XML: RAPPRESENTARE E INTERROGARE DATI SEMI-STRUTTURATI

Daniele Braga
Alessandro Campi
Stefano Ceri

La nostra società si caratterizza sempre più come società dell'informazione, nella quale l'infrastruttura informatica svolge il ruolo cruciale di gestire la conservazione e lo scambio dei dati. In questo scenario, XML è ormai diventato lo standard per la rappresentazione e lo scambio dei dati in moltissime applicazioni. Questo articolo descrive le caratteristiche di XML che ne hanno determinato il successo e presenta i linguaggi e le tecnologie per la manipolazione di dati codificati in XML.

1. DATI O DOCUMENTI? LA CONVERGENZA DI DUE VITE PARALLELE

"*La disposizione della memoria su un nastro infinito non è soddisfacente in pratica, per il dispendio di tempo andando avanti e indietro alla ricerca delle unità di informazione richieste nei vari momenti. Questa difficoltà doveva angustiare gli antichi egizi, che scrivevano i loro libri su rotoli di papiro*". Così nel 1947 A. M. Turing, padre dell'informatica teorica e pioniere di quella pratica, provocava una platea di matematici a proposito del principale difetto di un metodo per rappresentare ed elaborare l'informazione che lui stesso aveva teorizzato, e che consentiva di effettuare *qualsiasi* operazione computabile.

Un'accusa analoga poteva essere mossa, alla fine degli anni 90, alla consolidata tecnologia dei database relazionali, per i loro limiti nella gestione efficiente dei *dati semi-strutturati*, cioè informazioni di un tipo relativamente nuovo – per popolarità e diffusione – la cui mole ed importanza da allora sono cresciute sempre più rapidamente, in parallelo con il fenomeno Internet.

Alla base dall'affermazione di Turing, infatti, c'è una considerazione che è evidente per chiunque si occupi di informatica: il modo in cui sono rappresentate le informazioni ha un impatto sulla complessità e sull'efficienza delle applicazioni che devono gestirle. Nel caso in esame, i documenti che costituiscono il Web contengono prevalentemente dati se-

Dati e Documenti nel tempo

Nel mondo dei database
1970 database relazionali
1990 nested relational model e database object-oriented
1995 database semi-strutturati

La gestione dei documenti
1986 SGML
1990 HTML
1992 URL

Dati + documenti = informazione
1997 XML
1999 XPath 1.0 e XSLT 1.0
2004 XQuery 1.0 e XSLT 2.0 (draft quasi consolidati)

mi-strutturati, una tipologia di informazione troppo liberamente strutturata, come già accennato, per essere rigidamente imbrigliata nelle tabelle dei database relazionali e gestita in modo efficiente.
L'esigenza di una rappresentazione efficace per tali documenti, unitamente all'esigenza di flessibilità di utenti e applicazioni che spesso devono *integrare* dati provenienti da sorgenti *eterogenee*, ha favorito la diffusione di **XML**, rapidamente adottato come formato standard per la codifica di informazioni semistrutturate. Infatti la principale caratteristica di XML, e anche uno dei suoi principali punti di forza, è l'attenuazione della distinzione tra **dati** e **schema dei dati**, con la possibilità di rappresentare un ampio spettro di situazioni intermedie tra gli estremi dell'informazione totalmente destrutturata (ad esempio un segnale audio) e dell'informazione del tutto vincolata a uno schema rigido (come nei database relazionali). In questi mesi, inoltre, si stanno consolidando, dopo un processo lungo, controverso e faticoso ad opera del World Wide Web Consortium (W3C) anche le specifiche di **XPath, XQuery** e **XSLT**, i linguaggi candidati ad essere lo standard per interrogare e manipolare dati rappresentati in XML.
Questo articolo, dopo una breve introduzione ad XML, presenta dapprima alcuni scenari in cui esso si dimostra utile ed efficace, descrive poi XPath, XSLT e XQuery basandosi su una progressione di esempi, e termina con una panoramica sullo stato dell'arte delle tecnologie collegate ad XML.

2. XML PER RAPPRESENTARE LE INFORMAZIONI

2.1. Il segreto del successo di XML

XML (*eXtensible Markup Language*) è un "linguaggio a marcatori" (*tag language*) estremamente flessibile, derivato da SGML (*Standard Generalized Markup Language*) e nato per gestire la pubblicazione su larga scala di documenti elettronici tramite il Web. In seguito, XML si è rivelato adatto a rappresentare dati in contesti estremamente eterogenei.

Da SGML a XML

SGML è un linguaggio per la definizione di linguaggi di marcatura e offre costrutti per specificarne la sintassi (i vincoli da rispettare nel comporre i tag) e la semantica (il significato dei tag, cioè il comportamento del programma che li interpreta). SGML si consolida a metà degli anno '80 come sintesi di circa venti anni di sforzi profusi nel mondo dell'editoria elettronica per la standardizzazione di un meccanismo generale di definizione di stili di marcatura diversi, seppure in un contesto omogeneo. Già negli anni '60, infatti, si era avvertita la necessità di superare l'uso di codici di controllo specifici di ogni particolare formato, a vantaggio di uno schema generale di definizione (il cosiddetto "GenCode® concept") che permettesse di separare il contenuto di un documento dalla sua formattazione e di definire una struttura gerarchica in cui piccoli documenti potessero essere inclusi come parti di documenti più ampi. Nel 1969 IBM propose GML (ad opera di *G*oldfarb, *M*osher e *L*orie, ed anche acronimo di *Generalized Markup Language*), che per primo introdusse il concetto di "tipo di documento" come classe di documenti che si attengono a precise regole di struttura e di formattazione, definite da uno "schema di marcatura". SGML deriva da GML con l'aggiunta, per esempio, di un sistema di collegamento tra documenti tramite riferimenti.
Alcuni anni dopo, HTML fu definito a partire da SGML come un linguaggio di formattazione per documenti particolari, gli ipertesti. Un ipertesto, come oggi siamo ormai abituati a concepirlo, si rappresenta bene con un linguaggio di marcatura e presenta proprio le problematiche che hanno motivato la nascita di SGML. La semplicità di HTML è stata però ottenuta al prezzo del suo limite principale: l'impossibilità di estenderlo se non in modo "proprietario". Con l'aumentare della mole dei dati pubblicati sul Web si è diffusa l'idea di realizzare un linguaggio più generico, che consentisse di definire tag proprietari, ma che non avesse la complessità di SGML. Così si è aperta la strada che porta ad XML.

