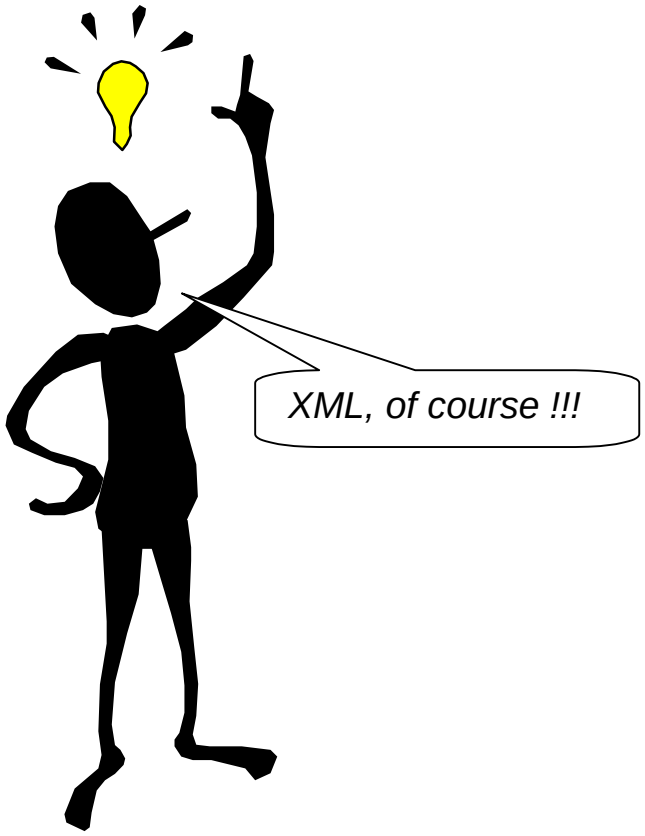
Marco Gambini Mat.214854417 A.A.2000/2001



La rappresentazione dei dati in XML

Relazione di Sistemi Informativi II

Sommario

Abstract

La comunità Internet sta riservando un enorme quantità di energia, di denaro e di sforzo nello sviluppo di un pacchetto estensivo di standard correlati ed incentrati su XML (Extensible Markup Language), il linguaggio emergente per la rappresentazione e la distribuzione di documenti sul Web.
Come conseguenza alla diffusione di XML e degli standard ad esso collegati è in atto un sostanziale incremento della quantità di dati codificati in tale forma. Al fine di ottimizzare la descrizione delle strutture e dei vincoli che li caratterizzano, le istituzioni accademiche, i colossi industriali legati ad Internet ed il W3C hanno proposto numerosi meta linguaggi. Analogamente, sono state avanzate o sono in corso di definizione svariate proposte per l'interrogazione, l'aggiornamento e la trasformazione dei dati in XML. Internamente al W3C è tuttora in corso un dibattito sulla definizione di uno standard per i meta linguaggi che descrivono la struttura dati e per gli altrettanto significativi linguaggi di interrogazione.
Questo lavoro offre una panoramica sullo stato dell'arte dei meta linguaggi descrittivi, tentando un'analisi comparativa delle loro principali caratteristiche ed una conseguente classificazione.
Si desidera poi approfondire un aspetto specifico all'interno di questo scenario, ovvero il problema dello scambio (o del semplice confronto) di informazioni XML descritte mediante strutture dati eterogenee.
Verrà infine analizzata una recente proposta di soluzione per tale problema: la definizione di un framework che consente, attraverso un meta modello ed un insieme di meta primitive, la produzione di documenti XML da una forma ad un'altra.

Introduzione ad XML

Dall' HTML all'XML

Come conseguenza della crescita esponenziale del Web registratasi negli ultimi dieci anni gli utenti hanno scoperto un numero sempre maggiore di metodi per comunicare. L'HTML (Hypertext Markup Language) ha rappresentato il fondamento di questo interscambio ed è stato utilizzato per rappresentare di tutto, dai testi accademici alla poesia, dai glossari ai cataloghi in linea.
La struttura delle pagine Web basate su HTML non dice tuttavia molto sul loro reale contenuto. Vediamo un semplice esempio rappresentato da una lista di definizione:

```
<!--Price list for individual fruit -->
<dl>
        <! --Fruit-->
        <dt> Apples </dt>
                <! --Price-->
                <dd> $1 </dd>
        <! --Fruit-->
        <dt> Oranges </dt>
                <! --Price-->
                <dd> $2 </dd>
</dl>
```

E' facile notare come la maggior parte dei tag offerti da HTML sia in grado di influenzare soltanto il layout e la presentazione del testo sulla pagina: questa lista non contiene definizioni di elementi ed è usata solamente per organizzare frutta e prezzi in un particolare modo. Inoltre queste informazioni sul layout non sono flessibili in quanto i tag vengono spesso usati per presentare le informazioni in modalità che estendono o violano il significato dei tag stessi.
La comunità informatica ha operato molti tentativi per imporre un ordine più flessibile ai dati contenuti nelle pagine. Molte di queste soluzioni proprietarie, quali ASP o Cold Fusion, hanno ottenuto un discreto successo, ma la maggior parte di esse è fautrice di approcci particolari che necessitano di modifiche per ogni nuovo spazio dei problemi. Spesso inoltre il loro impiego richiede investimenti ingenti, dovuti all'utilizzo ad hoc di hardware e software lato server.
XML e gli standard ad esso correlati consentono di sostituire o di estendere i sistemi di tag proprietari con linguaggi indipendenti dalla piattaforma che si adattano esattamente ai singoli spazi di lavoro. Inoltre evitano l'inserimento di tag speciali e di commenti atti a specificare il significato di un particolare campo in quanto rendono il campo stesso significativo sia per le applicazioni che per gli esseri umani. L'elenco dell'esempio precedente potrebbe essere trasformato in:

```
<FruitPriceList>
        <Fruit> Apples </Fruit>
                <Price> $1 </Price>
        <Fruit> Oranges </Fruit>
                <Price> $1 </Price>
</FruitPriceList>
```

Non soltanto le informazioni risultano meno confuse e presentate in maniera più chiara ma i campi sono anche identificabili da parte di un motore di ricerca: le mele da mangiare possono perciò essere facilmente distinte dalla Grande Mela!

XML permette inoltre di mantenere informazioni sul tipo di dato immesso in ogni campo mentre HTML consente di identificare soltanto una mezza dozzina di tipi di dati: abbreviazioni, acronimi, indirizzi, citazioni e variabili, che oltretutto sono più usati per influenzare la formattazione piuttosto che per identificare un campo logico.
In buona sostanza i tag di HTML definiscono un semplice schema del documento associando testo e grafica mediante intestazioni, paragrafi, liste, tabelle e illustrazioni. Ogni altro elemento che compone un testo (i calcoli, le categorizzazioni, la comprensione profonda della struttura dell'argomento) viene totalmente ignorato e lasciato unicamente alle indicazioni che derivano dai collegamenti ipertestuali e dai documenti ad esso collegati. I contenuti del Web sono stati costruiti sulla base di queste regole incapaci di cogliere la profonda differenza fra testo e dati: questi ultimi infatti possiedono una struttura ed un contesto. Mentre gli esseri umani sono in grado di riconoscere od estrapolare la struttura ed il contesto dei dati attraverso suggerimenti visuali, le macchine non ragionano in questo modo. Il principale punto di forza di XML è stata l'adozione di una filosofia capace di consentire ad una macchina l'interpretazione delle strutture dati: ciò si ottiene identificando in modo non ambiguo ogni suddivisione significativa di un documento come parte di una struttura ad albero. Un'intera automobile può così essere descritta come un completo elenco di parti suddivise in liste di componenti, dal motore alle minuterie metalliche. Analogamente un libro può essere descritto come l'insieme dei capitoli, dei paragrafi, delle note a piè di pagina, delle illustrazioni che lo compongono e così via.

Potenzialità e limiti dell'XML

Pur essendo sbalorditivo il paradigma di XML si scontra con alcuni limiti, primo dei quali l'incapacità di catturare le strutture dati non ad albero. Molte idee astratte ad esempio non possono essere rappresentate come strutture ad albero: il concetto di "onore" può essere chiaro ad una persona ma non lo si può suddividere in componenti, anche se esistono concetti correlati che formano quello di "onore". Un secondo problema è quello dell'esclusione, ovvero la certezza che un oggetto non possa apparire in modo non appropriato come uno dei propri discendenti: un'automobile normalmente possiede un solo motore ed un secondo motore in genere non è una parte del primo. Un libro invece potrebbe contenere riferimenti ad altri libri, potrebbe far parte di una collana, oppure potrebbe essere suddiviso in più volumi che a loro volta possono fare riferimento al libro originale. In altre parole è necessario garantire che l'oggetto rappresentato possieda una struttura ad albero, caratterizzata da una singola radice e da un insieme di rami privo di cicli completi. Mentre un'automobile è quasi interamente rappresentata dalla somma delle proprie parti ed il suo comportamento è ben definito dai collegamenti fra i componenti, un libro possiede una struttura più simile ad un grafo che non ad un albero, anche se esiste la possibilità di descriverlo in maniera sequenziale. È abbastanza semplice riconoscere in un bullone una foglia terminale dell'albero relativo ad un'automobile, ma non è altrettanto immediato comprendere se una nota a piè di pagina relativa ad un paragrafo rappresenti un elemento terminale o possa contenere al suo interno diversi paragrafi. XML deve adottare opportune strategie per evitare di inserire elementi figli dove questi non dovrebbero comparire: talvolta però anche le migliori strategie falliscono e non resta che affidarsi al buon senso dell'utente.
Si potrebbe pensare al problema dell'esclusione come ad una debolezza di XML; in realtà occorre considerare la necessità di raggiungere un giusto compromesso fra la potenza espressiva del linguaggio, la sua facilità d'uso e quella di implementazione. Poiché la perfezione descrittiva è teoricamente impossibile (teorema di Goedel) la scelta di un livello di imperfezione rispetto ad un altro è un criterio di progettazione. XML è stato progettato per essere utile e non costoso: come qualunque altro costrutto umano è un compromesso tra il desiderio di perfezione e la realtà.
Un'altra limitazione che caratterizza XML è il supporto non completo della tecnologia "ad oggetti". Come prova del fatto che i documenti XML non costituiscono degli oggetti si consideri che essi non

possono realmente ereditare la struttura dai loro antenati e non sono incapsulati, tanto che i dati al loro interno sono completamente esposti. Essi tuttavia esibiscono il polimorfismo ed altri comportamenti orientati agli oggetti, rappresentando quindi un passo avanti in questa direzione.
A fronte dei limiti elencati XML definisce nuove possibilità di sviluppo del Web, in virtù di una serie di potenti funzionalità:

1) Miglioramento della precisione:

XML consente di descrivere un documento in un modo che possa essere "compreso" da una macchina. I tag XML descrittivi quali **<fruit>** e **<price>** sono molto più significativi per le macchine rispetto agli anonimi tag di formattazione disponibili in precedenza con HTML. XML fornisce inoltre un meccanismo (la definizione del tipo di documento) che consente di condividere con chiunque la conoscenza sulla struttura dei dati.

2) Convalida della struttura del documento:

XML consente di forzare la convalida della struttura del documento: è possibile far rispettare la presenza di alcuni elementi rendendone altri facoltativi e collegare una struttura ad un'altra. In altre parole, scegliendo di includere l'elemento ABC si può forzare l'introduzione di un'istanza associata all'elemento XYZ. Alternativamente, se ABC e già presente, allora XYZ può essere omesso.

3) Flessibilità del layout:

Attraverso l'uso dei fogli di stile XML rende possibile l'effettiva variazione della presentazione dei documenti in funzione dell'uso che se ne intende fare. Uno stesso insieme di dati può essere presentato in maniera diversa (ad esempio come lista o come tabella) a seconda dello scopo e del contesto.

4) Robustezza ed indipendenza dalla piattaforma:

La robustezza e l'indipendenza di XML non hanno eguali in nessun altro meccanismo di trasporto dei dati o di elaborazione distribuita. Poiché XML descrive un database flat-file ogni applicazione che supporta un qualunque accesso ai database può utilizzare XML come minimo comun denominatore per il trasporto dei dati, generando e traducendo da XML ed utilizzando internamente formati normalizzati o proprietari. Inoltre, la possibilità di comporre i testi in formato ASCII puro conferisce ai documenti una considerevole resistenza ai danni. La rimozione di byte, anche di una lunga sequenza, non danneggia in modo significativo la parte di testo rimanente. Questo aspetto è in netto contrasto con molti altri formati, quali i dati compressi o gli oggetti Java serializzati, dove il danneggiamento o la perdita anche di un solo byte può rendere illeggibile il resto del file.

5) Progettazione di documenti ad oggetti:

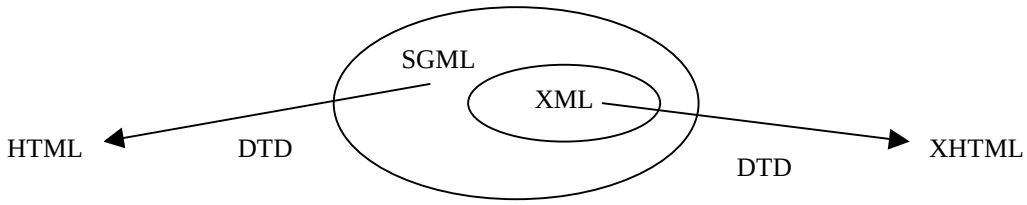
XML si sta sviluppando in modo da supportare i metodi di progettazione e di programmazione ad oggetti. Iniziative quali SOX (Schema for Object-Oriented XML) od altre equivalenti forniranno un completo accesso ad oggetti relativamente agli elementi XML.

In estrema sintesi si può affermare che se HTML ha trasformato Internet nella biblioteca del mondo, XML e gli standard ad esso collegati si pongono come obiettivo la trasformazione di Internet nel centro commerciale e finanziario del mondo.

