbia una conoscenza approfondita sia della collezione che del dominio di conoscenza relativo, ha il compito di definire le modalità tipiche di interazione per i vari *tipi di utente* e *formalizzare la metainformazione*.

Per quanto concerne il primo aspetto, è possibile definire alcune caratteristiche dell'utente, che possono influire sulle modalità di interazione: il *livello* (neofita o esperto), la *lingua* preferita per l'interazione, il *tipo di informazione* restituita (record completo, parte di esso, etc.).

Si noti, riguardo all'effetto delle caratteristiche specificate nel profilo utente, come l'informazione in Figura 2 e Figura 3 sia in due lingue diverse. Il profilo utente è modificabile durante la sessione, con effetto immediato.

L'aspetto più significativo è tuttavia quello dei metadati relativi ai singoli elementi. Alcune informazioni possono essere desunte direttamente dallo schema (es. il tipo, la molteplicità, la lista o l'intervallo di valori ammessi, etc.). Tuttavia, non sempre lo schema contiene tutte queste informazioni, e ve ne sono alcune che non possono essere espresse (riferimento ad ontologie, dipendenze dei valori di elementi). Di conseguenza, sono state identificate alcune proprietà, che possono essere suddivise, per comodità, in vari gruppi:

- *Identificazione* degli elementi: per poter identificare univocamente gli elementi o attributi nel contesto del documento.
- *Proprietà* degli elementi:
 - descrizione (per esprimere più chiaramente il loro significato e supportare lingue diverse)
 - ricercabilità (per individuare gli elementi che hanno una valenza stilistica, ma non semantica)
 - ruolo (nodi puramente strutturali o con contenuto)
- *Vincoli*
 - thesaurus, dizionari, liste esterne, liste locali
 - web service da invocare
 - dipendenze tra i valori assunti
- *Opzioni di pre-processing* (per esempio per supportare formati più adeguati alla cultura dell'utente)
- *Opzioni di post-processing* per manipolare e arricchire i documenti restituiti.

Una descrizione più organica è in [QH-Deliverable]

Dettagli implementativi

Il search engine: XCDE

Nell' implementazione corrente, è stato adottato come search engine la *libreria XCDE*, sviluppata presso il Dipartimento di Informatica dell' Università di Pisa (prof. Paolo Ferragina, gruppo algoritmi). Nell' ambito di questo progetto, la libreria è stata estesa allo scopo di supportare un query language sofisticato che combina alcune caratteristiche di XQuery ([Xquery]) con altre tipiche dei sistemi di Information Retrieval. Questa direzione di ricerca è stata suggerita recentemente anche in ambito W3C (XQuery-FT).

Si tratta di uno dei primi tentativi di implementare questa proposta utilizzando indici compressi su documenti XML. Il query language definito consente interrogazioni testuali sofisticate su documenti XML: prossimità di parole, ricerca con espressioni regolari o con corrispondenza parziale, estrazione di *snippet* per selezionare il contesto di un documento reperito, ricerca per sottostringhe, troncamento di prefissi e suffissi, etc..

La libreria, scritta in C, è utilizzabile liberamente per scopi non a fini di lucro.

La libreria XCDE. - La libreria fornisce un insieme di algoritmi e strutture dati per indicizzare e ricercare collezioni di documenti XML. I documenti devono essere *well-formed* e possono essere eterogenei, facendo riferimento a DTD o XML Schema diversi. La libreria supporta l' archiviazione e la gestione dei file XML in forma nativa, operando direttamente a livello di file system.

La principale struttura dati di indicizzazione è la classica lista invertita, in cui vengono utilizzate opportune tecniche di compressione. Vengono indicizzati indipendentemente le parole, i tag, gli attributi e i loro valori, consentendo così l' esecuzione efficiente di query sofisticate. Le query che coinvolgono la struttura del documento possono essere eseguite in modo efficiente grazie ad un nuovo metodo di indicizzazione della struttura gerarchica, basato su *strutture dati geometriche*.

Il testo originale viene compresso in modo che l' accesso ad alcune parti di esso possa essere eseguito senza decomprimere l' intero file. Questa caratteristica si rivela particolarmente utile per presentare rapidamente il risultato di una query (detto normalmente *snippet*). Complessivamente, il documento XML compresso con tutti i suoi indici ha una *dimensione all' incirca pari a quella del documento originale*. Questa caratteristica rende XCDE nettamente migliore di tutti gli altri approcci conosciuti, e può costituire un elemento importantissimo per applicazioni su apparecchiature con memoria limitata.

Una delle scelte di progetto in XCDE è stata quella di operare a livello di granularità del *singolo documento*. Lo svantaggio è che le ricerche su collezioni costituite da molti documenti di dimensioni ridotte possono risultare lunghe. D' altra parte, vi sono alcuni vantaggi, quali la realizzazione di indici distribuiti, la facilità di aggiornamento dell' indice a fronte di inserimenti o aggiornamenti dei documenti, possibilità di gestire agevolmente documenti XML eterogenei e di personalizzare gli indici sulla base delle caratteristiche del documento. È comunque possibile indicizzare collezioni di documenti abbastanza vaste senza pagare in prestazioni con due accorgimenti: o con un "*document merging*", combinando più documenti in un unico documento più ampio, o aggiungendo un "*indice leggero*" per filtrare parte della collezione di documenti prima di eseguire la query sui singoli documenti.