XML

Nel 1996 è stato proposto XML, un linguaggio che consentiva di realizzare browser completamente estensibili grazie alla possibilità di definire attraverso il linguaggio e "dentro" i documenti la tipologia dei tag ammessi e la struttura del linguaggio di marcatura. Lo standard

di riferimento era SGML, ma occorreva considerare anche esigenze nate col Web e con le sue tecnologie. Uno degli obiettivi era quello di poter includere nel documento una semplice specifica della sua struttura e del suo significato, in modo da renderlo "autocontenuto" dal punto di vista della possibilità di interpretarlo. Per fare ciò si è introdotto il DTD (*Document Type Definition*), una specifica sintetica della struttura di una classe di documenti.
In sintesi XML è:

- Un formato generale usabile in ogni contesto;
- Standardizzato dal W3C (www.w3.org);
- Leggibile sia per le applicazioni software che per l'occhio umano;
- Semplice da analizzare in modo automatico;
- Internazionalizzato (attraverso la codifica UNICODE);
- Semplice ed efficace nel dare semantica al testo;
- Indipendente da piattaforme e fornitori/venditori;
- Una tecnologia che non richiede grossi investimenti.

2.2. L'abc di XML

Un documento XML è un documento testuale le cui sezioni sono racchiuse all'interno di coppie di marcatori (*o tag*). I tag sono distinti dal testo libero (o *PCDATA*, acronimo di Parsable Character DATA) per il fatto di iniziare e finire con le parentesi angolari '<' e '>'. Ogni sezione è definita da un tag di apertura e un tag di chiusura che si corrispondono e che sono dotati di una propria struttura; in particolare il tag di apertura può contenere una lista di attributi, mentre il tag di chiusura è riconoscibile perché inizia con la sequenza '</'. L'unione di una coppia di tag corrispondenti e del loro contenuto è detta *elemento*; in un documento è sempre presente un unico elemento iniziale, detto *radice*, che racchiude l'intero documento.
Gli elementi possono sia avere *attributi*, elencati nei tag di apertura in forma di coppie nome/valore, per specificarne alcune caratteristiche, sia contenere *sotto-elementi* di arbitraria complessità, cioè che a loro volta possono (ricorsivamente) contenere altri sotto-elementi. Il contenuto di un elemento può anche essere costituito soltanto da PCDATA, o da una combinazione di PCDATA e di sotto-elementi, liberamente mescolati tra loro (in questo ultimo caso si doce che l'elemento è *misto*).
I tag di apertura e di chiusura si corrispondono col rigido vincolo per cui non si può chiudere un elemento finché non sono stati chiusi tutti i suoi sotto-elementi, realizzando così una struttura gerarchica rigidamente strutturata come le espressioni con parentesi correttamente annidate a più livelli. Un documento che rispetti queste semplici regole sintattiche si dice *ben formato*.
Il seguente esempio (Figura 1) mostra come un elenco di dischi di vari produttori può es-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<musica>
  <produttore>
    <nome> Karim </nome>
    <sede> Roma </sede>
    <catalogo>
      <disco anno="1961">
        <titolo> Nuvole Barocche </titolo>
        <artista> Fabrizio De André </artista>
      </disco>
      <disco anno="1965">
        <titolo> La Città Vecchia </titolo>
        <artista> F. De André </artista>
      </disco>
        …
    </catalogo>
</produttore>
        …
</musica>
```

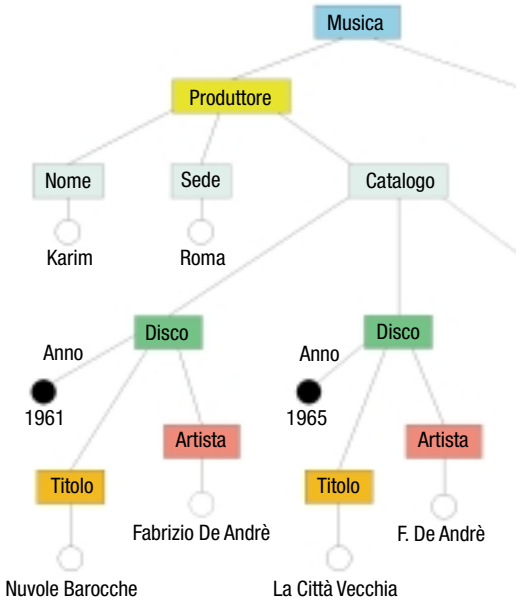

FIGURA 1
Esempio di documento ben formato

sere rappresentato in XML; gli elementi che rappresentano ogni disco sono elencati *dentro* (o *sotto*) ad un elemento catalogo, e così via fino all'elemento iniziale (musica). Per questo si dice che il modello dei dati di XML è *gerarchico*, e spesso se ne utilizza una rappresentazione ad albero che aiuta a visualizzare la gerarchia tra i vari elementi.
La classe di documenti che rispettano la struttura del documento d'esempio può essere per esempio definita dal DTD seguente:

```
<!ELEMENT musica (produttore+)>
<!ELEMENT produttore (nome, sede, catalogo)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT sede (#PCDATA)>
<!ELEMENT catalogo (disco+)>
<!ELEMENT disco (titolo, artista)>
<!ATTLIST disco anno CDATA #REQUIRED>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT artista (#PCDATA)>
```

Lo si legge piuttosto intuitivamente come segue: l'elemento radice *musica* contiene una lista di (uno o più) elementi *produttore*, ognuno dei quali è composto di una tripletta di elementi: *nome*, *sede* e *catalogo*. I primi due, come anche in seguito *titolo* e *artista*, contengono solo testo libero (PCDATA), mentre ogni catalogo contiene una lista di (uno o più) elementi disco, ognuno con un attributo "anno" (che essendo dichiarato #REQUIRED non può essere omesso), un titolo e il nome dell'artista. Ogni documento che sia ben formato e che rispetti i vincoli strutturali così specificati si dice *valido* rispetto al DTD.
Il principale limite di questo modo di specificare lo schema di una classe di documenti XML è la povertà del sistema di tipizzazione dei dati, che si riduce alla distinzione tra testo libero e tag. In molti contesti questa semplicità si è mostrata un vantaggio competitivo, ma per ovviare a questo limite, nel caso in cui sia necessario definire tipi di dato come nei tradizionali linguaggi di programmazione, è stato introdotto XML Schema (www.w3.org/XML/Schema), che prevede un ampio insieme di tipi di base e numerose primitive di costruzione di tipi complessi a partire dai tipi di base. Una trattazione esaustiva di XML Schema esula dagli scopi di questo articolo.

3. ADOTTARE XML? SCENARI, VANTAGGI, SVANTAGGI E SOLUZIONI IBRIDE

Le ragioni del successo di XML sono molteplici e assai diverse fra loro. Vediamo una rassegna di alcuni dei suoi più importanti ambiti applicativi.

3.1. XML come "lingua franca" per lo scambio di dati

Le caratteristiche di XML che abbiamo descritto mostrano come esso permetta a qualsiasi applicazione di rappresentare informazione in modo indipendente sia dal linguaggio di programmazione in cui è stata sviluppata sia dal sistema operativo su cui è in esecuzione. XML, pertanto, si candida ad essere un ottimo veicolo per scambiare informazioni tra applicazioni arbitrariamente eterogenee. Naturalmente XML può essere usato anche per lo scambio di dati tra componenti di una stessa applicazione: programmatori diversi possono lavorare in modo indipendente, a patto di concordare un formato comune per la rappresentazione dei dati da scambiare, sfruttando così una tecnologia già consolidata per l'interoperabilità esterna, al fine di conseguire una maggiore "apertura" anche nell'architettura interna delle applicazioni.

3.2. XML come linguaggio per la configurazione di sistemi di data management

La semplicità e la standardizzazione di XML ne fanno anche il linguaggio ideale per rappresentare le proprietà di configurazione del software, in virtù del requisito che lo vuole agevolmente leggibile sia per le applicazioni sia per l'occhio umano; il suo uso sistematico permette di avere file di configurazione altrettanto semplici da modificare tanto per intervento "manuale" diretto quanto per intervento mediato via software.
Un esempio significativo di questo uso di XML è *Hibernate* (http://www.hibernate.org), un potente strumento che consente al progettista/programmatore di garantire (con poche linee di codice) la *persistenza* degli oggetti di una applicazione Java su database relazionali. Hibernate consente, con poche istruzioni, di "salvare" (rendere persistente) in un database lo stato di un'applicazione Java (assieme a tut-

ti i suoi oggetti) per ricaricarlo in un secondo momento. XML interviene per specificare l'associazione tra le tabelle di un database (e i relativi campi) e le classi dell'applicazione (e relativi attributi). L'esempio seguente mostra com'è possibile associare la classe Java *Prodotto* (identificata tramite la sua posizione nella gerarchia dei package) alla tabella relazionale *T-PROD*, al fine di mantenere automaticamente allineati i valori degli attributi degli oggetti con quelli dei record della tabella.

