

**UNIVERSITA' DEGLI STUDI DEL PIEMONTE ORIENTALE
SEDE DI ALESSANDRIA - FACOLTA' DI SCIENZE M.F.N.**

CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

**SVILUPPO DI XMLSTUDIO:
EDITOR INTEGRATO PER DOCUMENTI XML**

**Sviluppo di un editor Java per l'inserimento di informazioni
semantiche a documenti testuali secondo lo standard del linguaggio XML**

**Tesi di Laurea
di Paolo Guagliumi
pguagliumi@libero.it**

**Relatore
Prof.ssa Paola Giannini
giannini@di.unito.it**

**Controrelatore
Prof.Luigi Portinale
portinal@mfu.unipmn.it**

I SESSIONE - ANNO ACCADEMICO 2001/2002

DEDICA DELLA TESI
A MIA MADRE E ALLA MIA FAMIGLIA

INDICE

INTRODUZIONE	8
Organizzazione della tesi:.....	10
CAPITOLO 1 - DEFINIZIONI, SPECIFICHE ED IMPORTANZA DELL'EXTENSIBLE MARKUP LANGUAGE	11
1.1 – La codifica informatica dei testi	11
1.2 – I dati sul Web e nei database	12
1.2.1 – La cultura database	13
1.2.2 – La possibilità di una convergenza	13
1.2.3 – Il formato dei documenti	17
1.3 – Lo Standard Generalized Markup Language	18
1.3.1 – La necessità di un nuovo linguaggio per il Web	20
1.4 - XML: il nuovo linguaggio per il Web	23
1.4.1 – Applicazioni di XML	24
1.4.2 – Il Parser ed il Processore	25
1.4.3 – La Document Type Declaration	26
1.4.4 – La Document Type Definition in XML (DTD)	26
1.4.5 – Gli elementi	29
1.4.6 – Gli attributi	31
1.4.7 – Le entità	33
1.4.7.1 – Le entità predefinite.....	33
1.4.8 – Un esempio completo di documento XML e DTD	34
1.4.9 – Simboli relativi alla struttura di una dichiarazione	35
1.4.10 – I Namespace	36
1.4.11 – XSL: la rappresentazione	36

CAPITOLO 2 - ANALISI DI JAVA & XML	38
2.1 – Java e XML: i linguaggi del momento	38
2.2 – L'utilità di Java per l'uso di XML	39
2.3 – L'operazione di Parsing XML in Java.....	40
2.3.1 – Parser DOM e SAX.....	41
2.3.2 – Introduzione ai Parser disponibili.....	42
2.3.3 – Come scegliere un Parser?	43
2.3.4 – I principali Parser sul mercato.....	45
2.3.5 – Confronto tra i vari Parser esistenti	47
2.3.6 – La scelta di Sun Project X (o Crimson) per XMLStudio	49
CAPITOLO 3 - L'EDITOR INTEGRATO XMLSTUDIO	51
3.1 – L'idea alla base di XMLStudio	51
3.1.1 – Che cos'è un documento RFC.....	51
3.1.2 – Aggiungere conoscenza ai documenti RFC	52
3.1.3 – JRfc: prima bozza di XMLStudio	53
3.1.4 – Il progetto Xdidattica.....	53
3.1.5 – Le possibilità d'impiego di XMLStudio	54
3.2 – Installazione ed uso di XMLStudio	55
3.2.1 – L'interfaccia di XMLStudio e le sue componenti	56
3.2.1.1 – La barra dei menù.....	56
3.2.1.2 – L'area di elaborazione del testo	57
3.2.1.3 – L'albero che mostra il documento XML.....	57
3.2.1.4 – Il pannello che consente di definire e modificare i tag	58
3.2.1.5 – I menù a tendina	59
3.2.2 – L'uso di SAX in XMLStudio	59
3.2.3 – L'uso di DOM in XMLStudio.....	60
3.2.4 – Alcune caratteristiche del codice sorgente	67