Una lunga storia

La scommessa su XML sembra rivelarsi vincente anche grazie al fatto che le idee in essa presenti sono in realtà molto vecchie e la loro correttezza è stata dimostrata attraverso migliaia di progetti.
Tentiamo allora di chiarirne meglio le origini: XML viene definito come un sottoinsieme proprio di SGML, nel senso che a quest'ultimo sono stati tolti alcuni elementi per rendere il nuovo linguaggio più semplice da analizzare, da comprendere e da usare. SGML è un meta linguaggio utilizzato per descrivere linguaggi applicativi: anche se possiede molte utili caratteristiche la complessità intrinseca lo rende estremamente difficile da imparare e da utilizzare. Anche se usato da molte grandi aziende e da organizzazioni governative SGML comporta spese e difficoltà di apprendimento tali da rendere difficile per i profani e per le piccole società l'accesso alla grande potenza dei linguaggi per la formattazione dei dati strutturati. Il duplice obiettivo di XML è quello di rendere possibile da un lato l'accesso ai non specialisti a gran parte della potenza di SGML, dall'altro quello di creare stabili implementazioni di un linguaggio di descrizione dei documenti strutturati, in modo che il costo degli editor, degli strumenti di convalida e degli altri strumenti di produzione si mantenga ad un livello ragionevole.
Poiché XML nasce dallo snellimento di SGML, un buon modo per seguirne la storia è quello di ripercorrere le tappe che hanno portato all'affermazione di SGML. Entrambi questi linguaggi sono stati concepiti nell'ambito dello sviluppo di sistemi in grado di automatizzare il processo editoriale. Le prime applicazioni in questo campo consentivano all'autore di inserire nella macchina il documento e la descrizione della formattazione desiderata. Il file contenente l'insieme del testo effettivo e della formattazione prese il nome di **rendition** (letteralmente "interpretazione") in quanto i dati dovevano essere interpretati dal software prima di essere visualizzati o stampati. Il risultato dell'interpretazione venne definito **presentazione**, sottintendendo con questo il processo di conversione in un testo comprensibile da parte di un qualunque essere umano. L'evoluzione dei sistemi di battitura dei testi ha portato a quello che oggi va sotto il nome di *desktop publishing*: i programmi più recenti sono caratterizzati da interfacce utente in grado di rendere a video presentazioni realistiche dei documenti editati. Questa feature è chiamata WYSIWYG (What You See Is What You Get) e trae origine dal fatto che un'interpretazione a sé stante non descrive efficacemente la presentazione associata.
La notazione utilizzata per l'impaginazione dei documenti che può essere considerata antenata del WYSIWYG (e che è ancora in uso al giorno d'oggi) è chiamata **markup di formattazione**. Secondo questa convenzione il testo viene formattato racchiudendolo fra istruzioni chiamate tag (od etichette) che ne specificano l'impaginazione. Poiché il markup utilizza soltanto i normali caratteri presenti su tutte le tastiere, i documenti possono essere creati utilizzando qualunque editor di testi anziché particolari programmi di elaborazione. Il markup di formattazione è una tecnica adeguata qualora l'unico obiettivo sia l'inserimento dei documenti, la descrizione della loro impaginazione ed eventualmente la loro stampa. La crescente necessità di compiere operazioni di ricerca e gestione delle informazioni contenute nei documenti ha spinto allora all'introduzione di uno strumento più potente: il **markup generalizzato**. In altri termini ci si è resi conto che per consentire ai computer di elaborare il testo svolgendo una quantità di operazioni molto maggiore di quella ottenibile con tecnologie WYSIWYG o semplici markup di formattazione è necessario fornire loro una grande quantità di informazioni sul documento. Con questo stratagemma si è tentato di aggirare l'ostacolo della non comprensione del testo da parte dell'elaboratore: si tratta come detto di un espediente, sufficiente però per consentirne una "comprensione" alla stregua dei semplici dati o dei numeri decimali. Il modo più efficiente per "istruire" l'elaboratore alla comprensione del testo è quello di ricorrere ad un'astrazione: anziché utilizzare i tag per descrivere diversi tipi di formattazione, essi vengono impiegati per descrivere il ruolo logico degli elementi associati. Lo scopo finale è quello di rendere l'astrazione

sufficientemente precisa: in questo modo le operazioni da effettuare sul documento (come la stampa, la ricerca e così via) possono essere completamente automatizzate, facendo lavorare il computer sui singoli elementi. Per garantire poi una buona qualità alla stampa od alla visualizzazione dei documenti si è pensato di indicare all'elaboratore come generare le impaginazioni partendo dalla rappresentazione astratta dei dati. A tal scopo sono stati introdotti i **fogli di stile**, delle raccolte di regole con cui esprimere le convenzioni di formattazione. Attraverso un foglio di stile un elaboratore è in grado di convertire automaticamente l'astrazione di un documento in un'interpretazione formattata. I documenti facenti uso di markup generalizzato non richiederanno alcuna riscrittura per sfruttare al meglio le nuove tecnologie: al più sarà sufficiente la creazione di ulteriori fogli di stile per visualizzare i documenti secondo i nuovi formati.
I linguaggi di markup generalizzato devono sottostare anche a precisi vincoli di validità: in altre parole necessitano di una specifica formale che imponga appropriate restrizioni sugli elementi e sulla struttura del documento. A questo proposito i linguaggi offrono la possibilità di utilizzare dizionari personalizzati di elementi, in modo da specificare vincoli appropriati per ogni singolo tipo di documento. In genere col termine "tipo di documento" si intende sia il dizionario utilizzato sia i vincoli sul suo utilizzo. Una volta completato un tipo di documento si può utilizzare un foglio di stile per istruire l'elaboratore in merito alla visualizzazione od alla stampa dei documenti conformi a tale tipo.
Queste intuizioni hanno portato alla nascita nel 1974 dello **Standard Generalized Markup Language** (SGML) ed alla sua successiva affermazione come standard nel 1986. Quando pochi anni più tardi Tim Berners Lee ha proposto la condivisione delle informazioni mediante l'uso di documenti ipertestuali, si è basato proprio su di un semplice tipo di documento SGML per sviluppare una versione ipertestuale chiamata **HyperText Markup Language** (HTML). Pur ereditando dall'SGML alcuni punti di forza, l'HTML fa uso di un prefissato insieme di elementi: questa mancanza di estensibilità ne impedisce una personalizzazione per particolari tipi di documenti. L'intervento del W3C (World Wide Web Consortium) ha sollecitato allora lo sviluppo di un sottoinsieme di SGML capace di raccoglierne le migliori caratteristiche senza tralasciare la semplicità del Web ipertestuale. Il comitato ha deciso di dare al nuovo linguaggio il nome di **Extensible Markup Language** (XML) e di creare per esso nuovi standard riguardanti i collegamenti ipertestuali ed i fogli di stile (Xlink e XSLT).



Questo schema mostra le relazioni esistenti fra SGML, XML e HTML: si noti la differenza tra XML, definito come sottoinsieme di SGML, ed HTML, definito come una sua produzione (così come XHTML è una produzione di XML). In particolare XHTML è progettato per la creazione di pagine Web XML che possano essere visualizzate con i normali browser HTML, senza nel frattempo rinunciare ai vantaggi offerti da XML. In tal modo è possibile riutilizzare o convertire molti milioni di pagine Web HTML già esistenti mantenendone la compatibilità con i browser attuali ed abilitando le nuove caratteristiche per coloro che usano agenti XML compliant.

Documenti validi e ben formati

XML ha una natura duplice essendo costituito da due meta-linguaggi entrambi descritti nello stesso documento. Il primo è un insieme di regole per la produzione di documenti XML, mentre il secondo è un insieme di regole per la produzione di un DTD (Document Type Definition), una sorta di vocabolario che definisce la struttura del documento, permette di esprimere dei vincoli sui suoi contenuti e ne consente una convalida nei confronti di tali vincoli. Va sottolineato che il DTD è un concetto e come tale è esprimibile attraverso molteplici sintassi: spesso tuttavia si utilizza il termine DTD intendendo le effettive dichiarazioni dei markup.
L'utilizzo dei linguaggi di descrizione delle strutture dati (DTD, XML-Schema ecc.), sebbene non obbligatorio, consente una convalida rigorosa della forma e della sintassi dei costrutti XML, indispensabile qualora si desideri controllare il contenuto dei singoli campi. Grazie ad essi è possibile definire le parti facoltative e quelle obbligatorie all'interno di ciascun documento, verificando poi l'effettiva presenza di queste ultime. Un secondo vantaggio discende dalla possibilità di condividere la struttura del documento e le informazioni ivi racchiuse con un pubblico molto ampio. Non è più necessario infatti utilizzare i formati proprietari dei dati che richiedono a tutte le parti coinvolte il possesso dello stesso software: l'uso dei meta linguaggi di descrizione consente la pubblicazione su Web della struttura dati, in modo da visualizzarla ed utilizzarla liberamente. Inoltre, poiché la descrizione può essere passata ad uno strumento automatizzato, aumentano le possibilità di conversione automatica dei dati da un formato ad un altro, o da un linguaggio ad un altro, a spese di un minimo intervento umano. Non va infine tralasciato un aspetto molto più sottile: se non ci si abitua a ragionare sulla struttura dei propri documenti piuttosto che a soffermarsi sulla loro formattazione si rischiano molteplici quanto inutili riscritture.
In termini di convalida della forma e della sintassi, XML possiede due nozioni differenti di correttezza. La prima sottintende semplicemente che il markup sia comprensibile e rappresenta l'equivalente XML di una "corretta pronuncia". Un documento XML contenente un markup comprensibile si dice **ben formato**; ciò accade quando vengono rispettate le seguenti regole:

1) I tag devono essere nidificati correttamente. Ogni coppia di tag (di apertura e di chiusura) deve contenere qualunque altra coppia di tag che inizia al proprio interno. In altre parole nessun elemento XML può iniziare in un'entità e terminare in un'altra.

2) Il nome del tag di chiusura di un elemento deve corrispondere a quello del rispettivo tag di apertura.

3) Nessun nome di attributo può apparire più di una volta nello stesso tag di apertura o in quello relativo ad un elemento vuoto.

4) I valori degli attributi non possono contenere riferimenti, diretti o indiretti, ad entità esterne. Questo vincolo è stato creato per semplificare l'elaborazione dei processori XML: qualora fossero possibili codifiche arbitrarie di caratteri nelle entità esterne, sarebbe difficile gestirle tutte correttamente all'interno di un attributo.

5) Il testo sostitutivo di un'entità riferita nel valore di un attributo non deve contenere il carattere "<".

6) La dichiarazione di un'entità parametrica deve precedere qualunque riferimento ad essa.

7) La dichiarazione di un'entità deve precedere qualunque riferimento ad essa che appaia sotto forma di

valore predefinito all'interno della dichiarazione di una lista di attributi. In altre parole non è possibile mischiare i dati binari in mezzo al testo senza qualche meccanismo di gestione dichiarato.

8) I riferimenti alle entità parametriche possono apparire soltanto nella descrizione della struttura del documento. Ciò significa che non è possibile riportare istruzioni di elaborazione nell'istanza del documento aspettandosi che queste assumono qualche significato.

La seconda nozione di correttezza XML è chiamata validità ed è correlata a quella del tipo di documento. Un documento XML ben formato si dice **valido** quando è stato verificato, con esito positivo, nei confronti di tutte le regole contenute nel documento che definisce la sua struttura. Il solo metodo pratico per decidere la validità di un documento XML è quello di utilizzare uno strumento automatico di analisi che legga il documento e ne verifichi la rispondenza con la sua descrizione strutturale: le verifiche manuali possono essere utili ma sono notoriamente soggette ad errori, anche quando si presta molta attenzione. I documenti privi di una dichiarazione di tipo non sono propriamente documenti non validi, in quanto non violano alcun vincolo, ma non possono nemmeno essere considerati validi, in quanto non è dato a sapere lo schema a cui sono conformi. Se un documento possiede una natura molto particolare ed una dimensione contenuta può essere sufficiente garantire che sia ben formato; se però deve far parte di un sistema informativo o se si tratta di un documento particolarmente esteso, è molto meglio scrivere un DTD e fare in modo che il documento sia valido. Dovendo costruire od estendere un sistema informatico, il fatto che la coerenza del documento sia garantita renderà la programmazione e la creazione di fogli di stile molto più semplice ed affidabile.