```
<hibernate-mapping>
   <class name="com.mokabyte.samples.Prodotto"
     table="T_PROD">
       <property name="id" type="string"/>
       <property name="descrizione" type="string"/>
   </class>
</hibernate-mapping>
```

Quello mostrato è un esempio molto semplice, ma il linguaggio consente la specifica di corrispondenze più sofisticate.

3.3. XML come ponte tra programmatori e grafici

Le attuali applicazioni Web hanno le esigenze contrastanti di pubblicare grandi quantità di dati e di permettere all'utente di visualizzarli o modificarli in modo semplice e intuitivo. Lo sviluppo e la manutenzione di tali applicazioni è un'attività particolarmente complessa, che richiede conoscenze multidisciplinari, per l'impiego di tecniche proprie dell'ingegneria del software unite alla capacità di realizzare pagine dall'aspetto gradevole. Si rende quindi necessaria la collaborazione di esperti informatici e designer grafici: il compito dei primi è realizzare la logica dell'applicazione (gestione dati, logica di business ecc.), realizzando lo strato software che elabora le richieste dell'utente e fornisce le informazioni richieste; il compito dei secondi è realizzare un livello di presentazione che mostri al meglio le informazioni fornite dallo strato sottostante. È importante notare che il livello di presentazione non è necessariamente unico: le richieste possono arrivare da personal computer, che possono accedere ad una versione del sito fornita di una raffinata interfaccia grafica (magari in Flash©), oppure da dispositivi con risorse di calcolo più limitate, come per esempio i telefoni cellulari, che impongono un'interfaccia meno sofisticata. La comunicazione tra i due livelli avviene grazie a XML: lo strato della logica applicativa risponde alle richieste fornendo informazioni codificate in XML, che lo strato di presentazione elabora costruendo dinamicamente un'interfaccia utente codificata in un modo qualsivoglia, ad esempio in Flash oppure con trasformazioni XSLT che generano HTML; vedremo più avanti in questo articolo che proprio XSLT è uno dei linguaggi chiave per utilizzare XML, esemplificando la generazione di pagine HTML.

3.4. ebXML

ebXML (*Electronic Business using eXtensible Markup Language*, http://www.ebxml.org) è un insieme di specifiche nate per offrire alle organizzazioni geograficamente distribuite la possibilità di effettuare transazioni commerciali attraverso il Web.
Il valore di ebXML consiste nell'essere uno standard aperto, condiviso a livello globale e nato da una solida esperienza nell'ambito degli scambi commerciali elettronici. ebXML rappresenta un concreto passo avanti verso la creazione di un unico mercato elettronico globale in cui ogni soggetto può entrare liberamente.

3.5. I Web Service

I Web Service, sempre più usati per fornire servizi informatici via Web, rappresentano un nuovo paradigma per la costruzione di *sistemi informativi distribuiti* basati sulla composizione di servizi.
I Web Service sono applicazioni tra loro indipendenti e basate su standard aperti. Possono essere pubblicati, ricercati e liberamente utilizzati attraverso la rete. Possono essere aggregati per creare nuove applicazioni, servizi e processi. Possono essere pubblicati con una "auto-descrizione", che un servizio può sfruttare per ricercare altri servizi con cui tentare di interagire dinamicamente. I pilastri tecnologici su cui poggia l'idea dei Web Service sono tre: **SOAP**, il protocollo per lo scambio di messaggi nella comunicazione con un servizio; **WSDL**, Il linguaggio standard per la specifica dell'inter-

faccia di un servizio; **UDDI**, per l'indicizzazione dei servizi. Questi standard sono tutti e tre basati su XML e il motivo principale è che tutti i linguaggi di programmazione e tutte le piattaforme possono spedire e ricevere dati in formato XML. Proprio XML, quindi, rende i Web Service realmente utilizzabili da parte di applicazioni eterogenee e su piattaforme eterogenee.

3.6. Ontologie e Semantic Web

La disponibilità dei Web service rende possibile l'utilizzo di servizi altrui per realizzare parte di un servizio che si vuole offrire. Un requisito fondamentale per poter comporre Web Service eterogenei nella realizzazione di un ulteriore servizio è la disponibilità di descrizioni ricche e molto accurate dei servizi stessi (il loro "contratto di utilizzo"); inoltre, per velocizzare e semplificare la ricerca, il confronto e la negoziazione tra diversi servizi che soddisfano un certo profilo tecnico e contrattuale, tali descrizioni devono essere pubblicamente accessibili e analizzabili tramite un processo automatico.
Il primo passo in questa direzione è rappresentato dalla standardizzazione dei linguaggi di descrizione delle interfacce dei servizi (WSDL, UDDI), ma, guardando gli elenchi di Web Service disponibili in rete (come per esempio http://www.xmethods.com), si nota che la ricerca e la pubblicazione di Web Service si basano sull'ipotesi che i termini usati siano comprensibili a tutti i soggetti coinvolti. Al momento sembra ancora impossibile costruire un servizio senza che il client e il server concordino sull'uso dei termini della transazione che si vuole realizzare.
Il motivo per cui è necessaria questa contrattazione preliminare non è connesso ai Web Service in sé, ma a un limite intrinseco di XML: il fatto che XML sia agevolmente trattabile con sistemi automatici non ne garantisce di per sé la totale interoperabilità. XML permette di inviare messaggi trattabili in modo automatico, ma che quei messaggi abbiano senso per il destinatario non può essere garantito da un'analisi automatica. Con questo non si vuole sminuire il ruolo ricoperto dall'XML, ma precisare che esso può offrire solo interoperabilità *sintattica* e strutturale (e già si è detto quanto questo sia di aiuto per realizzare applicazioni distribuite), ma non una reale condivisione di conoscenza, quando non vi sia già una *semantica* condivisa.
Come conseguire quindi interoperabilità tra due parti che non sono mai entrate in contatto prima? Per colmare questa lacuna e tentare di rispondere a questa esigenza si può ricorrere alle **ontologie**, che sono specifiche formali di concettualizzazioni che descrivono una comprensione comune di un dominio, la quale è concordata da una pluralità di soggetti e può essere deliberatamente condivisa tra persone diverse e applicazioni diverse.
L'idea fondamentale del **Semantic Web** è affiancare al meccanismo di scambio di dati alla base dei Web Service anche una descrizione dei domini realizzata tramite ontologie. L'uso delle ontologie permetterà in futuro di mediare tra sistemi molto eterogenei e (per esempio) di rendere la ricerca di Web Service più efficace. Le ontologie, nate indipendentemente da XML e dai Web Service, hanno probabilmente trovato nel Semantic Web la loro applicazione fondamentale, ma per uscire dal mondo accademico ed avere una concreta ricaduta sulle tecnologie di più largo uso occorre che siano espresse in un formato standard; anche in questo contesto XML si è rivelato il miglior candidato per rendere le ontologie lo standard di riferimento per denotare il significato dei termini e rappresentare le relazioni che tra essi intercorrono. È auspicabile che in futuro, anche grazie a tutta l'infrastruttura tecnologica messa a disposizione da XML, le ontologie permettano la realizzazione di applicazioni sempre più complesse, che riescano progressivamente a catturare il significato dei loro domini applicativi.

3.7. Standard per la descrizione di domini applicativi in XML

Oltre che per gli scopi precedentemente descritti, XML è usato come standard per rappresentare informazioni in moltissimi domini applicativi specifici. A titolo puramente esemplificativo, ecco alcuni esempi di linguaggi usati in domini molto diversi tra loro, scelti tra i moltissimi standard esistenti:
- *HealthCare Level Seven (http://www.hl7.org)*;

- *Geography Markup Language (GML - http://www.opengeospatial.org)*;
- *Systems Biology Markup Language (SBML - http://sbml.org)*;
- *XML based Business Reporting standard (XBRL - http://www.xbrl.org)*;
- *Global Justice XML Data Model (GJXDM - http://it.ojp.gov/jxdm)*.

3.8. Il denominatore comune

Cosa lega tutte le applicazioni viste finora? Perché XML si è rivelato particolarmente adatto per tutte? Come si è visto, i contesti applicativi descritti sono estremamente diversi. Ad unirli è solo il bisogno di uno standard semplice e generale. XML risponde a questo bisogno in quanto è semplice da elaborare da parte dei programmi e si basa su un modello dei dati semplice e intuitivo.

3.9. L'ostacolo comune: dove sono i dati e i documenti XML?

Possiamo quindi pensare al Web (di oggi o di domani) come a un grande database di pubblico accesso? È una definizione suggestiva, ma anche poco realistica, perché la maggior parte dei dati è protetta da applicazioni che li mascherano e ne limitano l'accessibilità. Anche se non mancano le applicazioni che espongono i dati in formati aperti (tra cui quelli citati precedentemente), è chiaro che la presenza di numerosi standard per la codifica e o scambio dei dati non può vincere di per sé la resistenza a rendere le informazioni pubblicamente accessibili; ogni comunità tenderà sempre a proteggere il proprio patrimonio informativo.

4. LINGUAGGI DI INTERROGAZIONE E TRASFORMAZIONE

Abbiamo visto come XML sia un linguaggio usato in contesti molto diversi. Questa eterogeneità nell'uso ha portato alla nascita di diversi linguaggi per interrogare e manipolare XML.
Il linguaggio di trasformazione più noto è XSLT (*eXtensible Stylesheet Language Transformations*), usato principalmente per generare pagine HTML partendo da dati in formato XML. Tra i linguaggi di interrogazione il primo ad essere realmente usato è stato XPath, un linguaggio semplice dal limitato potere espressivo. In parallelo, il mondo relazionale ha prodotto XML/SQL, un'estensione di SQL per i documenti XML. Da ultimo, il W3C sta concludendo il processo di standardizzazione per XQuery, il linguaggio più adatto per le interrogazioni. XQuery e XSLT permettono entrambi di esprimere tutte le possibili interrogazioni e trasformazioni di documenti XML, ma nascono con scopi diversi e ben precisi, e risulta più facile effettuare alcune determinate operazioni con XSLT e altre con XQuery.

4.1. XPath

XPath è il linguaggio base per interrogare dati XML; permette di usare una sintassi simile a quella dei pathname dei file per individuare gli elementi in base alla loro posizione nell'albero che rappresenta il documento (cioè per "navigare" nella struttura, come si dice a volte). Una espressione XPath è una stringa contenente nomi di elementi e operatori di navigazione e selezione, che specificano i "passi" da compiere per raggiungere una particolare porzione del documento a partire dall'elemento radice:

.	indica il nodo corrente
..	passa al nodo padre del nodo corrente
/	passa a un nodo figlio del nodo corrente
//	passa a un discendente del nodo corrente
@	passa a un attributo del nodo corrente
*	indica qualsiasi nodo
[...]	predicato di selezione applicato al nodo
[n]	selesione per posizione (n è un numero)

Per esempio:

```
doc(musica.xml)//catalogo/disco
```

estrae tutti gli elementi <disco> contenuti in elementi <catalogo> del documento musica.xml. È anche possibile selezionare gli elementi da estrarre applicando dei filtri con condizioni sul valore di sottoelementi contenuti negli elementi da estrarre, come in questo esempio

```
doc("musica.xml")//catalogo/disco[artista=
'Bob Dylan']
```

in cui si estraggono gli album di Bob Dylan, o in base a condizioni su sottoelementi di elementi in cui l'elemento di interesse è contenuto. Per esempio, questa interrogazione

doc("musica.xml")//catalogo/disco[artista='L. Cohen']/titolo

estrae solo i titoli degli album di Leonard Cohen.

4.2. XSL/XSLT

XSLT è un linguaggio basato su regole (chiamate *template*) e usato per trasformare un documento XML in un altro documento XML o in un altro tipo di documento (spesso HTML). Più template che concorrono alla trasformazione di una determinata tipologia di documenti sono raccolti in fogli di stile (o *stylesheet*), e si dice che un dato output è il risutato dell'applicazione del foglio di stile al documento in input.
Ogni template XSLT si applica a specifici elementi XML, opportunamente identificati, e produce come risultato altri elementi XML o qualsiasi altro tipo di informazione testuale. Durante la sua esecuzione, un template può anche aggiungere o eliminare elementi, cambiarne l'ordine e filtrare gli elementi oggetto della trasformazione in base al valore dei loro attributi o sottoelementi. Intuitivamente, possiamo immaginare queste trasformazioni come trasformazioni di alberi XML in altri alberi che descrivono la struttura sintattica del risultato (anche se non XML).
Per selezionare gli elementi su cui agiscono, i template utilizzano XPath. Per caratterizzare le parti del documento XML sorgente a cui applicare le trasformazioni, quindi, si possono esprimere sia vincoli di tipo strutturale, individuando elementi in specifiche posizioni, sia vincoli sui valori, considerando solo gli elementi le cui sotto-parti soddisfano opportune condizioni.
Il motore di esecuzione delle regole procede a una scansione ordinata del documento sorgente in base alla gerarchia degli elementi (più precisamente, una visita ricorsiva che esplora di ogni nodo tutti i figli seguendo l'ordine da sinistra a destra dell'albero); quando nella scansione si incontra un elemento XML che soddisfa le condizioni di una regola, quest'ultima produce in output il contenuto specificato al suo interno, ed eventualmente specifica come deve proseguire l'esecuzione. Le parti per cui non viene trovata nessuna corrispondenza con i template vengono attraversate senza che ciò abbia alcun effetto (vedi riquadro 1).
Vediamo ora un semplice esempio. Con riferimento al codice del riquadro 1, dopo il prologo, che specifica che uno stylesheet è a tutti gli effetti un documento XML, l'elemento radice <xsl:stylesheet> definisce l'inizio del foglio di stile, il cui contenuto è rappresentato, in questo caso, da una sola regola, definita nell'elemento **<xsl:template>**.
L'attributo **match** è usato per definire gli elementi a cui il template deve essere applicato; nell'esempio il template si applica alla sola radice del documento (la semplicissima espressione XPath '/' individua, infatti, tale elemento).
La parte rimanente del template (in verde) è la porzione di documento restituita in output; in questo caso si tratta della struttura di una pagina HTML che contiene solo una tabella con una intestazione e nessuna riga. Se visualizzato in un browser, il risultato appare come di seguito.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <tr><th>Album</thv<th>Autore</th><th>Etichetta</th></tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheetv
```

Dati in output

RIQUADRO 1

Album	Autore	Etichetta

Vediamo ora di definire una trasformazione che riempie la tabella con I dati contenuti nel documento precedentemente considerato. Per farlo, ci serviremo della direttiva <xsl:value-of> che seleziona il valore di un

elemento del documento XML sorgente di modo da poterlo inserire nel risultato (vedi riquadro 2).
La direttiva value-of utilizza un attributo **select** per denotare attraverso un'espressione XPath gli elementi di cui produrre in output il contenuto. Nell'esempio qui sopra si estraggono alcuni dati dal secondo album nel catalogo del primo produttore.

Album	Autore	Etichetta
La Città Vecchia	F. De André	Karim

L'esempio presentato funziona correttamente perché estrae solamente un album. Ma se rimuovessimo il selettore di posizione del disco [2] le prime due espressioni XPath restituirebbero tanti dati quanti ve ne sono nel documento, e pertanto i titoli e gli autori sarebbero visualizzati tutti nella stessa casella?

Album	Autore	Etichetta
Nuvole Barocche La Città Vecchia..	Fabrizio De André F. De André	Karim

Per assegnare correttamente titolo, autore e casa discografica a righe diverse della tabella occorre sfruttare un costrutto che definisce una iterazione: <xsl:for-each> itera l'esecuzione di quanto contenuto al suo interno per tutti i dati risultanti dalla valutazione di una espressione XPath, ancora una volta espressa nell'attributo select (vedi riquadro 3).
Questa trasformazione, dopo l'inserimento dell'intestazione della tabella, inserisce per ogni album del catalogo, una riga con una casella col titolo e una col nome dell'artista.