3.3 – JavaHelp come manuale utente	69
3.3.1 – Le caratteristiche di JavaHelp	71
3.4 – L’uso pratico di XMLStudio	71
3.4.1 – Creazione di uno specifico foglio di stile CSS associato	76
3.5 – Esempi d’uso di XMLStudio	79
3.6 – Organizzazione dei file e directory	79
3.7 – Diagramma delle classi di XMLStudio	80
CAPITOLO 4 - CONFRONTO CON ALTRI STRUMENTI E PROGETTI..85	
4.1 – Gli editor XML sul mercato.....	85
4.1.1 – Microsoft XML Notepad.....	85
4.1.2 – Peter’s XML Editor	87
4.1.3 – XMLEditPro	88
4.1.4 – Cooktop	89
4.1.5 – TXE The XML Editor	90
4.1.6 – XMLOperator	91
4.1.7 – Altri Editor XML.....	91
4.2 – Due approcci differenti per l’editing XML	92
4.3 – L’editor Visual_XML	93
4.3.1 – Creazione visuale di file XML	93
4.3.2 – Creazione visuale di file XSL	94
4.3.3 – Creazione automatica di file DTD.....	95
4.4 – Confronto tra XMLStudio e Visual_XML.....	96
CAPITOLO 5 - CONCLUSIONE E SVILUPPO DI XMLSTUDIO	97
5.1 – Lo sviluppo di XMLStudio.....	97
5.1.1 – Internazionalizzazione di XMLStudio	97

5.1.2 – Fogli di stile XSL	98
5.1.4 – Testo colorato	99
5.1.5 – Seleziona paragrafo tag	100
5.2 – Malfunzionamenti e problemi rilevati	100
5.2.1 – Problema legato al word-wrap	100
5.3 – Conclusioni	101
APPENDICE.....	103
Appendice A.....	103
Appendice B.....	106
Esempio di articolo informatico	107
Articolo.xml	107
Articolo.dtd	121
Articolo.css.....	123
Visualizzazione di Articolo.xml nel browser.....	123
XMLStudio.xml	124
Esempio di articolo medico.....	126
Medicina.xml.....	126
Medicina.dtd.....	128
Medicina.css.....	128
Visualizzazione di Medicina.xml nel browser.....	129
Medicinatag.xml.....	130
Esempio di articolo storico.....	131
Storia.xml	131
Storia.dtd	137
Storia.css	138
Visualizzazione di Storia.xml nel browser.....	139
Storiatag.xml	140
BIBLIOGRAFIA	141

BIBLIOGRAFIA WEB.....	142
HOME PAGE DI XMLSTUDIO	142
RINGRAZIAMENTI	143

Introduzione

Internet può essere vista come una rete di librerie geograficamente distribuite e multimediali, interconnesse tra di loro: per poter utilizzare in modo efficace l'informazione in essa contenuta, occorre una riflessione sui *documenti*, che sono il veicolo primo di accesso e trasmissione dell'informazione e sul loro ruolo nella disponibilità di conoscenza. Essi forniranno tanta più conoscenza quanto meglio saranno organizzati.

I motori di ricerca mondiali iniziano a manifestare inefficienze legate alle tecniche di indicizzazione e ai loro limiti intrinseci, per cui nasce l'esigenza di modificare alla radice il modo con cui si organizzano i documenti.

L'obiettivo della tesi è legato alla realizzazione di un'applicazione che permette di aggiungere informazione semantica ai documenti testuali. Tale informazione è realizzata aggiungendo dei marcatori¹ che descrivono il significato di un determinato paragrafo, porzione di testo o parola. I documenti etichettati in questo modo acquisiscono ulteriore conoscenza semantica e strutturale, pur mantenendo inalterato il loro contenuto.

“Il principio è semplice: più informazione viene aggiunta ad un documento e un miglior uso di esso ne può derivare”².