Struttura dei documenti XML

Un documento XML è fisicamente costituito da una o più **entità**: con questo termine si intendono le porzioni di testo sorgente, che possono avere dimensione variabile fra il singolo carattere e l'insieme di caratteri contenuti in un libro. Questa struttura costituisce un metodo di organizzazione non lineare e modulare del testo: sarà compito del parser (il programma che si occupa dell'analisi del sorgente) riorganizzare il tutto in forma lineare. Le entità sono dotate di nomi: questo facilita l'inserimento di riferimenti che consentono l'utilizzo delle entità stesse. Il parser sostituirà il riferimento all'entità con il corpo della medesima, detto **testo sostitutivo**. Si tratta in sostanza di un funzionamento abbastanza simile a quello delle macro in un qualunque editor di testi o linguaggio di alto livello. In questo modo si facilita il riutilizzo automatico del testo in ambiti differenti e si permette la propagazione in tutti i punti di richiamo delle modifiche apportate ad un'entità. A questo proposito va precisato che l'elaboratore può reperire i contenuti direttamente sul Web ed inserirli poi all'interno del documento: è il caso della cosiddette entità esterne. Da un punto di vista semantico le entità possono essere analizzate o non analizzate: le prime rappresentano tutto ciò che può essere compreso dal processore XML, come ad esempio i tag, mentre le restanti sono costituite dai dati binari o da quelli significativi solo per le applicazioni (il testo puro, le immagini, i filmati e gli elementi multimediali in genere).
Se le entità specificano i singoli agglomerati di byte costituenti un documento XML, gli elementi ne descrivono la struttura logica. Ciascuno di essi può contenere altri elementi oppure parole e frasi che vengono normalmente considerati il testo del documento. L'elemento che contiene tutti gli altri è chiamato **radice**, sottolineando con ciò il fatto che è l'unico elemento non dipendente da altri. Gli elementi che derivano dalla radice sono detti invece **sottoelementi**: nel caso contengano altri elementi si dicono più propriamente **rami**, mentre se contengono semplici dati di tipo testuale si dicono **foglie**. Vediamo un esempio pratico di un oggetto che può essere organizzato in una gerarchia ad albero: una bottiglia di aspirina e è composta da un corpo, da un coperchio di qualche tipo, da una o più etichette

che ne descrivono il contenuto e dal contenuto stesso. Dovendo assemblare le bottiglie di aspirina a partire dai loro componenti si noterebbe l'esistenza di un ordine nelle operazioni da eseguire per evitare di perdere le pillole: prima vanno poste le etichette, poi le pastiglie ed infine il coperchio. Una descrizione XML potrebbe essere la seguente:

```
<bottle>
    <top>
        coperchio di sicurezza
    </top>
    <body>
        <body-type>
            contenitore da 100 pastiglie
        </body-type>
        <contents>
            <count>
                100
            </count>
            <content-type>
                aspirine
            </content-type>
        </contents>
    </body>
    <labelling>
        <frontlabel>
            XYZ indicazioni generiche
        </frontlabel>
        <rearlabel>
            XYZ istruzioni e controindicazioni
        </rearlabel>
    </labelling>
</bottle>
```

Ogni elemento secondario è correttamente nidificato all'interno del proprio elemento contenitore e l'unico elemento radice è la bottiglia. Notiamo come, in analogia con HTML, anche XML è strutturato per mezzo di coppie di tag corrispondenti (uno di apertura ed uno di chiusura) che racchiudono la maggior parte del contenuto di un documento:

HTML: **<H1> Headline </H1>**

XML: **<headline1> Headline </headline1>**

XML incoraggia, ma non obbliga, una lunghezza di descrizione leggermente maggiore rispetto ad HTML; inoltre tutti linguaggi XML sono sensibili ai caratteri maiuscoli e minuscoli. E' anche possibile che un elemento contenga informazioni aggiuntive sotto forma di attributi, che descrivono le proprietà dell'elemento stesso. Gli attributi sono usati mediante sintassi identica ad HTML, ma devono essere sempre racchiusi tra virgolette:

HTML: **<IMG height=20 width=20 src="myimage.gif">**

XML: **<image height="20" width="20" source="myimage.gif" />**

Diversamente da HTML non è consentito l'uso di alcun tag senza il corrispondente tag di chiusura: anche i tag vuoti devono essere chiusi. Il metodo migliore per farlo consiste nell'inserire una barra alla fine del tag, separata attraverso uno spazio dal resto del contenuto.
Inoltre, l'ordine dei tag XML non è così libero: essi vanno sempre chiusi nel contesto in cui sono stati aperti. L'inserimento dei tag nell'ordine sbagliato, tollerato dalla maggior parte dei browser HTML, è proibito in XML. Il primo tag aperto è sempre l'ultimo che viene chiuso:

CORRETTO: <i><b> Italics Bold Text <b><i>

ERRATO: <i><b> Italics Bold Text <i><b>

Analizziamo nel dettaglio la struttura di un documento XML: esso dovrebbe cominciare con una dichiarazione che identifichi il file come un file XML, specificando al contempo la versione di XML utilizzata.

<? xml version =”1.0” encoding=”ISO-8859-1” ?>

Questa dichiarazione è in realtà facoltativa poiché esistono sul Web molti files HTML e SGML che sono documenti XML ben formati. Oltre alla versione è possibile specificare la codifica, ossia l'insieme dei caratteri utilizzati nel documento. A differenza di HTML, che presuppone l'uso del set di caratteri ASCII a 7 bit, XML consente di utilizzare svariati set di caratteri alfabetici e/o ideogrammi, sia per i nomi degli elementi che per il loro contenuto. L’insieme dei caratteri predefinito è Unicode, spesso usato sui sistemi Unix, che rappresenta una sorta di ASCII esteso. Tuttavia, parte del set di caratteri Unicode è compatibile con ASCII: in particolare esiste una codifica, UTF-8, caratterizzata dall’uguaglianza a livello di bit e di byte dei primi 128 caratteri. Va anche detto però che la maggior parte dei sistemi Windows usa la codifica ISO-8859-1, ossia l'insieme di caratteri usato per rappresentare l'inglese britannico, quello americano e la maggior parte degli altri linguaggi dell'Europa occidentale.
La seconda istruzione che costituisce il prologo di un documento XML è la dichiarazione del tipo di documento. Il suo scopo è l’identificazione del file contenente le definizioni dei tipi di elemento, degli attributi, delle entità e la specifica dei punti in cui tali definizioni sono da ritenersi valide:

<! DOCTYPE doc-name SYSTEM “{uri}” >

Mentre il prologo contiene informazioni riguardanti l’interpretazione dell’istanza, il corpo di un documento XML contiene i dati veri e propri, organizzati come una gerarchia di elementi. Nonostante lo schema associato al documento indichi i tipi di dati utilizzabili nell’istanza, il tipo di dato predefinito è sempre CDATA, ossia il normale carattere. La maggior parte degli elementi contenuti nei documenti XML può essere scandita in maniera sequenziale, seguendo il principio della nidificazione: un tag di apertura rappresenta l'inizio di un nodo o di una foglia, mentre il tag di chiusura ne costituisce il termine. Tutti i tag incontrati fra un tag di apertura ed il corrispondente tag di chiusura segnano l'inizio di un nuovo nodo o di una foglia. La parte rimanente della struttura logica del documento è definita dagli attributi associati a ciascun elemento.

Analisi comparativa dei principali linguaggi di schema XML

Introduzione

Fra i circa 12 linguaggi di definizione dei tipi di documento XML proposti fino al gennaio 2001 alcuni presentano caratteristiche particolarmente significative, tali da giustificarne una trattazione approfondita:

1) Sviluppo e supporto fornito da organizzazioni autorevoli, con elevata probabilità di sopravvivenza al processo di standardizzazione.

2) Comprovata e diffusa conoscenza di applicazioni già implementate.

3) Adozione di una filosofia differente dallo standard de-facto rappresentato dal DTD.

Sulla base di queste considerazioni si è deciso di confrontare tra loro DTD, XML-Schema, XDR, SOX, Schematron, DSD e RELAX.

DTD (Document Type Definition):
è un sottoinsieme del DTD SGML e rappresenta lo standard de facto per i linguaggi di descrizione XML, possedendo quindi le maggiori probabilità di sopravvivenza alla selezione che definirà uno standard. Rappresenta il mondo reale facendo uso di strutture gerarchiche di elementi: se confrontato con gli altri linguaggi possiede capacità espressive più limitate.

XML Schema:
Rappresenta uno sforzo in atto da parte del W3C per migliorare ed eventualmente rimpiazzare il DTD, rispetto al quale offre maggior espressività e possibilità di utilizzo, anche grazie all'elevato numero di applicazioni circolanti. Possiede caratteristiche innovative quali l'ereditarietà degli attributi e degli elementi, oltre che un potente meccanismo di definizione dei dati da parte dell'utente.

XDR (XML-Data Reduced):
Precedentemente conosciuto come XML-Data, è stato successivamente rivisto e potenziato in seguito allo sforzo di Microsoft che ne fa uso all'interno del framework BizTalk. Risente inoltre dell'influenza del DCD, un altro linguaggio nato dalla collaborazione fra Microsoft e IBM, con cui condivide molte caratteristiche comuni.

SOX (Schema for Object-Oriented XML):
Si tratta di un linguaggio di descrizione alternativo, pensato per la definizione delle strutture sintattiche ed, almeno in parte, di quelle semantiche all'interno dei documenti XML. Come suggerito dal nome, estende il DTD in una forma orientata agli oggetti, consentendo l'uso dei tipi di dati estensibili e dell'ereditarietà fra i tipi degli elementi.

Schematron:
Creato da Rick Jelliffe, si distingue dagli altri linguaggi in quanto orientato alla validazione degli schemi mediante pattern, piuttosto che alla loro definizione. Propone una definizione di schema

semplice quanto basta per essere contenuta in un'unica pagina e tuttavia consente, attraverso XPATH, la specifica di potenti vincoli.

DSD:
E' stato sviluppato congiuntamente dai laboratori At&T e da BRICS col duplice obiettivo di fornire una descrizione di elementi ed attributi dipendente dal contesto e di offrire al tempo stesso meccanismi flessibili di inserimento dei dati. Si caratterizza per una potenza espressiva vicina a quella di XSLT e, analogamente a Schematron, pone un forte accento sui vincoli.

RELAX (REgular LAnguage description for XML) :
Si tratta di un nuovo linguaggio in corso di standardizzazione da parte della JSA (Japanese Standard Association) e dell'ISO (International Standard Organization). Rappresenta una tappa fondamentale nella migrazione dal DTD all' XML-Schema ed offre quindi tutte le caratteristiche proprie del DTD. In aggiunta, è in grado di esprimere regole di descrizione sensibili al contesto grazie all'uso di simboli non terminali.

Analisi delle caratteristiche

A) Schema:

A1) Sintassi XML:

L'utilizzo della sintassi XML nei linguaggi di schema comporta numerosi vantaggi: gli utenti non debbono imparare una nuova sintassi proprietaria; il linguaggio XML è facilmente implementabile all'interno di applicazioni esistenti (browser, editor, ecc.); i file di descrizione si possono memorizzare nel medesimo sistema di archiviazione XML contenente i documenti ed infine si tratta di un linguaggio facilmente estensibile. Tutti i linguaggi di descrizione, ad eccezione del DTD, adottano questa sintassi.

A2) Namespace:

I namespace offrono un modo per riutilizzare i tag all'interno documenti, senza incorrere nei problemi derivanti da possibili collisioni. Col termine **collisione** si intende la possibilità che un elemento od una combinazione elemento/attributo definito in uno schema possieda lo stesso nome di un'analoga combinazione definita in un altro schema. Facendo esplicito riferimento al contesto nel quale è definito un particolare tag, si garantisce l'univocità dei nomi agli elementi ed agli attributi. Per esempio, dal momento che un DTD garantisce al suo interno tale univocità, è possibile correlare ad esso un namespace utilizzando l'URL del DTD come identificatore univoco.
Supponiamo di voler definire l'elemento "book" riutilizzando l'elemento "address" definito altrove (tramite percorso specificato dall'URI) ed aggiungendo un elemento specifico "title". In XML-Schema questo è possibile mediante il codice:

```
<schema xmlns:z="URI">
    <element name="book">
        <complex type>
            <element name=" title" type="string">
            <element name="address"  type="z:address">
        <complex type>
```

```
        </element>
</schema>
```

XDR possiede una sintassi simile, così come Schematron che dispone dell'attributo 'Ns' per l'elemento <schema>. SOX supporta invece l'elemento <namespace> e gli attributi 'prefix' e 'type' per qualificare i nomi:

```
<namespace prefix="z" namespace="URI" />
<elementtype name="book">
        <model>
                <element type="title">
                <element prefix="z" type="address">
        </model>
</elementtype>
```

Mentre il DTD non implementa i namespace, sia DSD che RELAX sono progettati per supportarli, anche se al momento le relative specifiche non risultano disponibili.

A3) Include & Import:

A volte è opportuno includere nello schema porzioni di codice specificate esternamente. Questa esigenza si riscontra in particolare quando le dimensioni dello schema aumentano notevolmente: è preferibile allora optare per una struttura modulare che garantisca di fatto una miglior leggibilità. L'importazione di uno schema (o di una sua parte) facente riferimento allo *stesso* namespace dello schema principale presuppone l'utilizzo della direttiva "include"; diversamente si utilizza la direttiva "import". La prima è supportata da XML-Schema, SOX, DSD e RELAX, mentre la seconda soltanto da XML-Schema e SOX. In particolare questi ultimi prevedono la possibilità di importare namespace multipli, che possono sovrascrivere quello di default dichiarato nello schema principale.

B) Datatype:

I tipi dei dati si possono classificare in due categorie: i tipi semplici e quelli complessi. Un tipo semplice non può contenere elementi e non è possibile associarvi degli attributi; un tipo complesso può invece sfruttare entrambe queste possibilità.

B1) Tipi di dati built-in:

Si tratta di primitive o di tipi semplici derivati dalle specifiche del linguaggio di definizione. La maggior parte dei linguaggi, ad eccezione di Schematron e DSD, supporta un insieme di tipi built-in comprendente le stringhe ed i tipi specifici di XML (ID, NMTOKEN, ecc.). In particolare XML-Schema, SOX, XDR e RELAX supportano un set esteso di tipi built-in, comprendente molti di quelli usati nei linguaggi di programmazione general-purpose. Essendo invece l'obiettivo di Schematron la validazione delle strutture XML, esso non fornisce esplicitamente alcun tipo built-in. DSD si comporta allo stesso modo, pur offrendo la possibilità di simulare molti tipi di dati attraverso il supporto alle espressioni regolari.

B2) Tipi user-defined:

In XML-Schema si possono creare nuovi tipi semplici mediante ereditarietà applicata ai tipi built-in od a quelli derivati. In SOX è possibile definire nuovi tipi attraverso l'utilizzo di tre costrutti: <Enumeration>, <Scalar> e <Varchar>. DSD utilizza il costrutto <StringTypeDef> insieme ad un vasto gruppo di operatori e di espressioni regolari per consentire la definizione di nuovi tipi di dato. Per esempio la descrizione di un codice postale americano a nove cifre si può operare nel modo seguente:

```
<StringTypeDef  ID="zipcode">
        <Sequence>
                <Repeat value="5" />
                        <CharSet value="0123456789" />
                </Repeat>
                <Optional>
                        <String value="-" />
                        <Repeat value="4" />
                                <CharSet value="0123456789" />
                        </Repeat>
                </Optional>
        </Sequence>
<StringTypeDef>
```

B3) Vincoli sul dominio dei tipi:

Non è importante solo il tipo del dato ma anche i valori considerati legali per il dato stesso. I vincoli sul dominio di un certo dato permettono, attraverso l'uso di opportuni costrutti, di definire i valori consentiti per quel tipo. XML-Schema e RELAX supportano una grande varietà di controlli (range, precisione, lunghezza, maschera, ecc.) e di espressioni regolari per compiere verifiche in questo senso. SOX dal canto suo offre numerose primitive di controllo tra cui l'enumerazione, il valor massimo e minimo, la lunghezza massima, ecc. Nonostante ciò SOX non supporta un vero e proprio pattern-language. Benchè Schematron non supporta ne i tipi built-in né quelli user-defined è possibile simulare alcuni di questi attraverso il supporto di XPath. Ad esempio, il tipo integer per l'elemento <E> può essere simulato nel seguente modo:

```
<rule context = 'E'>
        <assert test = 'floor(.) = number(.)' >
                E can have only integer value.
        </assert>
</rule>
```

Come già mostrato nel caso dei tipi user-defined, DSD supporta un insieme di operatori pattern-related in grado di imporre vincoli sul dominio di tali tipi.