Album	Autore	Etichetta
Nuvole Barocche	Fabrizio De André	Karim
La Città Vecchia	F. De André	Karim

Si noti come le select di direttive contenute in una direttiva più esterna utilizzino espressioni XPath che proseguono la discesa nella gerarchia del documento da dove si era fermata la discesa definita nella direttiva che le contiene. Nell'esempio le select degli elementi che costruiscono le caselle nelle tabelle (**titolo, artista, ../../nome**) sono la prosecuzione della path expression **produttore/catalogo/disco**. Si noti inoltre che per includere l'etichetta nella tabella sono necessari due passi "verso l'alto" (realizzati con l'operatore ".." che accede al nodo padre del nodo corrente), fino all'elemento <produttore>, da cui poi ridiscendere per recuperarne il nome.
Gli elementi su cui iterare l'applicazione di una trasformazione possono essere filtrati per considerarne solo un sottoinsieme; a questo scopo si sfruttano i predicati ammessi nelle espressioni XPath. Per esempio l'aggiunta di un predicato all'attributo select permette di aggiungere alla tabella solo le righe relative agli album di Ray Charles:

```
<xsl:template match="/">
  <html>
    <body>
      <table border="1">
      <tr> <th> Album</th> <th> Autore </th> <th> Etichetta </th> </tr>
      <tr><td><xsl:value-of
select="produttore[1]//disco[2]/titolo" /></td>
          <td><xsl:value-of
select="produttore[1]//disco[2]/artista"/></td>
          <td><xsl:value-of select="produttore[1]/nome" /></td> </tr>
    </table>
    </body>
  </html>
</xsl:template>
```

RIQUADRO 2

```
<xsl:template match="/">
  <html>
    <body>
      <table border="1">
        <tr><th>Album</th><th>Autore</th><th>Etichetta</th></tr>
        <xsl:for-each select="produttore/catalogo/disco">
          <tr> <td><xsl:value-of select="titolo"/></td>
             <td><xsl:value-of select="artista"/></td>
             <td><xsl:value-of select="../../nome"/></td> </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
</xsl:template>
```

RIQUADRO 3

<xsl:for-each select="produttore/catalogo/disco[artista='Ray Charles']">

Oltre a selezionare in base al valore di un certo sottoelemento, è possibile, con la direttiva <xsl:sort/>, ordinare gli elementi in base al valore di uno dei loro sottoelementi, indicato come al solito attraverso una espressione XPath inserita nell'attributo select. Per esempio si può ottenere un ordinamento degli album in base all'ordine alfabetico dei loro autori con la seguente modifica:

```
<xsl:for-each select="produttore/catalogo/disco">
  <xsl:sort select="artista"/>
  <tr> <td><xsl:value-of select="titolo"/></td>
       <td><xsl:value-of select="artista"/></td> </tr>
</xsl:for-each>
```

XSLT offre anche caratteristiche mutuate dai linguaggi di programmazione, come la possibilità di effettuare test condizionali. In questo ultimo semplice esempio:

```
<xsl:if test="prezzo != 10">
   ...trasformazione...
</xsl:if>
```

si vincola l'applicazione di una trasformazione al fatto che il prezzo sia diverso da 10.

4.3. XQuery

XQuery è il linguaggio standard per interrogare documenti XML. Il nucleo di XQuery è rappresentato dalle *flwor expression*, il costrutto più versatile del linguaggio. Ogni interrogazione è costruita con espressioni che possono essere annidate le une dentro le altre; la costruzione di una flwor expression può essere assimilata a quella di una classica espressione select-from-where dell'SQL. Inoltre, XQuery supporta la costruzione di nuovi frammenti XML di arbitraria complessità, sia ex-novo sia ricombinando arbitrariamente i frammenti estratti.
Una flwor expression si compone di cinque clausole (*flwor* deriva dalle loro iniziali):

- **for** per iterare l'esecuzione in base a sequenze di valori
- **let** per legare singole variabili a intere sequenze di valori
- **where** per esprimere predicati
- **order by** per imporre un ordinamento al risultato
- **return** per costruire il risultato

La return e una almeno tra for e let sono necessarie, le altre sono opzionali. Esempio:

```
for $cd in doc ("musica.xml")//disco
return $cd
```

La clausola **for** valuta l'espressione che definisce la variabile $cd, ottenendo un insieme di elementi, e itera all'interno di tale insieme, assegnando alla variabile $cd un riferimento ad un diverso elemento dell'insieme per ogni iterazione.
La clausola **return** costruisce il risultato dell'interrogazione. L'esempio restituisce semplicemente l'insieme dei valori di $cd, cioè di tutti i dischi che si trovano nel documento "musica.xml"
Le clausole **let** consentono di introdurre variabili legate a insiemi di valori:

```
let $tuttiCD := doc("musica.xml")//disco
return count( $tuttiCD )
```

La clausola **let** valuta l'espressione (//disco) e assegna *l'intero insieme* dei dischi alla variabile $tuttiCD. La valutazione di una clausola **let** non effettua alcuna iterazione, ma genera un singolo assegnamento della variabile all'intero insieme restituito dall'espressione, passato poi nella return alla funzione che ne conta gli elementi.
La clausola **where** esprime una condizione: solamente gli assegnamenti (operati da for e let) che la soddisfano sono utilizzati dalla clausola **return**. Le condizioni nella clausola **where** possono contenere diversi predicati in AND o in OR. Per esempio:

```
for $cd in doc("musica.xml")//disco
where $cd/artista="F. De André" and
$cd/@anno<"1980"
return $cd
```

restituisce tutti i dischi di De André antecedenti al 1980. Si noti che @anno è preceduto dall'operatore XPath '@' poiché è un attributo.
La clausola **return** definisce l'output di una flwor expression, e può essere un nodo, una

sequenza ordinata di nodi o un valore atomico. Può contenere costruttori di elementi, riferimenti a variabili definite nelle clausole for e let ed espressioni annidate. Un costruttore di elemento consta di un tag iniziale e di un tag finale che racchiudono una lista di espressioni che determinano il contenuto di tale elemento:

```
<ElencoAlbumEurope>
{ for $pro in doc("musica.xml")//produttore
  let $cdEur := $pro/disco[artista="Europe"]
  return  <CasaDiscografica>
                { $pro/nome,
                  $cdEur/titolo }
            </CasaDiscografica>
}
</ElencoAlbumEurope>
```

In questo caso, all'interno di un unico elemento esterno ElencoAlbumEurope, per ogni produttore viene generato un elemento CasaDiscografica contenente il nome del produttore e un elenco dei titoli degli album degli Europe tutti i titoli degli album estratti (selezionati in base all'artista) sono inseriti in un unico elemento.
La clausola **order by** ordina i frammenti estratti in base a uno o più sottoelementi.
Per esempio questa interrogazione

```
for $cd in doc("musica.xml")//disco
order by $cd/titolo
return <cd>
            { $cd/titolo,
              $cd/artista }
        </cd>
```

estrae i dischi ordinati rispetto al titolo.
Oltre ai costrutti delle FLWOR Expression, XQuery contiene anche un ricco insieme di funzioni predefinite, e offre la possibilità di definire nuove funzioni, in cui si possono usare tutti i costrutti classici della programmazione (if-then-else, cicli while, cicli for,...).