La marcatura è realizzata con l'utilizzo del linguaggio XML (Extensible Markup Language) che permette la definizione di tag da parte dell'utente, che non è così confinato ad uno schema predefinito.

¹ Nella tesi i vocaboli *marcatore*, *tag*, *etichetta* hanno lo stesso significato e si riferiscono a sequenze di caratteri delimitate dai segni < e > come ad esempio <nome> ... </nome>.

² Jon Bosak – Sun Microsystems (frase estratta da un articolo pubblicato su Nature).

Procedendo sistematicamente nell'analisi e generazione di documenti secondo questa metodologia di marcatura, si rendono più efficienti le ricerche in due modi³ distinti:

- Il primo è relativo al punto di vista dell'utente che limita la ricerca ad un certo numero di pagine con caratteristiche specifiche.
- Il secondo è inerente all'apertura di grandi basi di dati attualmente chiuse⁴ ai robot di ricerca che indicizzano il Web, con la possibilità di ricerche specifiche nei linguaggi⁵ di markup specializzati, nati dal metalinguaggio XML.

Le possibilità di accrescimento della conoscenza intesa come patrimonio comune, insite nell'uso di questa metodologia, sono non indifferenti, perché sia grazie all'azione individuale di inserimento di ulteriore conoscenza nei documenti, che alla collaborazione globale delle persone che ne deriva mettendo a disposizione di altri i propri documenti XML così arricchiti, è possibile poter usufruire meglio di quel patrimonio comune all'umanità intera che è *il sapere*, da sempre trasmesso principalmente attraverso documenti.

Lo strumento realizzato in questa tesi, XMLStudio è un'applicazione, scritta in linguaggio Java, che permette di manipolare documenti testuali aggiungendo, rimuovendo tag, fornendo schemi (DTD) che potranno essere utilizzati per confrontare l'etichettatura dei documenti. Il tutto realizzato consentendo all'utente la possibilità di navigare anche in modo strutturale nel documento.

Il linguaggio Java è stato scelto per il suo ottimo connubio con XML e per la sua portabilità che consente al programma di essere eseguito su diversi sistemi operativi: Java è un linguaggio indipendente dalla piattaforma, XML è un modo

³ W.Wayt Gibbs – The Web Learns to Read (Scientific American - 1998).

⁴ Ad esempio la FIAT (o qualsiasi altra casa automobilistica) potrebbe permettere l'indicizzazione da parte dei motori di ricerca di un suo catalogo di componenti e pezzi di ricambio per le automobili.

⁵ Tra i linguaggi derivati dal metalinguaggio XML vi sono il *Chemical Markup Language* (CML) che mostra graficamente la struttura molecolare dei componenti descritti in pagine Web CML; il *Math Markup Language* (MathML) che rende immediata la visualizzazione di espressioni matematiche complesse nel browser: prima della creazione di questo linguaggio l'unico modo per visualizzare formule matematiche nel browser era salvarle come immagini mentre oggi vi è la possibilità di visualizzare equazioni con poche righe di testo; *MusicML* un linguaggio musicale costituito da un insieme di etichette per note, tempi e altri elementi, memorizzato come testo ma che nel browser viene visualizzato come pentagramma.

di rappresentare i dati in modo indipendente dalla piattaforma per cui Java ed XML sono complementari per loro natura, specialmente in applicazioni legate ad internet.

Molti editor XML sono presenti sul mercato (alcuni di essi sono gratuiti mentre altri a pagamento) ma l'approccio di trasformazione di documenti secondo la metodologia di XMLStudio è seguita solo da un altro editor italiano relativo ad un progetto di nome Xdidattica che verrà analizzato nella tesi.

L'utente che desiderasse imparare ad usare XMLStudio è aiutato nell'apprendimento dello stesso da un help, in formato JavaHelp, sensibile al contesto. Alcuni esempi, inclusi nella directory *esempi* di XMLStudio e riportati nella *Documentazione di XMLStudio*, permettono di rendersi conto di quale sia l'uso concreto del programma.