B4) Explicit null:

Spesso risulta utile la distinzione fra valori sconosciuti, presenti od assenti, operata mediante il supporto esplicito al valore "null". Solo XML-Schema rende disponibile un attributo 'Nullable' per

indicare che il contenuto dell'elemento è nullo. In un'istanza di documento XML invece viene settato al valore "true" l'attributo 'null' dell'elemento per indicarne il contenuto nullo.

XML-SCHEMA: <element name="fullname" nullable="true" />
ISTANZA: <fullname xsi:null="true"> </fullname>

C) Attributi:

C1) Valori di default:

RELAX e Schematron non supportano valori di default per gli attributi. In RELAX questa caratteristica è stata omessa intenzionalmente allo scopo di mantenere la compatibilità coi processori XML esistenti: il settaggio dei valori di default è lasciato quindi ai programmi applicativi. Per quanto riguarda il DTD, se nella dichiarazione di un attributo non compaiono i valori #REQUIRED o #IMPLIED è sottinteso che il valore dell'attributo coincida con quello dichiarato a default. Nell'esempio seguente:

```
<!ATTLIST list type (bullets|ordered) "ordered">
<!ATTLIST form method CDATA #FIXED "POST">
```

notiamo come l'attributo 'type' dell'elemento <list> possieda a default il valore "ordered", mentre l'attributo 'method' dell'elemento <form> sia caratterizzato dal valore prefissato "POST". Gli altri linguaggi supportano meccanismi equivalenti: soltanto il DSD offre qualcosa di più sofisticato. I valori di default vengono infatti associati ad un'espressione booleana come in questo esempio, in cui il valore di default "John Doe" per l'attributo 'nm' viene associato agli impiegati di sesso maschile:

```
<Default>
    <Context>
        <Element Name="employee">
            <Attribute Name="gender" Value="M" />
        </Element>
    </Context>
    <DefaultAttribute Name="nm" Value="John Doe" />
</Default>
```

C2) Scelta fra gli attributi:

Si tratta di un'operazione utile ai progettisti nel momento in cui desiderano associare ad un elemento la presenza di un solo attributo scelto all'interno di una lista. Solamente Schematron e DSD sono in grado di supportare questa caratteristica. Ecco come DSD consente di vincolare la presenza di uno solo fra gli attributi 'fn' e 'gn' relativi all'elemento <person>:

```
<ElementDef ID="person">
    <AttributeDecl Name="fn" IDType="ID" />
    <AttributeDecl Name="gn" IDType="ID" />
    <OneOf>
        <Attribute Name="fn" /> <Attribute Name="gn" />
    </OneOf>
</ElementDef>
```

C3) Optional vs. Required:

Tutti i linguaggi sono in grado di definire la presenza obbligatoria od opzionale del valore di un attributo relativo ad un certo elemento. Il DTD ad esempio utilizza allo scopo la keyword #REQUIRED

C4) Vincoli sul dominio di un attributo:

DTD, XDR e SOX prevedono soltanto il costrutto di enumerazione mediante il quale è possibile elencare tutti i valori consentiti per un determinato attributo. In XML-Schema invece è possibile definire degli attributi basati su tipi semplici il cui dominio è stato preventivamente vincolato. RELAX consente di applicare agli attributi vincoli quali <Enumeration>, <Mininclusive>, <Maxinclusive>. Schematron sfrutta invece regole del tutto simili a quelle mostrate nella sezione dei vincoli sui datatype (B3). In DSD infine è previsto l'utilizzo di operatori quali <Union> e <Repeat>, applicabili al costrutto <StringType>, al fine di vincolare i valori di dominio degli attributi.

D) Elementi:

D1) Valori di default:

Ciascun elemento può possedere un valore di default di tipo semplice o complesso. Questa funzionalità però viene supportata soltanto da XML-Schema e DSD. In XML-Schema si può definire una qualunque stringa come valore di default qualora l'elemento sia costituito da un tipo di dato semplice. L'esempio seguente mostra l'associazione di un valore di default all'elemento di tipo stringa <fullname>:

```
<element name="fullname"  type="string"  default="John Doe" />
```

DSD è più versatile in quanto permette, attraverso il tag <DefaultContent>, di associare valori di default sia a tipi semplici sia a tipi complessi. Ad esempio è possibile specificare come default per l'elemento di tipo complesso <address> la coppia di valori "Los Angeles" e "CA":

```
<Default>
    <Context>
        <Element Name="address" />
    </Context>
    <DefaultContent>
        <city> Los Angeles </city>
        <state> CA </state>
    </DefaultContent >
</Default>
```

D2) Contenuto degli elementi:

Il contenuto degli elementi può essere nullo, può basarsi su datatype od altri elementi, oppure ancora può essere misto (datatype ed elementi). Il DTD supporta tutti questi contenuti attraverso le dichiarazioni:

EMPTY: **<!ELEMENT element1 EMPTY >**
TEXT: **<!ELEMENT element2 (#PCDATA) >**
ELEMENT: **<!ELEMENT element3 (x? | y* | z+) >**
MIXED: **<!ELEMENT element4 (#PCDATA | x)* >**

XML-Schema, RELAX, Schematron, XDR e DSD si comportano in maniera analoga, mentre SOX offre un supporto limitato ai primi tre tipi di contenuti, tramite i costrutti <Empty />, <String /> ed <Element />.

D3) Sequenza ordinata:

Si definisce ogni qualvolta vada preservato l'ordine fra i sottoelementi. In DTD i sottoelementi separati da un operatore "," devono rispettare l'ordine con cui vengono elencati. Anche negli altri linguaggi occorre preservare l'ordine ove non specificato diversamente; in genere cio è possibile utilizzando il costrutto <Sequence>. L'esempio seguente mostra la definizione in SOX dell'elemento <person>, vincolato a contenere il sottoelemento <fn> seguito dal sottoelemento <ln>:

```
<elementtype name="person">
    <model>
        <sequence>
            <element name="fn" />
            <element name="ln" />
        </sequence>
    </model>
</elementtype>
```

D4) Sequenza non ordinata:

Contrariamente a SGML, che dispone di un operatore & per creare una sequenza non ordinata, il DTD non offre un supporto esplicito in tal senso: occorre pertanto codificare ogni possibile combinazione dei sottoelementi. Ad esempio, la sequenza non ordinata di sottoelementi (a & b & c) deve essere espressa in DTD mediante la sintassi ((a,b,c)|(a,c,b)|(b,a,c)|(b,c,a)|(c,a,b)|(c,b,a)). In XML-Schema è possibile specificare una sequenza non ordinata mediante il costrutto <All>. In Schematron invece la sequenza non ordinata è ottenuta per default, qualora cioè non vengano specificati particolari pattern. In DSD una singola espressione è in grado di descrivere un insieme di sequenze consentite, mentre più espressioni possono descrivere l'intero insieme di sequenze permesse, un gruppo per ciascuna espressione. Si tratta di un metodo più flessibile che, se sfruttato in modo intelligente, permette di contemplare "elementi floating" come ad esempio gruppi misti di sequenze ordinate e non ordinate. In XDR infine, l'attributo 'Order' settato con valore "many" specifica che i sottoelementi possono comparire in un ordine qualunque.

D5) Scelta fra i sottoelementi:

Il DTD mette a disposizione un operatore "|" per consentire la scelta di un unico sottoelemento all'interno di una lista. In XML-Schema, RELAX e SOX la stessa caratteristica è supportata mediante il costrutto <Choice>. Schematron consente invece di effettuare una scelta fra i sottoelementi utilizzando delle regole simili a quelle viste per la scelta di un attributo. XDR utilizza l'attributo

‘Order’ settato al valore ”one”, mentre invece DSD contempla il costrutto <OneOf>, come mostra il seguente esempio:

```
<ElementDef  ID=”person”>
      <OneOf>
            <Element  Name=”fn” />
            <Element Name=”gn” />
      </OneOf>
</ElementDef>
```

D6) Occorrenza minima e massima:

Alcuni linguaggi permettono di vincolare le occorrenze minime e massime degli elementi e del loro contenuto. In DTD e RELAX si può effettuare soltanto un controllo parziale mediante gli operatori di Kleene:

“?” per le occorrenze 0/1;
“*” per le occorrenze 0/molti;
“+” per le occorrenze 1/molti.

XML-Schema ed XDR consentono un approccio più preciso grazie agli attributi ‘MinOccurs’ e ‘MaxOccurs’ che compaiono nella dichiarazione dei vari elementi. In SOX invece la definizione di un elemento è accompagnata dall’attributo ‘Occurs’ che ne indica l’occorrenza. Quest’ultima può coincidere con una coppia di valori nella forma “MinOccur, MaxOccur”, o con uno qualunque degli operatori di Kleene. In Schematron si possono formulare controlli analoghi mediante le espressioni:

```
<Assert test=”count(E) >= k”>
<Assert test=”count(E) <= l”>
```

In DSD l’occorrenza degli elementi può essere definita <Optional>, <Zero or More>, <One or More> oppure <Union>, ma non è possibile fornire un’indicazione numerica della cardinalità minima e massima.

E) Ereditarietà:

In accordo col paradigma orientato agli oggetti l’ereditarietà è ottenuta per estensione o restrizione dei basetype. Analizziamo questa caratteristica facendo riferimento ai tipi di dati semplici ed a quelli complessi.

E1) Tipi semplici (estensione):

In teoria si possono creare nuovi tipi semplici derivandoli da altri già esistenti grazie a vincoli di dominio più rilassati. L’insieme dei valori ammissibili per il nuovo tipo costituisce un superset di quello del tipo base. Nella pratica però nessun linguaggio è in grado di supportare questa caratteristica.

E2) Tipi semplici (restrizione):

L'insieme dei valori ammissibili per il nuovo tipo è un sottoinsieme di quello del tipo base. XMLSchema supporta questa caratteristica, come mostra il seguente esempio in cui un codice postale americano a 9 cifre viene creato a partire dal tipo "string":

```
<simpleType name="zipcode" base="string">
    <restriction base="string">
        <pattern value="[0-9] {5} (-[0-9]{4})?" />
    </restriction>
</simpleType>
```

Le occorrenze dei codici postali sono ottenute restringendo il dominio degli elementi di tipo "string" mediante uso di pattern expression. Anche SOX permette di raffinare dati di tipo built-in o di tipo derivato allo scopo di ottenerne dei nuovi. In questo esempio il tipo "RGB" si ottiene rendendo ammissibili solo alcuni valori per il tipo "color":

```
<datatype name="RGB">
    <enumeration datatype="color">
        <option> Red   </option>
        <option> Green </option>
        <option> Blue  </option>
    </enumeration>
</datatype>
```

E3) Tipi complessi (estensione):

In questo frangente soltanto XML-Schema e SOX offrono un supporto: il primo sfrutta il costrutto <Extension base="SomeBaseType" />. In aggiunta ad esso si utilizza anche il tag <ComplexContent> per dichiarare che il contenuto dei nuovi tipi è complesso (ad esempio, quando i tipi estesi contengono degli elementi). SOX offre le stesse funzionalità grazie al costrutto <Extends type="basetype">. Sia in XML-Schema sia in SOX i nuovi elementi e/o attributi vengono sempre aggiunti in fondo alla struttura del tipo base. Nel seguente esempio si definisce l'elemento <new-person> che eredita il contenuto dell'elemento <person>, aggiungendovi l'elemento <address> e l'attributo 'email':

```
<elementtype name="new-person">
    <extends type="person">
        <append>
            <element name="address" type="addr" />
        </append>
        <attdef name="email" datatype="string" />
    </extends>
</elementtype>
```

E4) Tipi complessi (restrizione):

XML-Schema è l'unico linguaggio che permette la derivazione dei tipi complessi per restrizione. In questo caso il concetto di ereditarietà sottintende la ridefinizione del contenuto originario mediante

aggiunta di vincoli di cardinalità piuttosto che la modifica del contenuto stesso. Per chiarire il concetto mostriamo come il nuovo tipo <NewItemType>, ottenuto derivando <ItemType>, sia vincolato a contenere almeno 100 elementi di tipo <Item>:

```
<complexType name="ItemType">
    <sequence>
        <element name="Item" minOccurs="0" />
        <element name="Price" />
    </sequence>
</complexType>

<complexType name="NewItemType">
    <restriction base="ItemType">
        <sequence>
            <element name="Item" minOccurs="100" />
            <element name="Price" />
        </sequence>
    </restriction>
</complexType>
```

F) Unicità:

F1) Unicità degli attributi:

Per dichiarare l'unicità di un attributo DTD, RELAX, SOX, XDR e DSD fanno uso del tipo ID, mentre XML-Schema utilizza la parola chiave <Unique> ed i costrutti <Selector> e <Field> che permettono di specificare rispettivamente la collocazione dell'attributo interessato dall'unicità ed il suo nome. Schematron, pur non possedendo un costrutto esplicito, è in grado di simulare l'unicità degli attributi utilizzando il pattern "count()=1".

F2) Unicità per i non attributi:

Linguaggi di descrizione come XML-Schema, Schematron o XDR permettono di specificare l'unicità anche riguardo ad elementi arbitrari od oggetti composti (attributi + elementi). XML-Schema utilizza ancora il costrutto <Unique>; il seguente codice ad esempio assicura l'unicità dell'elemento <phone> all'interno di un sottoelemento <addr> dell'elemento <persona>.

```
<unique>
    <selector> person/addr </selector>
    <field> phone </field>
</unique>
```

In XDR invece gli elementi possono far uso del tipo ID come se fossero attributi (nonostante questa funzionalità non sia ancora supportata da Internet Explorer 5):

```
<ElementType name="phone" dt:type="ID" />
```

Tuttavia XDR non è in grado di supportare l'unicità per gli oggetti composti.