4.4. SQL/XML

SQL/XML (www.sqlx.org) è l'estensione di SQL (parte di ANSI/ISO SQL 2003) per XML. Mentre XQuery è XML-centrico, SQL/XML è SQL-centrico. Esso permette di creare strutture XML con poche e potenti funzioni dedicate. SQL/XML è più semplice da imparare per un utente SQL e può sfruttare gli strumenti a disposizione per il mondo relazionale, ma XQuery, linguaggio nato appositamente per i dati semi-strutturati, si adatta meglio alle peculiarità del diverso modello dei dati.

4.5. XQBE (XQuery By Example)

XQBE (*XQuery By Example*) è un dialetto grafico di XQuery che includiamo in questa rassegna anche perché sviluppato dagli autori presso il Politecnico di Milano. XQBE si ispira a QBE (*Query By Example*), l'interfaccia grafica per la formulazione di semplici interrogazioni SQL di MS Access®. XQBE è stato progettato per essere intuitivo e semplice da mappare su XQuery, ed essere quindi un'interfaccia grafica capace di appoggiarsi sui vari motori XQuery esistenti.
Il successo ottenuto da QBE dimostra che un'interfaccia visuale per un linguaggio di interrogazione è efficace quando è basata su un'astrazione grafica del modello dei dati da interrogare. Rispettando tale principio, XQBE è basato su strutture grafiche ad albero con annotazioni, che rispecchiano la natura gerarchica di XML e in cui gli *elementi* sono rappresentati da rettangoli, gli *attributi* da pallini neri, i PCDATA da pallini bianchi.
XQBE permette di costruire flwor expression annidate, di inserire nuovi elementi XML e di ristrutturare documenti esistenti. Tuttavia il potere espressivo di XQBE è limitato rispetto a quello di XQuery. Per esempio non è supportata la definizione di funzioni da parte dell'utente o l'uso della disgiunzione. Queste limitazioni sono precise scelte progettuali, ritenute necessarie per ottenere un linguaggio sufficientemente semplice.
Nella figura 2 vediamo l'interrogazione XQBE che estrae tutti i dischi di Fabrizio De André precedenti al 1990, mostrandone l'anno di pubblicazione e il titolo.
L'interrogazione è divisa in due parti: la parte sinistra descrive quali condizioni vogliamo porre sui dati da estrarre, la parte destra, il documento da costruire. Nell'esempio la parte sinistra estrae tutti gli elementi

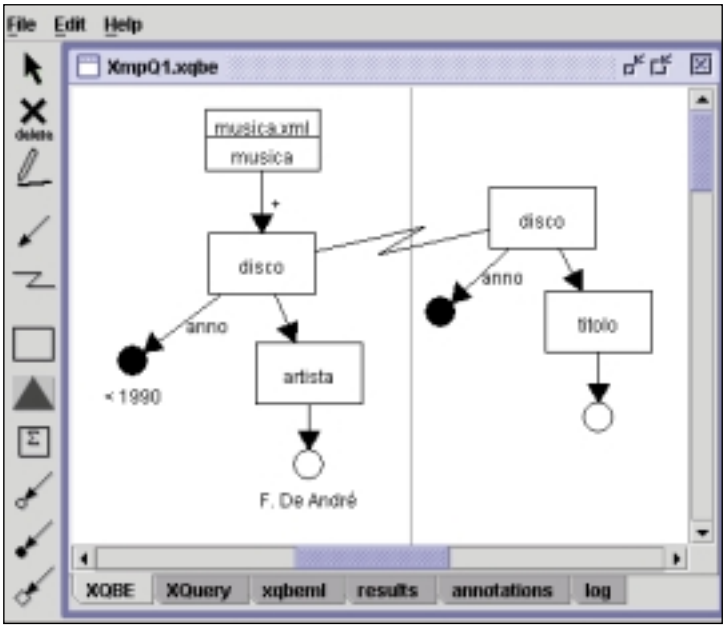

FIGURA 2
Interfaccia grafica XQBE

"disco" che siano discendenti dell' elemento iniziale (radice) "musica", che contengano un sotto-elemento "artista" il cui valore è "F. De André" e con un attributo "anno" minore di 1990. I dischi così selezionati sono "trasportati" per mezzo di un *binding edge* (collegamento a forma di fulmine) nella parte destra, dove sono utilizzati per costruire il risultato: nell'esempio se ne trattengono solo l'attributo "anno" e il sotto-elemento "titolo". Qui di seguito, la stessa interrogazione espressa in XQuery:

```
for $d in doc("musica.xml")/musica//disco
where $d/artista = "F. De André" and $d/@anno < 1990
return <disco anno="{ $d/@anno }">
            { $d/titolo }
        </disco>
```

5. LO STATO DELL'ARTE DELLA TECNOLOGIA

Gli strumenti per estrarre informazioni da fonti XML si possono classificare in due tipologie:
1. *Sistemi di pura interrogazione*: si limitano a caricare in memoria centrale i documenti dal file system e a eseguire interrogazioni o trasformazioni su di essi.
2. *Sistemi che offrono un supporto persistente*: hanno un proprio sistema di gestione della memoria (sia volatile sia persistente) e di indicizzazione, che aumentano notevolmente l'efficienza delle interrogazioni e permettono di manipolare documenti molto più grandi di quanto permettano i primi.

5.1. Sistemi di pura interrogazione

Sax e Dom
Le tecnologie più usate dai linguaggi di programmazione sono **DOM** (*Document Object Model*) e **SAX** (*Simple Api for Xml*). Queste due tecnologie offrono ai programmatori alcune potenti astrazioni per realizzare in modo semplice il codice che naviga o modifica dati in formato XML (arrivino da file, dalla rete o siano creati *ad hoc* all'interno dell'aplicazione).
Le differenze tra queste due tecnologie sono poche ma importanti, specialmente dal punto di vista delle prestazioni. DOM è un modello a oggetti di facile utilizzo, basato su classi che creano in memoria una rappresentazione ad albero del documento XML. SAX, invece, è una tecnologia più "leggera", basata su eventi e priva di un proprio modello a oggetti; questo significa che richiede di creare un "parser" *ad hoc* per processare ogni documento XML, ma si tratta anche della soluzione ideale per trattare documenti di grosse dimensioni.