La libreria fornisce una API con un ricco insieme di funzioni C con cui è possibile implementare la maggior parte delle funzionalità di base di XQuery, e supportare query più

complesse di tipo IR. L' obiettivo è la gestione efficace ed efficiente di documenti caratterizzati da una rilevante porzione di testo libero e/o struttura gerarchica profonda, in cui i valori degli attributi sono sequenze complicate di numeri e lettere. Questo è lo scenario tipico che si incontra nella gestione di archivi di testi letterari con tag XML-TEI, una delle applicazioni che hanno portato alla definizione della libreria. In questo contesto, la granularità a livello di singolo documento non costituisce un problema.

XCDE Query Syntax – La sintassi supportata da XCDE per la formulazione della query è simile alla classica sintassi SQL: *SELECT-FROM-RETURN*.

L' output della query (lo snippet) può essere formattato nella clausola Return, che, a sua volta, include una porzione di *XML well-formed text*.

Un attributo particolare, denominato *xml_var (pivot)* viene aggiunto nella clausola SELECT per identificare i sottoalberi che soddisfano la query.

Ambiente software

L' ambiente applicativo in cui è stato sviluppato il progetto è:

- Zope come application server;
- Python come linguaggio di programmazione;
- Soaplib Python;
- parsedXML;
- Postgresql (per la memorizzazione di dizionari e thesauri)

La versione attuale dipende da parsedXML, in quanto utilizza le sue funzioni DOM per accedere a XML schema. La prossima versione utilizzerà invece le funzioni SAX incluse in Python.

Tra i componenti completamente realizzati fino a questo punto sono:

- la selezione dello User Profile e della Document Collection;
- un parser per XMLSchema (con alcune limitazioni);
- un modulo che trasforma un XML Schema in un albero XHTML, consentendo la navigazione sull' albero che rappresenta il documento;
- l' interfaccia di amministrazione per memorizzare gli *Element Metadata*;
- un *form builder* basato su XMLSchema e *Element Metadata*;
- un componente modulare per assemblare la query, editare i queryFragment e la query finale, tradurre la query nel formato richiesto da XCDE;
- il wrapper per chiamare le funzioni di libreria di XCDE e recuperare i risultati restituiti;
- un web service per verificare i termini consultando un dizionario;
- una applicazione di test per la navigazione su un thesaurus, che consente di selezionare uno o più valori da inserire nella query.

Il caso di test

Come campione è stata utilizzata la collezione di documenti a cui fa riferimento il W3C Working Draft [Xquery-FT].

Su questa collezione è stata predisposta una annotazione RDF dello XMLSchema in cui sono stati specificati vincoli di:

- lista locale
- lista esterna
- thesaurus

- elementi correlati

Conclusioni e sviluppi futuri

È stato definito un contesto consistente per l' accesso intelligente e controllato a collezioni di documenti XML.

Lo schema della collezione non è stato modificato, ma annotato esternamente utilizzando RDF.

L' amministratore della collezione può fare riferimento a una qualunque fonte di conoscenza esterna, invocando i thesauri di supporto specificando l' opportuno web service (se esiste) o con un tradizionale accesso CGI. Le modalità di invocazione del servizio o della particolare applicazione vengono specificate su una opportuna risorsa XML, identificata da un URI.

L' interfaccia è generale, e può essere utilizzata verso una qualunque collezione di documenti e adattata a qualunque motore di ricerca. Al momento, tuttavia, non è prevista la specifica delle funzioni e degli operatori di ricerca supportati dal motore di ricerca sottostante.

Alcune funzioni non sono state ancora implementate, ma non presentano difficoltà concettuali.

Come sviluppi futuri, sono in fase di considerazione la possibilità di utilizzare questa interfaccia per aggiornare documenti XML e quella di accedere collezioni di documenti eterogenei. Quest' ultima linea di ricerca prevede di poter rappresentare e comprendere la semantica della struttura dei documenti.

Ringraziamenti

Un doveroso ringraziamento va al prof. Paolo Ferragina del Dipartimento di Informatica dell' Università di Pisa, e ai suoi collaboratori Alessandro Perucci e Rosanna Rossi.

Bibliografia

- | | |
|------------------|--|
| [DC] | The Dublin Core Home Page, URL: http://dublincore.org/ |
| [SemWeb] | http://www.semanticweb.org/ |
| [QH-Deliverable] | http://www.weblab.isti.cnr.it/projects/QH/docs/deliverable.html |
| [XCDE] | http://butirro.di.unipi.it/~ferrax/xcde/xcdelib.html |
| [XQuery] | XQuery 1.0: An XML Query Language - W3C Working Draft 22 August 2003, http://www.w3.org/TR/xquery/ |
| [XQuery-FT] | XQuery and XPath Full-Text Requirements - W3C Working Draft 02 May 2003, http://www.w3.org/TR/xquery-full-text-requirements/ |