F3) Attributi designati come chiavi:

In un DB i valori chiave devono soddisfare i requisiti di esistenza ed unicità. Similmente, XMLSchema sfrutta i costrutti <Unique> e <Key> per designare un attributo al ruolo di chiave. Schematron, linguaggio finalizzato alla validazione degli schemi, esprime lo stesso vincolo mediante questo codice:

```
<rule  context = "person">
        <assert  test = "@ssn and count(@ssn) = 1"> Is ssn unique? </assert>
        <assert test = "string-length(@ssn) > 0"> Is ssn not empty? </assert>
</rule>
```

F4) Altri elementi designati come chiavi:

XML-Schema e Schematron permettono di definire come chiavi elementi arbitrari od oggetti composti. Per esempio si può definire una chiave costituita dalla combinazione del codice di dipartimento (elemento) e dal nome di un impiegato (attributo):

```
<key  name="ekey">
        <selector>  employee  </selector>
        <field> dept/code  </field>
        <field> @name  </field>
</key>
```

F5) Attributi designati come chiavi importate:

Il concetto di "chiave importata" identifica sia gli attributi che costituiscono la foreign key sia quelli referenziati dalla chiave medesima. Analogamente all'uso del tipo ID, i linguaggi DTD, XDR, RELAX, SOX e DSD utilizzano il tipo IDREF per identificare un attributo che compone una chiave importata. XML-Schema usa invece il costrutto <Keyref>. Inoltre XML-Schema e DSD dispongono rispettivamente dei costrutti <Refer> e <PointsTo> per specificare a chi si riferisce attualmente la chiave importata. Schematron offre il supporto a questa caratteristica mediante l'uso di patterns. Il seguente codice mostra come l'attributo 'dno' dell'elemento <employee> costituisca l'identificatore univoco dell'elemento <dept>:

```
<rule context = "employee[@dno]">
        <assert test = "(name(id(@dno)) = 'dept')'> Error occurred. </assert>
</rule>
```

F6) Chiavi importate che non referenziano attributi:

XML-Schema consente la definizione di chiavi importate che identificano elementi arbitrari od oggetti composti, sempre grazie al costrutto <Keyref>. Analoga opportunità è offerta dal DSD; nell'esempio seguente un attributo "book-ref" identifica un elemento <book>:

```
<AttributeDecl   ID ="book-ref"  IDType="IDRef">
        <PointsTo>
                <Context> <Element Name="book" /> </Context>
        </PointsTo>
</AttributeDecl>
```

G) Sensibilità al contesto:

In genere la presenza di elementi ed attributi è accettata limitatamente a specifici contesti. Nei linguaggi di schema si riconoscono tre tipologie principali di regole di sensibilità al contesto:

1) Definizioni di attributi dipendenti dai valori di altri attributi.
2) Definizioni di elementi (o del loro tipo di contenuto) dipendenti dai valori di altri attributi.
3) Definizioni di elementi (o del loro tipo di contenuto) dipendenti dai valori di elementi progenitori.

Vediamo nel dettaglio i tre casi:

1) Spesso un attributo 'a1' di un elemento <E> ha rilevanza soltanto in presenza di un valore, generico o specifico, per l'attributo 'a2'. Ad esempio il seguente frammento di codice Schematron indica che la presenza dell'attributo 'one' dell'elemento <E > è vincolata alla presenza dell'attributo 'two':

```
<rule  context="E">
  <report  test="(@one) or not (@one and @two)">
        E cannot have attribute 'one' alone.
  </report>
</rule>
```

Il DSD supporta agevolmente questa caratteristica grazie al suo ricco insieme di operatori booleani. In questo esempio l'attributo 'salary' viene definito soltanto quando lo studente è di classe "TA":

```
<ElementDef  ID="student">
  <If><Attribute  Name="TA"  Value="yes">
        <Then><Optional>
              <AttributeDecl  Name="salary" />
        </Optional></Then>
  </If>
</ElementDef>
```

2) Questa regola, supportata da RELAX, Schematron e DSD, vincola il contenuto di un elemento a dipendere dal valore di un suo attributo: si supponga ad esempio che un elemento <val> possieda l'attributo 'type', il quale può assumere valore "integer" o "string". Il tipo di dato contenuto nell'elemento <val> dovrà allora dipendere dal valore dell'attributo "type". La correttezza del seguente codice XML dovrà essere validata dal processore:

```
LEGAL:      <val type="integer"> 1000 </val>
LEGAL:      <val type="string"> thousand </val>
ILLEGAL:    <val type="integer"> thousand </val>
```

3) In questo caso il contenuto del generico elemento <E1> è vincolato al contenuto del progenitore <E2>, all'interno del quale <E1> è innestato. Anche questa regola è supportata soltanto da RELAX, Schematron e DSD. La notazione seguente esprime nella sintassi di RELAX due regole: una consente all'elemento <para>, annidato nell'elemento <section>, di possedere l'elemento figlio

<footnote>; mentre l'altra impedisce all'elemento <para>, contenuto nell'elemento <cell>, di contenere l'elemento <footnote>:

```
<elementRule  role = ”para”  label = ”paraWithFNotes”>  // L'elemento <para> contiene <footnote>
        <mixed>
                <element name = ”footnote” occurs = ”*” />
        </mixed>
</elementRule>

<elementRule  role = ”para”  label = ”paraWithoutFNotes” >  // L'elemento <para> è vuoto
        <mixed>
                <empty />
        </mixed>
</elementRule>
<tag name = ”para” />

<elementRule role = ”section”>                                  // L'elemento <section> contiene <para>
        <ref  label = ”paraWithFNotes”  occurs = ”*” />
</elementRule>

<elementRule role = ”cell”>                                     // L'elemento <cell> contiene <para>
        <ref  label = ”paraWithoutFNotes”  occurs = ”*” />
</elementRule>
```

H) Aggiornamento dello schema:

Al fine di soddisfare le mutevoli necessità di rappresentazione dei dati deve essere garantita la possibilità di aggiornamento agli schemi XML. Le modifiche comprendono quelle situazioni in cui si aggiungono/eliminano degli elementi/attributi od in cui si procede ad una riorganizzazione dello schema. Consideriamo in particolare due caratteristiche legate all'aggiornamento:

H1) Contenuto estensibile del modello:

Uno schema a contenuto estensibile permette l'aggiunta di nuovi elementi o di nuovi attributi all'interno degli elementi esistenti. Grazie a questo meccanismo di estensione i nuovi elementi non debbono essere a loro volta dichiarati. L'unico linguaggio in grado si supportare questa caratteristica è Schematron che prevede a default la possibilità di estendere i contenuti del modello. Mediante l'utilizzo della funzione count( ) di XPath è anche possibile implementare la chiusura del contenuto, ovvero impedire a nuovi attributi di entrare a far parte della struttura di un dato elemento. Col seguente codice ad esempio si impone la chiusura dell'elemento <person> (dopo aver definito tutti sottoelementi ammessi):

```
<rule context =”person”>
      <assert  test =”count(name | address) = count(*)”>
            There is an extra element.</assert>
</rule>
```

H2) Versioni:

RELAX e DSD consentono la coesistenza di più versioni di un attributo o di un elemento: con ciò si intende la possibilità di specificare una serie di definizioni fra loro diverse caratterizzate però dallo stesso nome. In RELAX ciò si ottiene condividendo ridefinendo il tag <ElementRule>. Data la definizione dell'elemento <section>:

```
<tag name="section" />
<elementRule  role="section">
        <element  name="para"  occurs="*" />
</elementRule>
```

E' possibile modificare la definizione di <section> in modo che possa contenere, oltre agli elementi <para>, anche gli elementi <fig>:

```
<elementRule  role="section">
    <choice  occurs="*">
        <element  name="para" />
        <element  name="fig" />
    </choice>
</elementRule>
```

Il parser di RELAX non ravvisa problemi fintanto che l'istanza del documento fa match con l'una o con l'altra definizione. Il DSD definisce gli elementi facendo uso sia dell'attributo "Name" sia di quello "ID": in tal modo attributi con nomi identici sono legali fintanto che i rispettivi ID sono diversi. Inoltre, grazie ai costrutti <RenewID> e <CurrIDRef>, è possibile rivisitare qualunque definizione. Ecco un esempio di ridefinizione del vincolo DSD "book-constraints":

```
<ConstraintDef  ID = "book-constraints" />
<ConstraintDef  RenewID = "book-constraints" >
    <Constraint  CurrIDRef = "book-constraints" />
        …modifiche….
</ConstraintDef>
```

Caratteristiche ottimali per un linguaggio di descrizione dei dati

A seguito dell'analisi svolta si può tentare l'individuazione dell'insieme di caratteristiche ottimali per un linguaggio di schema:

1) Facilità d'uso:

Con ciò si intende la facilità nell'apprendimento e nell'utilizzo del linguaggio da parte di un progettista. Nonostante la sintassi proprietaria, il DTD è probabilmente il linguaggio più facile da imparare a da utilizzare. Dal momento che le nuove funzionalità aggiunte a RELAX, ad XDR ed a SOX sono alquanto versatili, è ragionevole pensare che la curva di migrazione dal DTD verso tali linguaggi non sia ripida.

Per quanto riguarda Schematron, benchè faccia uso di una sintassi semplice e potente, costringe gli utilizzatori ad imparare un altro linguaggio: XPath. XML-Schema e DSD, disponendo di un esteso insieme di funzionalità, risultano più difficili da apprendere rispetto agli altri linguaggi. Infine, poiché DSD utilizza operatori espliciti per le espressioni regolari (es: <Repeat>, <OneOf>, ecc.), uno schema definito mediante tale linguaggio risulta più prolisso di una descrizione XML-Schema o Schematron e, di conseguenza, più difficile da decifrare.

2) Potere espressivo:

Considerando dapprima la "natura intrinseca" dei linguaggi analizzati è possibile operare una distinzione in due grandi categorie, distinguendo fattori quali: struttura basata su grammatiche vs. struttura basata sui pattern; orientamento alla definizione dei dati vs. orientamento alla validazione delle strutture, ecc. Il DTD, XML-Schema, RELAX, XDR e SOX appartengono alla classe basata sulle grammatiche (od in termini equivalenti a quella orientata alla definizione delle strutture dati), mentre Schematron appartiene all'altro gruppo. Il DSD è collocato a metà strada, supportando le caratteristiche di entrambi i gruppi.
I linguaggi basati sulle grammatiche offrono particolari vantaggi nelle interrogazioni XML, poiché la definizione e la conoscenza dello schema aiuta gli utilizzatori a scrivere delle query ottimizzate e facilita loro le operazioni di debugging. Sull'altro fronte, i linguaggi basati sui pattern sono più potenti per quanto concerne l'espressività dei vincoli nelle applicazioni.
Analizzando invece le funzionalità dei vari linguaggi, XML-Schema appare il più autorevole dal momento che supporta la maggior parte delle caratteristiche considerate, eccezion fatta per alcuni aspetti legati alla gestione vincoli. Rispetto all'XML-Schema il DSD è carente riguardo ai namespace, ai datatype, alla modularità ed alle caratteristiche Object-Oriented. Anche Schematron, paragonato a XML-Schema, mostra dei limiti sui datatype, sulla modularità e sulle caratteristiche Object-Oriented. Tuttavia XML-Schema si dimostra il peggiore in assoluto relativamente al supporto delle regole di sensitività al contesto. SOX ed XDR offrono un supporto ai datatype più modesto rispetto ad XML-Schema ed inoltre, confrontandoli con DSD e Schematron, emergono carenze in diversi aspetti (possibilità di specificare vincoli sul dominio dei dati, contenuto dei dati, sensitività al contesto, proprietà di unicità e definizione delle chiavi, controllo delle versioni). RELAX eccelle nel supporto alle regole di sensitività al contesto, ma offre funzionalità modeste riguardo all'ereditarietà, alle proprietà di unicità ed ai valori di default. Il DTD appare infine come il linguaggio meno espressivo fra tutti, essendo in grado di supportare solamente i datatype ed i contenuti più basilari.

3) Supporto ai database:

Poiché i database rappresentano soltanto una fra le possibili applicazioni di uno schema XML, non esistono ragioni specifiche affinché un linguaggio di descrizione supporti in modo esplicito le caratteristiche legate ai DB (come ad es. le chiavi, i valori nulli, ecc.). Tuttavia, dal momento che la maggior parte dei dati disponibili sul web sono memorizzati in database è rilevante anche l'analisi di questo aspetto. Bisogna però concludere che nessun linguaggio soddisfa pienamente le esigenze. Il DDL (Data Definition Language) di SQL permette infatti di specificare un insieme di attributi e di relazioni, di definire il dominio di valori associato a ciascun attributo, i vincoli di integrità, gli indici per ciascuna relazione, le opzioni sulla sicurezza, ecc. I linguaggi visti si dimostrano invece più limitati: nonostante XML-Schema offra il supporto ad un ampio insieme di tipi built-in, non è in grado di esprimere, ad esempio, una clausola SQL arbitraria quale CHECK od ASSERT. Inoltre, benchè Schematron e DSD siano in grado di formulare tali vincoli di integrità, non gestiscono alcun indice fisico per il miglioramento delle prestazioni. E' lecito ritenere che nel prossimo futuro questa

rappresenterà una delle aree di ricerca di maggior interesse, dal momento che una considerevole quantità di documenti viene generata per mezzo delle interrogazioni formulate dagli utenti sui database sottostanti.

Classificazione dei linguaggi analizzati

Una ragionevole tassonomia dei linguaggi presentati si basa su tre categorie:

1) Core Schema Languages:
Il DTD è il linguaggio rappresentativo di questa classe, caratterizzata da un potere espressivo limitato. Il supporto alla struttura dello schema è minimo ed è molto carente quello relativo ai datatype ed ai vincoli. Dal momento che il potere espressivo del DTD è così povero, la traduzione di uno schema da un altro linguaggio è immediata, mentre quella inversa comporta delle probabili perdite nei dati.

2) Extended Schema Languages:
Nella classe intermedia si possono collocare RELAX, SOX ed XDR: il loro supporto ai datatype non è soddisfacente (ad es. mostrano carenze riguardo alla gestione dei valori “null” ed ai tipi user-defined), mentre è da considerarsi adeguato il supporto alla struttura dello schema. Inoltre essi mostrano delle carenze per quanto concerne il contenuto dei dati e le caratteristiche legate ai database. Analogamente al DTD non sono in grado di gestire con sufficiente granularità i vincoli al fine di esprimere in modo adeguato la semantica dello schema.