IPSI-XQ
L'IPSI XQuery Demonstrator (IPSI-XQ) è un prototipo che implementa XQuery secondo le specifiche del W3C. Le interrogazioni vengono eseguite precaricando in memoria centrale il documento, quindi è uno strumento poco efficiente per interrogare documenti di grandi dimensioni; è però un ottimo strumento per imparare XQuery.

5.2. Sistemi che offrono un supporto persistente

SQL Server 2005 (Yukon)
Yukon, l'edizione di Microsoft© SQL Server che sarà disponibile a breve (2005), introduce il supporto ad XQuery per interrogare i dati memorizzati in colonne con tipo XML. Per agevolare la scrittura di query, in MS SQL Server Workbench è stata introdotta una finestra di progettazione dedicata a

XQuery, *XQuery Designer*, per creare query semplicemente selezionando e trascinando i nodi XML dalla rappresentazione della struttura dei dati. L'editor di testo può sempre essere utilizzato per aggiungere costrutti XQuery complessi. È possibile utilizzare XQuery Designer per creare query da inserire poi all'interno di istruzioni select o per la creazione di report.

Oracle
Oracle gestisce i dati in formato XML attraverso XSU (XML/SQL Utility), una interfaccia per java e PL/SQL (linguaggio di programmazione proprietario di Oracle). Grazie a XSU, i documenti XML dotati di struttura regolare possono essere inseriti in una o più tabelle. Tutto ciò è facilitato dalla tecnologia object/relational di Oracle, in cui una tabella può contenere altre tabelle annidate; ciò permette di mappare direttamente l'annidamento gerarchico dei documenti XML.

DB2
DB2 gestisce i dati in formato XML sfruttando un file DAD (*Data Access Definition*), a sua volta in formato XML, in cui viene specificata la mappatura dei documenti in input: un documento XML può essere inserito in una o più tabelle utilizzando una funzione che accetta come parametri il documento da processare e un DAD, che indica la colonna di destinazione per ogni elemento XML.

Tamino Software AG
Tamino XML Server di software AG consente l'archiviazione, la gestione e la pubblicazione di documenti XML. Esso offre le capacità di storage nativo XML e di interrogazione dei dati. Inoltre, il prodotto offre servizi cosiddetti "di classe enterprise" (sicurezza, affidabilità, clustering, fault tolerance ecc...). Lo storage service è in grado di immagazzinare dati sia in formato XML che non-XML (per esempio in formato grafico o video), abilitando XML Server anche come soluzione di gestione dei contenuti. Il motore di interrogazione estende XPath e supporta XQuery.

Xyleme
Xyleme, nato nel 1999 per iniziativa del gruppo Verso dell'INRIA, usa le proprietà strutturali di XML per fornire maggiore precisione alla ricerca di informazioni sul Web. I documenti XML sono letti e memorizzati localmente in un data warehouse su cui sono disponibili servizi per l'acquisizione e la memorizzazione di documenti XML, la classificazione semantica, la gestione temporale delle versioni, l'interrogazione con sofisticate funzioni di ricerca nel testo e la gestione a eventi dei contenuti.

eXist
eXist (http://exist.sourceforge.net) è un database XML nativo open source capace di eseguire interrogazioni XQuery in modo efficiente sfruttando una opportuna indicizzazione dei documenti. eXist supporta anche XUpdate, un linguaggio di modifica di dati in formato XML. Il sistema supporta XQuery nella versione del novembre 2003, ad eccezione del supporto per XML Schema.

6. CONCLUSIONI

XML, nato nel 1996 con lo scopo di offrire un metodo semplice, ma generale e rigorosamente formalizzato, per separare il contenuto informativo di un documento dalla sua formattazione, si è presto diffuso anche come linguaggio per rappresentare i dati internamente alle applicazioni informatiche, oltre che come standard per lo scambio di informazioni tra applicazioni che si trovano a cooperare attraverso il Web.
Questo articolo ha dapprima contestualizzato XML da un punto di vista "storico", motivandone la nascita e illustrando il percorso che ha portato alla sua definizione. Successivamente ha offerto una rapida panoramica su alcuni dei molteplici scenari d'uso di XML, e ha poi presentato le principali caratteristiche dei suoi linguaggi di interrogazione, soffermandosi in particolare su XPath, XSLT e XQuery. In conclusione, per dare uno sguardo anche allo stato dell'arte della tecnologia, si sono illustrate le principali caratteristiche di alcuni sistemi di gestione di basi di dati che oggi supportano XML e XQuery.
È opinione degli autori che nei prossimi anni XML e XQuery diverranno sempre più protagonisti dell'evoluzione della rappresentazione e

gestione dei dati, e saranno una parte sempre più rilevante del bagaglio culturale dei professionisti della società dell'informazione.

Bibliografia

[1] XML: http://www.w3.org/XML/

[2] XMLSchema: http://www.w3.org/XML/Schema

[3] Semantic Web: http://www.w3.org/2001/sw/

[4] SOAP: http://www.w3.org/2000/xp/Group/

[5] XQBE: http://dbgroup.elet.polimi.it/xquery/XQBE.html

[6] XQuery: http://www.w3.org/XML/Query

[7] XSLT: http://www.w3.org/Style/XSL/ù

Daniele Braga laureato in Ingegneria Informatica al Politecnico di Milano nel 2001, collabora col gruppo di Basi di Dati all'attività di ricerca sui linguaggi di interrogazione per dati semi-strutturati; sta terminando il dottorato di ricerca in Ingegneria dell'Informazione e partecipa all'attività didattica nell'ambito di vari corsi.
daniele.braga@polimi.it

Alessandro Campi è ricercatore del Dipartimento di Elettronica e Informazione del Politecnico di Milano. Vi ha conseguito nel 2004 il Dottorato di Ricerca in Ingegneria dell'Informazione. I suoi interessi riguardano i linguaggi di interrogazione visuali per dati semi-strutturati. È titolare del corso di Basi di Dati per Ingegneria delle Telecomunicazioni.
alessandro.campi@polimi.it

Stefano Ceri professore ordinario del Dipartimento di Elettronica e Informazione al Politecnico di Milano, è stato anche professore-visitatore all'Università di Stanford. I suoi interessi più recenti riguardano l'uso di XML per la gestione dei dati e i metodi e gli strumenti per la progettazione di applicazioni Web. È autore di molte pubblicazioni internazionali, tra cui otto libri e circa duecento articoli su riviste e atti di convegni.
stefano.ceri@polimi.it