3) Expressive Schema Languages:
I linguaggi che entrano di diritto nella categoria di maggior espressività sono XML-Schema, Schematron e DSD. Mentre XML-Schema supporta pienamente le caratteristiche relative ai datatype ed alle strutture degli schemi, Schematron rappresenta un flessibile pattern-language in grado di descrivere dettagliatamente la semantica dello schema. Il DSD infine tenta di implementare tutte queste caratteristiche, aggiungendovi ulteriori funzionalità.

Conversioni fra le strutture e scambio dei dati

Una panoramica sul problema

Le diversità nel potere espressivo e nella sintassi emerse dall'analisi precedente inducono a ritenere che lo scambio (od il semplice confronto) di dati XML provenienti da fonti eterogenee risulti difficoltoso. A tal scopo sono stati finora proposti svariati tools di conversione fra i diversi linguaggi descrittivi. Ad esempio XDR_to_XSD è in grado di tradurre schemi XDR nelle corrispondenti strutture XSD, mentre XMLSpy effettua le conversioni degli schemi gestendo un numero predefinito di formati. Nonostante quindi sia possibile ricondurre le rappresentazioni eterogenee di dati XML ad un formato comune e versatile, può sorgere la necessità di tradurre questo formato in uno differente, ad esempio quando l'informazione deve poi essere trasmessa in un forma particolare.
In realtà questo problema si riconduce a quello più vasto e rilevante dello scambio di informazioni strutturate fra contesti (tools od applicazioni) che adottano differenti modelli per la rappresentazione dei dati. Un esempio molto studiato in ambito XML è quello della traduzione delle informazioni in un modello logico (ad es. il modello relazionale o quello ad oggetti), con l'intenzione di memorizzare tali dati in forma permanente. All'interno di questo contesto molto dibattuto dalla comunità scientifica ci si prefigge come obiettivo la definizione di un framework integrato per il trattamento e lo scambio di dati Web (semi) strutturati, descritti mediante formati e modelli differenti.
Il primo passo da affrontare consiste nella precisazione di un meta-formalismo capace di rappresentare sia le primitive fondamentali adottate dai vari linguaggi di descrizione, sia i costrutti di base impiegati dai tradizionali modelli di database. Alla base di questa caratterizzazione si colloca un preciso requisito: qualunque definizione espressa mediante una sintassi specifica può essere ricondotta ad una definizione espressa nel meta-formalismo. Questa proprietà, oltre all'ovvia conseguenza di ricondurre sintassi eterogenee ad un unico formato, costituisce il fondamento delle conversioni fra gli schemi. Un secondo vantaggio fornito dal meta-formalismo è la possibilità di fungere da valido riferimento per un linguaggio di query XML: poiché il meta-formalismo generalizza i vari modelli, il linguaggio di interrogazione potrebbe essere parametrico rispetto al modello scelto per la rappresentazione dei dati.
Il passo successivo e più concreto consiste nella definizione di un metodo di conversione efficiente fra le rappresentazioni che fanno uso del meta-formalismo come livello di riferimento. Nel framework qui definito la traduzione degli schemi e, se necessario, delle corrispondenti istanze, si basa sulla traduzione delle meta-primitive coinvolte. Questo approccio rende ogni traduzione generalizzata ed indipendente dallo specifico modello su cui essa opera. Un altro aspetto interessante deriva dal fatto che un certo numero di proprietà generali delle traduzioni possano essere analizzate sulla base delle proprietà dei modelli coinvolti.

L'ambiente di riferimento

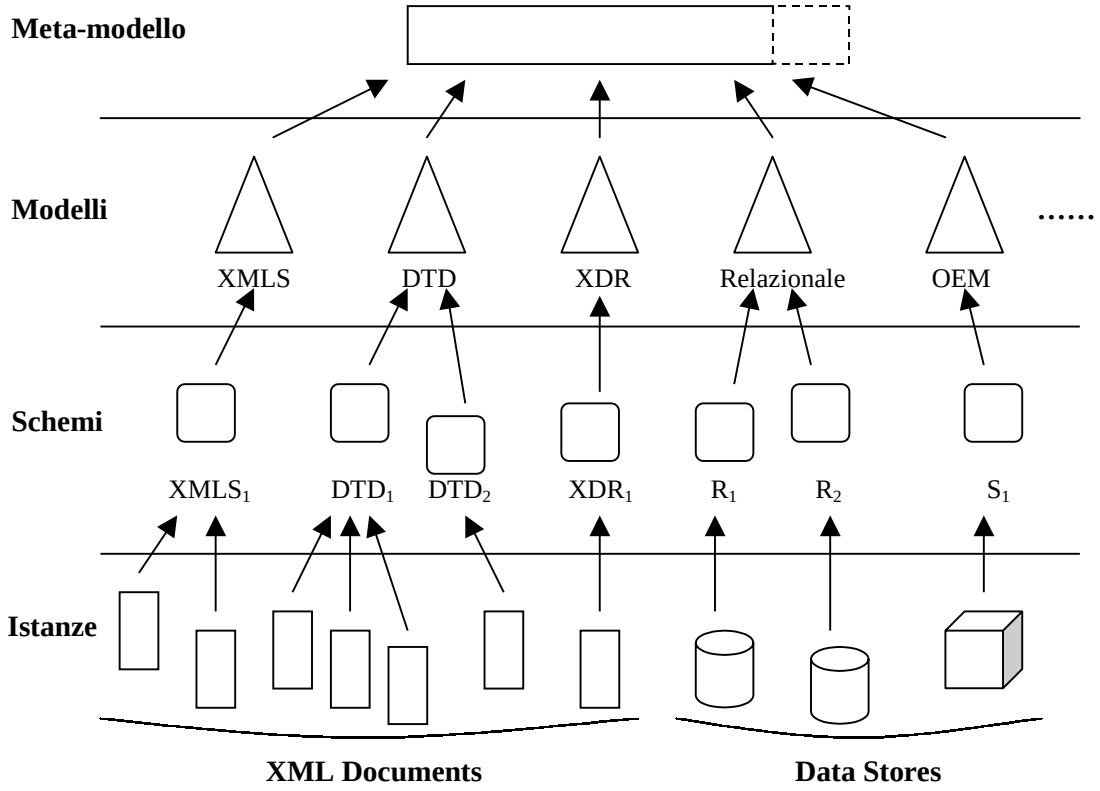
Lo scenario di riferimento consiste nell'identificazione di quattro livelli:

1) Al primo livello si collocano le **istanze** dei documenti ossia i contenitori dei dati correnti. A tal proposito si assume che i dati siano organizzati secondo un qualche formato (semi) strutturato per lo scambio delle informazioni su Web (es: XML).

2) Il secondo livello è caratterizzato dagli **schemi** che descrivono la struttura delle istanze.

3) E' possibile poi raggruppare gli schemi facendo uso di opportuni formalismi che saranno nel seguito definiti **modelli**.

4) Infine a livello più alto è collocato il **meta-modello**, ovvero il formalismo utilizzato nella generalizzazione dei vari modelli.



L'idea in esame non prevede limiti al numero dei modelli supportati: il framework può infatti essere esteso aggiungendo nuovi modelli. E' chiaro che questo approccio potrebbe generare delle carenze espressive da parte del meta-modello: in tal caso è prevista la possibilità di una sua estensione in accordo con le caratteristiche introdotte dalle nuove rappresentazioni.
Nel proseguo si utilizzerà il termine **elemento** per individuare i componenti di uno schema, il termine **primitiva** riguardo ai componenti di un modello ed il termine **metaprimitiva** relativamente ai componenti del meta-modello. Per chiarire il tutto con un esempio, in un documento XML contenente la descrizione di una persona comparirà l'elemento <persona>, composto a sua volta da una collezione non ordinata di sottoelementi (come ad es. <nome> ed <età>). Esisterà poi una primitiva <all> (il tag usato in XML-Schema per identificare una tupla di sottoelementi) ed una corrispondente metaprimitiva <unordered sequence>. Per convenzione si assume che la sintassi "tipo di elemento" identifichi la primitiva utilizzata nella descrizione di un elemento, mentre l'espressione "tipo di primitiva" si riferisca alla metaprimitiva corrispondente ad una primitiva. Nell'esempio precedente il tipo dell'elemento <persona> è la primitiva <all>, il cui tipo a sua volta è la metaprimitiva <unordered sequence>. Va sottolineato che, considerando all'interno del medesimo framework sia i modelli dati tradizionali sia i linguaggi di descrizione dei documenti strutturati, il termine "primitiva" assume un significato abbastanza ampio: può riferirsi infatti al costrutto di un database (come un set od un aggregation) ma può anche rappresentare un vincolo (come ad es. una cardinalità).

L'approccio adottato

Il principio su cui basare le conversioni potrebbe essere il seguente: definiti due modelli M1 (sorgente) ed M2 (obiettivo), ottenuti entrambi grazie al meta-modello, per ciascuna istanza (I1) di uno schema (S1) relativo ad M1 è possibile ottenere un'istanza equivalente (I2) di uno schema (S2) relativo ad M2. In tal caso si dirà che I2 (S2) rappresenta la conversione di I1 (S1) nel modello M2. Alla definizione del framework contribuiscono queste ulteriori considerazioni:

1) Le primitive utilizzate dai più noti linguaggi di descrizione dei dati sono riconducibili ad numero piuttosto ridotto di categorie (basetype, sequenze ordinate, sequenze non ordinate, selezioni, cardinalità e poche altre). Allo stesso tempo molte di queste primitive corrispondono a costrutti ben noti (lexical type, set, sequence, aggregation, disjoint union, set), propri dei modelli dati concettuali. Da ciò discende la possibilità di definire, attraverso un insieme di meta-primitive corrispondenti alle categorie citate, un meta-modello capace di rappresentare adeguatamente i modelli logici dei dati. Pur non trattandosi di un approccio "completo", al quale cioè si possano ricondurre *tutti* i linguaggi di schema esistenti, rappresenta ugualmente un metodo flessibile: alla definizione di un modello caratterizzato da una nuova primitiva corrisponderà l'aggiunta nel meta-modello del relativo tipo.

2) Dal momento che non esiste una definizione precisa riguardo la correttezza di una conversione, si seguirà un approccio pragmatico. Assumendo che le primitive corrispondenti ad una stessa meta-primitiva siano caratterizzate da una semantica comune, le conversioni operanti sulle singole primitive vengono definite nel seguente modo: per ogni primitiva x del modello sorgente a cui non corrisponde una primitiva dello stesso tipo nel modello target M, si tenta di sostituire x con un'altra primitiva, questa dotata di corrispondente in M. Questo processo è basato su di un insieme predefinito di trasformazioni elementari che implementano delle conversioni standard fra le primitive e che vengono considerate corrette per definizione. Ciascuna di queste conversioni opera sia a livello di schema sia a livello di istanza. Nello specifico, ognuna di esse effettua una ristrutturazione dello schema ed include un algoritmo di conversione dei dati che opera in modo coerente con la ristrutturazione eseguita. Per esempio, una sequenza ordinata di tuple si può trasformare in una relazione in cui compaiono le medesime tuple, ognuna delle quali dotata di un elemento aggiuntivo che ne identifica l'ordine. Va sottolineata la possibilità di effettuare conversioni più complesse mediante composizione degli step elementari.

Il meta-modello

Le cognizioni necessarie alla definizione di un meta-modello sono racchiuse nei nuclei di tre linguaggi descrittivi per dati XML: il DTD, XMLSchema ed XDR. A ciò vanno aggiunti due modelli logici spesso utilizzati nel contesto di XML: quello relazionale, basato sui valori, e quello OEM (Object Exchange Model), in rappresentanza dei modelli semi-strutturati. Il meta-modello sarà costituito da un insieme piuttosto limitato di meta-primitive che possono anche essere combinate, formando in tal caso un pattern.

L'elenco completo è costituito da:

1. **Basetype:**
corrispondono a primitive le cui istanze sono costituite da valori stampabili: ad esempio in DTD gli unici basetype disponibili sono i #PCDATA ed i CDATA.

2. **Object type:**
si riferiscono a primitive le cui istanze sono costituite da oggetti identificati mediante un OID univoco. La struttura di tali oggetti viene poi definita per mezzo di altre primitive.

3. **Sequenze ordinate:**
corrispondono a primitive le cui istanze sono sequenze ordinate di altre primitive, definite componenti della sequenza. L'operatore di concatenazione è un esempio di primitiva utilizzata in DTD per definire delle sequenze ordinate.

4. **Sequenze non ordinate:**
si riferiscono a primitive le cui istanze sono sequenze non ordinate di altre primitive, definite anche in questo caso componenti della sequenza. Ad esempio, l'operatore "all" è la primitiva usata per definire una sequenza non ordinata in XML-Schema.

5. **Selezioni:**
si riferiscono a primitive le cui istanze possono essere scelte fra quelle di altre primitive. In DTD ad esempio le selezioni vengono implementate attraverso l'operatore "|".

6. **Cardinalità:**
vengono espresse mediante una coppia di valori (Min,Max) e sono associate a primitive di tipo 3, 4 o 5. In particolare si riferiscono a primitive le cui istanze sono insiemi di istanze delle primitive associate.

7. **Chiavi:**
sono associate a primitive di tipo 3 o 4 e corrispondono a primitive che esprimono l'unicità di un componente della primitiva associata. Un noto esempio di questa metaprimitiva è costituito dal concetto di chiave del modello relazionale.

8. **Chiavi importate:**
sono associate a primitive di tipo 3 o 4 e corrispondono a primitive che esprimono vincoli referenziali fra un componente della primitiva associata e la chiave di un'altra primitiva. Ad esempio XMLSchema utilizza il tag <keyref> per definire primitive di questo tipo.

Definizione del modello

Una volta definito il meta-modello è possibile introdurre la struttura di un qualsiasi modello facendo uso di una sintassi che specifichi:

(1) le primitive offerte dal modello in termini di meta-primitive del meta-modello;
(2) il modo in cui è possibile combinare queste primitive, ovvero i pattern consentiti;
(3) la sintassi utilizzata dal DDL del modello per chiarire l'impiego di tali primitive nella definizione degli schemi.

Inoltre è necessaria anche una funzione con cui assegnare un nome alle primitive ed ai relativi componenti. Come già sottolineato, deve essere sempre garantita l'estensibilità dell'approccio adottato: se ad una primitiva di un modello non corrisponde nessuna metaprimitiva, si assume che sia possibile estendere in modo opportuno il meta-modello.
Per semplicità sono riportati di seguito alcuni esempi informali di definizione di un modello, con riferimento alle meta-primitive descritte nel paragrafo precedente:

- **Modello relazionale:**

Può essere definito grazie ad un insieme ristretto di primitive: il **dominio**, che è un basetype, la **tupla**, che corrisponde ad una sequenza non ordinata di valori propri di un certo dominio, e la **relazione**, che corrisponde all'associazione di una cardinalità (Min=0, Max=N) alla tupla.

- **OEM:**

Anche questo modello si può definire attraverso un numero limitato di primitive: il **tipo atomico** (un basetype del meta-modello), l'**oggetto atomico** (corrispondente all'applicazione del tipo oggetto a quello atomico), e l'**oggetto complesso** che corrisponde all'applicazione del tipo oggetto ad una sequenza non ordinata di oggetti (atomici o complessi).

- **DTD:**

Si può definire una versione semplificata del DTD mediante: due basetype (**#PCDATA** per gli elementi e **CDATA** per gli attributi), tre forme di cardinalità denominate **occorrenze** (*, ? e +), i tipi **ID** ed **IDREF** (usati per definire chiavi e chiavi importate), gli **attributi** (valori di tipo ID, IDREF o CDATA dotati di un identificativo) e gli **elementi**. Un elemento, a cui può essere associata una sequenza non ordinata di attributi, può essere di tipo semplice o composto. Nel primo caso si tratta di un valore di tipo #PCDATA dotato di identificativo, mentre nel secondo si tratta di una selezione o di una sequenza ordinata di sottoelementi.

- **XML Schema:**

Si tratta di un formalismo abbastanza complesso; una versione semplificata comprende: 42 basetype (stringhe, interi, float, ecc.) chiamati **simple type**; una primitiva di cardinalità che consente la specifica dei valori minimi e massimi (chiamati **maxOccurs** e **minOccurs**). Si definiscono poi gli **attributi** (valori con identificatore relativi ad un basetype) e gli **elementi** (tipi semplici o complessi dotati di identificatore). Ad un tipo complesso può essere associata una sequenza non ordinata di attributi ed un "gruppo" di elementi (eventualmente dotati di cardinalità). A sua volta, un gruppo può essere costituito da una sequenza (ordinata), un insieme (sequenza non ordinata) od una selezione. Infine è possibile definire chiavi e chiavi importate sia sugli attributi che sugli elementi, mediante le primitive Key e Keyref.

- **XDR:**

Possiede più o meno le stesse primitive di XML Schema ma utilizza una sintassi differente. Ad esempio la sequenza ordinata è chiamata "seq" mentre la selezione si definisce "one".

Conversioni fra modelli

Come specificato nel paragrafo introduttivo l'obiettivo finale è la definizione di un sistema in grado di eseguire conversioni di schemi e istanze: sorgente e destinazione delle conversioni sono rappresentate da modelli che si richiamano al meta-modello.
E' necessario un passo preliminare consistente nella definizione di una **relazione di inclusione** (<=) fra due modelli. Si tratta intuitivamente di una relazione d'ordine parziale basata su:

1) Dominio rappresentato dall'insieme dei pattern disponibili nei vari modelli;
2) Definizione di un ordine parziale su tali pattern;

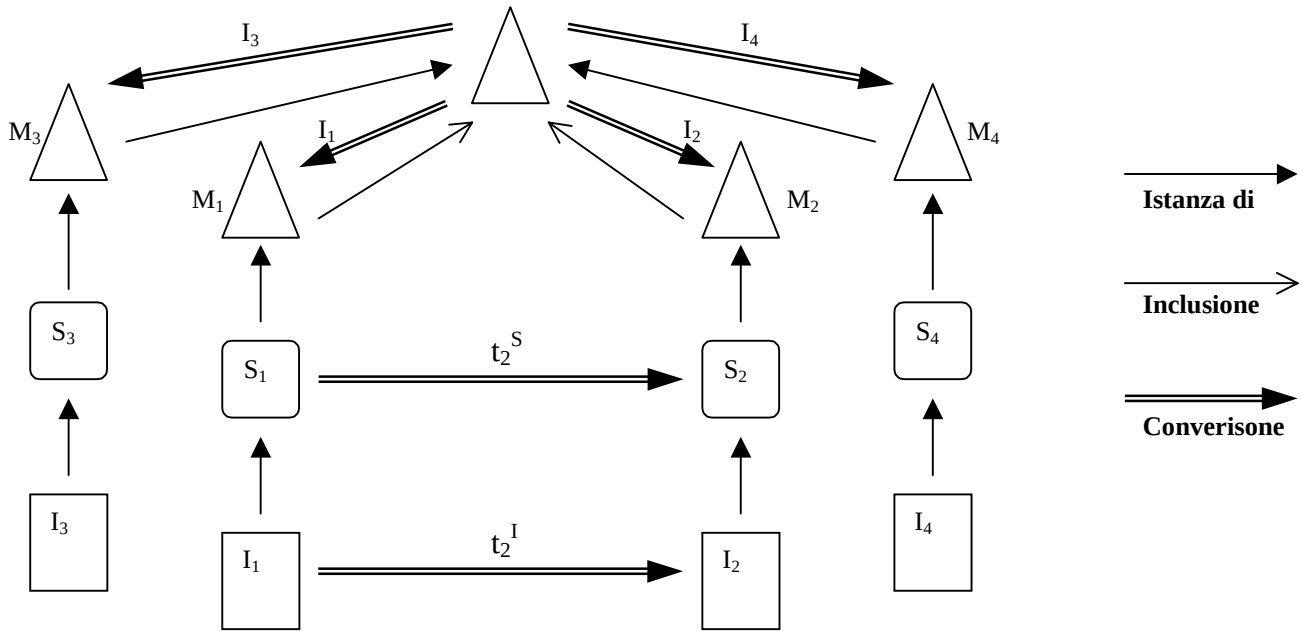
La relazione consente di affermare che, ad esempio, un pattern del DTD in cui compare una primitiva di cardinalità (nello specifico: ? (0,1), + (1,N) , * (0,N)) è incluso nel corrispondente pattern di XML Schema (in cui minOccurs e maxOccurs possono assumere qualunque valore).
Sfruttando questa relazione è possibile introdurre una nozione di meta-modello, utilizzato poi come riferimento per le conversioni. In modo intuitivo si tratta di una struttura che include una rappresentazione per qualunque pattern appartenente ad uno dei modelli iniziali. Le conversioni si basano allora sui seguenti principi:

- Conversione delle primitive di base nei rispettivi modelli. Più precisamente, per ogni primitiva P del modello sorgente M (rappresentata dalla metaprimitiva P) a cui non corrisponde una primitiva P' (dello stesso tipo di P) nel modello target M', occorre definire una rappresentazione di P mediante altra primitiva ammissibile in M'.

- Scomposizione del processo di conversione: una prima parte riguarda il passaggio dal modello sorgente al meta-modello, la seconda riguarda invece la traduzione dal meta-modello al modello target. Il primo passo è banale poiché, per definizione, qualunque schema di un modello è uno schema del meta-modello. Il passo successivo si realizza convertendo gli elementi dello schema sorgente (il cui tipo è privo di corrispondenza nel modello target) in elementi di tipo ammissibile. Grazie a questo approccio definire la conversione fra il meta-modello ed i singoli modelli è sufficiente per implementare una conversione fra due modelli qualunque. Segue che il numero delle conversioni necessarie è lineare nel numero dei modelli invece che quadratico, come si avrebbe nel caso in cui si definisse una conversione per ciascuna coppia di modelli.

- Dal momento che il numero dei costrutti è limitato si possono realizzare alcune conversioni predefinite, che possono poi essere utilizzate per comporre conversioni più complesse.

La seguente figura mostra una visione d'insieme del framework in cui sono indicate tutte le conversioni dal meta-modello ai singoli modelli. Una generica conversione t è basata su due componenti: una conversione t^S che opera sugli schemi ed una corrispondente conversione t^I che opera sulle istanze. Quindi, la conversione dell'istanza I1 (dello schema S1) nel modello M2, fa uso delle componenti t_2^S e t_2^I per ottenere rispettivamente lo schema S2 e l'istanza I_2.



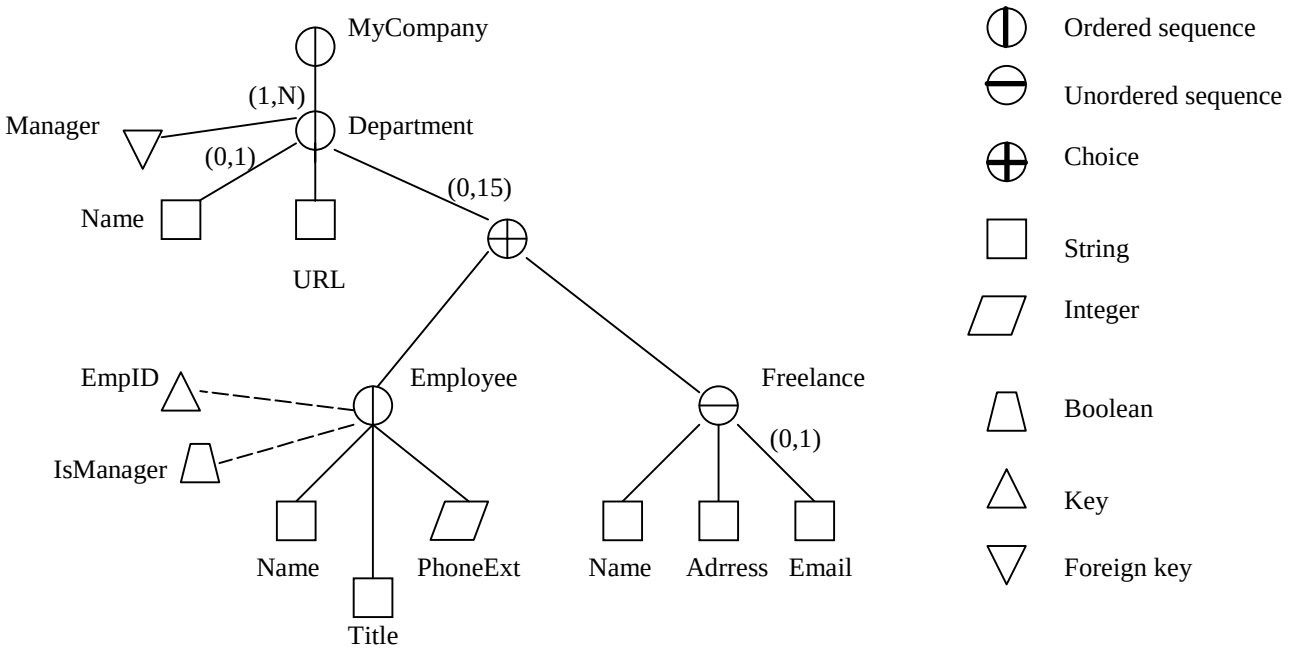
Un esempio di conversione

Come si realizza la conversione di una struttura XML-Schema in una struttura DTD od in uno schema relazionale? Dato il seguente XML-Schema:

```
<xsd:schema>
    <xsd:element  name = “My Company”>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element  name = “DepartmentType”  maxOccurs = “unbounded” />
            </xsd:sequence>
        </xsd:complexType>
        <xsd:key  name = “EmpKey”>
            <xsd:selector  xpath = “Department/Employee” />  <xsd:field  xpath = “@EmpID” />
        </xsd:key>
        <xsd:keyref  name = ”ManFK”  refer = “Empkey”>
            <xsd:selector  xpath = ”Department” />  <xsd:field  xpath = “Manager” />
        </xsd:keyref>
    </xsd:element>
    <xsd:complexType  name = ”DepartmentType”>
        <xsd:sequence>
            <xsd:element  name = “DepName”  type = “xsd:string” />
            <xsd:element  name = “URL”  type = “xsd:string” />
            <xsd:element  name = “Manager”  type = “xsd:string”  minOccurs =”0” />
            <xsd:choice  maxOccurs = “15”>
                <xsd:element  name = “Employee”  type = “EmployeeType” />
                <xsd:element  name = “Freelance”  type = “FreelanceType” />
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType >
```

```
<xsd:complexType  name = ”EmployeeType”>
        <xsd:sequence>
                <xsd:element  name = “Name”  type = “xsd:string” />
                <xsd:element  name = “Title”  type = “xsd:string”  minOccurs=”0” />
                <xsd:element  name = “PhoneExt”  type = “xsd:integer” />
        </xsd:sequence>
        <xsd:attribute  name = “EmpID”  type = “xsd:string”  use = “requred” />
        <xsd:attribute  name = “IsManager”  type = “xsd:boolean”  use = “optional” />
</xsd:complexType >
<xsd:complexType  name = ”FreelanceType”>
        <xsd:all>
                <xsd:element  name = “Name”  type = “xsd:string”  minOccurs = “0” />
                <xsd:element  name = “Address”    type = “xsd:string”  minOccurs = ”0” />
                <xsd:element  name = “Email”  type = “xsd:string” minOccurs = ”0”  />
        </xsd:all>
</xsd:complexType >
<xsd:schema>
```

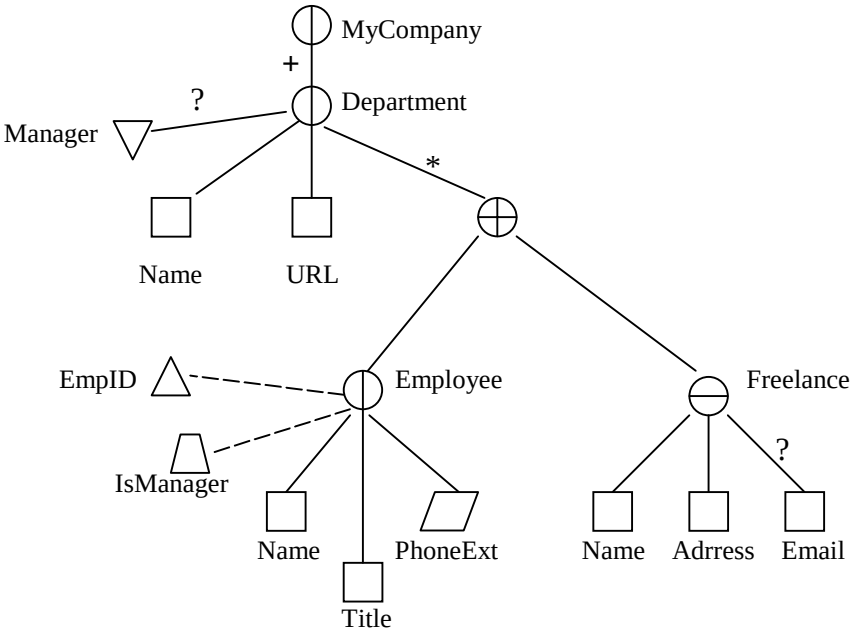
Nell’ambito del meta-modello questo codice XML-Schema può essere rappresentato mediante il seguente grafico (in cui le linee tratteggiate indicano gli attributi):



La conversione di questo schema in un DTD si ottiene attraverso le seguenti trasformazioni di primitive elementari:

1) Tutti i basetype vengono convertiti in stringhe.
2) Le sequenze non ordinate sono convertite in sequenze ordinate.
3) Le cardinalità vengono semplificate, in accordo con i limiti imposti dal DTD.
4) Le chiavi primarie vengono convertite in attributi di tipo ID.
5) Le chiavi importate sono convertite in attributi di tipo IDREF.

Applicando queste operazioni si perviene al seguente schema, in cui si fa ancora uso della notazione del meta-modello:



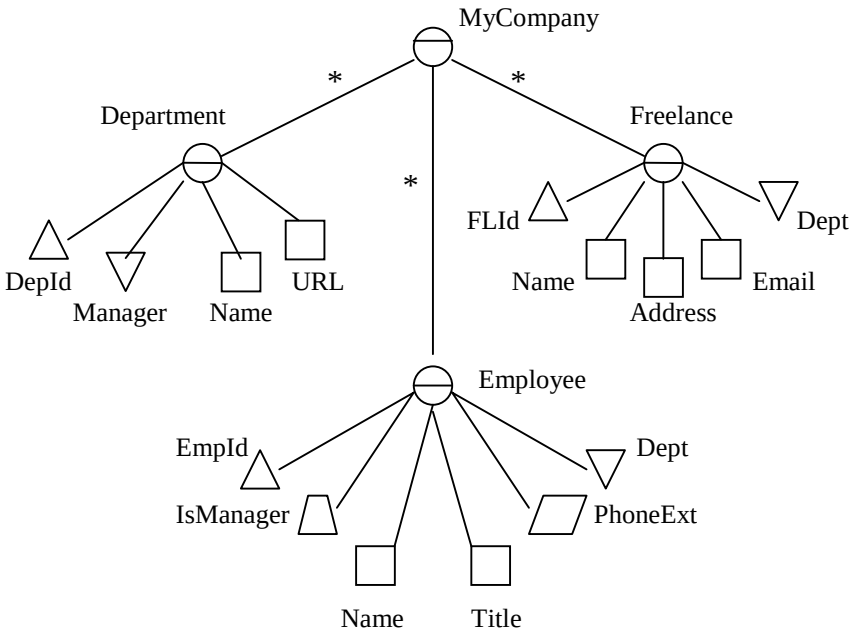
Ad essa corrisponde la seguente rappresentazione in DTD:

```
<!ELEMENT   MyCompany  (Department +) >
<!ELEMENT   Department   (DepName, URL, (Employee | Freelance) +) >
<!ATTLIST    Department    Manager  IDREF  #IMPLIED >
<!ELEMENT   DepName      (#PCDATA) >
<!ELEMENT   URL          (#PCDATA) >
<!ELEMENT   Employee     (Name, Title ?, PhoneExt) >
<!ATTLIST    Employee     EmpID  ID  #REQUIRED  IsManager  CDATA  #IMPLIED >
<!ELEMENT   Freelance    (Name ?,  Address ?,  Email ?) >
<!ELEMENT   Name         (#PCDATA) >
<!ELEMENT   Title        (#PCDATA) >
<!ELEMENT   PhoneExt     (#PCDATA) >
<!ELEMENT   Address      (#PCDATA) >
<!ELEMENT   EMail        (#PCDATA) >
```

A livello di istanze questa conversione richiede semplicemente di trasformare gli elementi che includono chiavi importate in attributi, dal momento che in DTD soltanto gli attributi possono essere di tipo IDREF.
In modo del tutto simile la conversione della stessa struttura in uno schema relazionale si può ottenere attraverso le seguenti trasformazioni:

1) Eliminazione delle selezioni;
2) Conversione delle sequenze ordinate in sequenze non ordinate;
3) Risoluzione delle sequenze non ordinate innestate;
4) Semplificazione delle cardinalità;
5) Aggiunta di chiavi primarie alle sequenze che ne sono prive;
6) Fusione degli attributi e degli elementi.

Seguendo questi passi si perviene ad uno schema del tipo:



Utilizzando la notazione tradizionale si otterrebbe il seguente schema relazionale:

- **Department (DeptID, Manager, Name, URL)**
- **Employee (EmpID, IsManager, Name, Title, PhoneExt, Dept)**
- **Freelance (FlID, Name, Address, Email, Dept)**

A livello di istanze questa conversione risulta più complessa e richiede la trasformazione di un documento XML in un insieme di relazioni conformi alla conversione effettuata a livello di schema.

Proprietà delle conversioni

Le conversioni così definite presentano un certo numero di proprietà interessanti:

1) Un primo requisito che caratterizza una conversione è la **validità**, ossia la proprietà tale per cui lo schema (o l'istanza) ottenuto risulti valido per il modello target. Inoltre si dovrebbe preservare una nozione di **equivalenza**. Fra le varie nozioni proposte in letteratura, una condizione necessaria (ma non sufficiente) per l'equivalenza fra due schemi è la corrispondenza 1:1 fra i rispettivi insiemi di istanze.

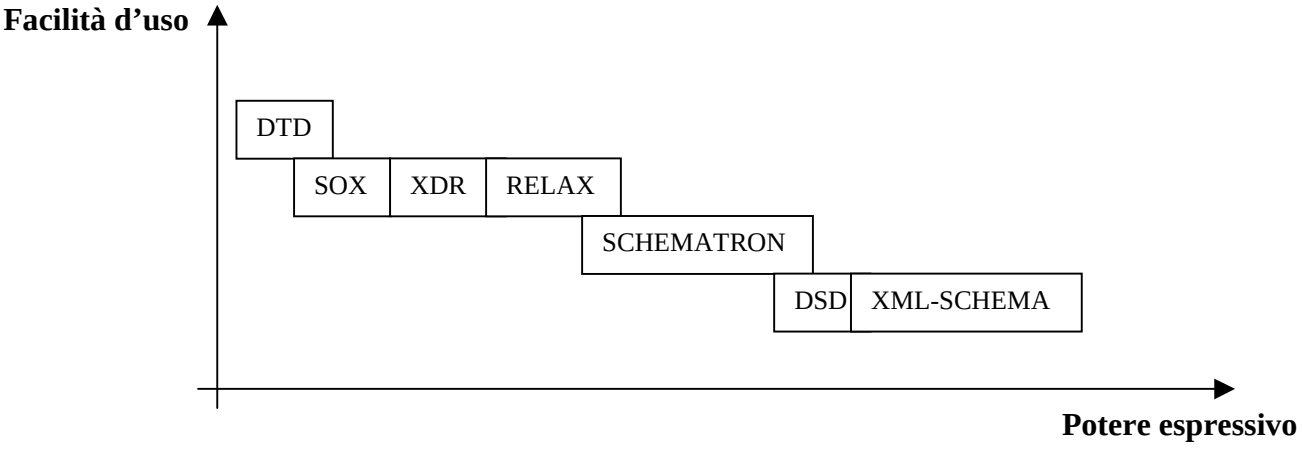
2) Esistono poi coppie di modelli tali che per ogni schema sorgente è sempre possibile trovare uno schema target equivalente: ciò è vero ad esempio per XDR ed XML-Schema. Tuttavia questo non è vero in generale: è ovvio che l'equivalenza non può essere garantita qualora il modello sorgente permetta una rappresentazione più dettagliata rispetto a quello target. Ad esempio, se il sorgente è un XML-Schema che associa cardinalità quali (1,10) o (5,N) mentre il target è un DTD (in cui sono ammesse soltanto cardinalità di tipo (0,1), (1,N) e (0,N)) è chiaro che non possono esistere schemi equivalenti. In questi casi si parla di ***perdita*** del contenuto informativo nel processo di conversione.

3) In altri casi è possibile determinare uno schema target equivalente per ciascun schema sorgente, ma l'informazione riportata nello schema target è meno ricca di quella che compare nel corrispondente schema di partenza. E' noto infatti che, dato uno schema sorgente nel formato XML-Schema, è possibile determinare uno schema relazionale ad esso equivalente ma privo della stessa semantica. In questi casi è quasi impossibile operare le traduzioni inverse. Nonostante il contenuto informativo sia equivalente, si dice che l'informazione subisce un **degrado**. Facendo un parallelo con la termodinamica si potrebbe affermare che l'energia in questo caso si conserva mentre l'entropia aumenta, senza alcuna possibilità di diminuzione futura.

Sia nel caso di perdita del contenuto informativo sia in quello di degrado esistono diverse possibilità di conversione degli schemi, con caratteristiche differenti a seconda del contesto.

Conclusioni

Sebbene la maggior parte del contenuto disponibile su Internet sia stato progettato per essere letto da essere umani e non per essere manipolato dai calcolatori, è opinione diffusa che questo scenario sia in rapida evoluzione. Tim Berners-Lee, da più parti considerato il padre fondatore del World Wide Web, ritiene che siamo ormai prossimi ad una nuova era, quella del "**Web semantico**". Alla base di questa svolta si pone l'idea di progettare e distribuire dati in forma strutturata il cui utilizzo non sia limitato a fini di semplice visualizzazione, ma preveda anche l'automatizzazione, l'integrazione ed infine il riutilizzo dei dati stessi nel contesto di applicazioni differenti. Come ogni miglioramento nel rapporto uomo-macchina, l'adozione di una nuova tecnologia richiede l'affermazione di uno standard: a questo proposito il W3C opera da tempo per unificare gli sforzi della comunità di ricerca attorno alla sigla dell'XML.
In questo lavoro sono state illustrate le caratteristiche peculiari del nuovo linguaggio, evidenziandone potenzialità e limiti. L'intuizione di vero spessore che ha caratterizzato SGML prima ed XML poi è stata l'adozione del markup generalizzato. Tutto ciò presuppone l'adozione di un modello logico del documento caratterizzato da una struttura gerarchica che, pur non adattandosi alla rappresentazione di qualunque concetto, ben si addice al trattamento elettronico di gran parte del contenuto informativo presente e futuro. A questo va aggiunto il plusvalore legato alla estensibilità, che fa di questo strumento un metalinguaggio capace di definire vocabolari specifici per le singole applicazioni. La strutturazione dei documenti conferisce ad XML una doppia natura: quella di un insieme di regole per la produzione di testi ben formati e quella di una serie di specifiche per la definizione degli schemi dei documenti. Il secondo aspetto, rappresentato in astratto dal Document Type Definition, si concretizza mediante molteplici sintassi, legate ad altrettanti linguaggi per la definizione degli schemi. A tal proposito si è operata un'analisi comparativa di sette produzioni ritenute tra le più significative fra quelle a tutt'oggi proposte. Prescindendo dagli aspetti specifici, l'analisi del seguente grafico conduce ad una considerazione conclusiva: alla facilità d'uso della sintassi DTD si contrappone l'espressività di linguaggi più raffinati quali XML-Schema, DSD e Schematron.



La proliferazione di queste produzioni pone diversi problemi in ordine alla possibilità di scambiare o confrontare dati provenienti da fonti eterogenee. Una delle proposte avanzate in questo ambito dalla comunità scientifica consiste in un framework basato sulla definizione di un meta-modello. La ragione di questo approccio si riflette nella volontà di ridurre il numero di conversioni fra le primitive necessario per effettuare la traduzione di uno schema. Anziché una conversione per ciascuna coppia di

modelli, l’uso del meta-modello rende sufficiente un numero di conversioni lineare nel numero dei modelli, semplificando notevolmente il processo di traduzione e limitando l’overhead di sistema.
Un’ultima nota riguarda l’attuale grado di utilizzo delle possibilità offerte da XML. Si può affermare che per molti versi XML sia uno strumento ancora non adeguatamente sfruttato: la comunità Web necessita ancora di un po’ di tempo per comprendere a fondo i punti di forza (ed i limiti) di questa tecnologia. Ciò nonostante XML rappresenta già a pieno titolo un elemento fondamentale per la realizzazione delle applicazioni di prossima generazione.

Bibliografia

1) Charles Goldfarb – Paul Prescod "**XML**" pp.562 Ed. McGraw-Hill 1999

2) Elliotte Rusty Harold "**XML Bible**" pp.871 Ed.Tecniche Nuove 2000

3) Lee Ann Philips "**Usare XML**" pp.811 Ed.Mondadori Informatica 2000

4) Angela Bonifati – Dongwon Lee "**Technical Survey of XML Schema and Query Languages**" pp.38

5) Dongwon Lee – Wesley W.Chu "**Comparative Analysis of six Schema Languages**" pp.12

6) Riccardo Torlone – Paolo Atzeni "**Management and Translation of XML Database**" pp.10

7) Anna Stefani "**E venne l'ora della Rete semantica**" E-business Trade Ed.Mondadori Giu2001 pp.4

Sitografia

1) World Wide Web Consortium: www.w3c.org

2) The Web distributed data exchange: www.xml.com

3) XML community: www.xmlhack.com