Organizzazione della tesi:

- Il *primo capitolo* è un'introduzione ad aspetti dell'XML utilizzati durante la progettazione dell'applicazione XMLStudio; vengono anche esposti dei concetti di base utili per il suo utilizzo.
- Il *secondo capitolo* illustra l'uso dei linguaggi Java e XML, che si integrano costituendo un ottimo connubio per la realizzazione di applicazioni.
- Il *terzo capitolo* mostra nel dettaglio l'editor integrato XMLStudio, specificando l'idea alla base della sua realizzazione ed alcuni elementi utili a comprenderne l'implementazione.
- Il *quarto capitolo* prende in esame altri editor XML e li confronta con XMLStudio per far emergere la differenza di approccio di base che essi hanno.
- Il *quinto capitolo* elenca alcuni possibili sviluppi futuri del programma XMLStudio e riporta le conclusioni del lavoro di questa tesi.

CAPITOLO 1

Definizioni, specifiche ed importanza dell'eXtensible Markup Language

1.1 – La codifica informatica dei testi

Con l'avvento dell'informatica e l'effettiva realizzazione degli elaboratori, è scaturita la possibilità di codificare, manipolare, gestire i testi in formato elettronico in modo rapido. Per “codifica informatica” si intende la rappresentazione dei testi su di un supporto digitale in un formato utilizzabile da un computer. Dall'arrivo di Internet, una mole sempre più consistente di informazione viene riportata sul World Wide Web⁶ al fine di essere pubblicamente disponibile. Questa informazione è contenuta in pagine HTML⁷. La dimensione dell'informazione contenuta nel Web continua ad aumentare e per poterla utilizzare in modo efficace, nasce l'esigenza di

⁶ Creato al Cern di Ginevra nel 1992, è un sistema basato su ipertesti per accedere alle risorse su Internet e organizzarle. Il WWW è accessibile tramite appositi programmi di navigazione, i browser (i più diffusi sono Netscape Navigator, Microsoft Internet Explorer e Opera, reperibili gratuitamente). Ha avuto un tale successo negli ultimi anni che molti erroneamente identificano la rete con il World Wide Web. Il suo grande merito, oltre alla possibilità di presentare l'informazione con testo, immagini, video e audio, è quello di aver dato un grande impulso alla diffusione di Internet nel mondo.

⁷ HTML (HyperText Markup Language): linguaggio di marcatura ipertestuale per la costruzione di pagine pubblicate nel World Wide Web. L'HTML è stato originariamente sviluppato da Tim Berners-Lee durante la sua permanenza al CERN ed è inizialmente diventato popolare grazie al browser Mosaic sviluppato dalla NCSA. Nel corso degli anni '90 è prosperato con la crescita esponenziale del Web.

organizzarla. Gran parte dei dati ha intrinsecamente meno struttura perché proviene da basi di dati (in cui l'informazione è molto strutturata) ma la sua pubblicazione in HTML rimuove questa informazione originaria collassandola nel semplice markup possibile con i suoi tag.

Si pensi ad esempio all'operazione più frequentemente svolta in internet: l'uso di un motore di ricerca per trovare delle informazioni. I documenti testuali, se analizzati, evidenziano delle regolarità nella loro struttura. Il problema legato all'indicizzazione dei documenti, nasce dalla difficoltà nell'estrarre risultati rilevanti da file che hanno un limitato contenuto semantico: i documenti sono sequenze di caratteri nei quali quasi sempre è assente il significato effettivo delle singole parole. Algoritmi complessi possono cercare di interpretare il senso di paragrafi considerando il contesto e la vicinanza con cui certi vocaboli compaiono. Ma l'informazione semantica insita nel documento è spesso ridotta.

XML nasce con il preciso intento di colmare anche queste lacune: è rilevante l'importanza che sta acquistando come standard per la rappresentazione di informazioni in Internet perché facilita la pubblicazione di dati in formato elettronico attraverso l'uso di una sintassi facilmente leggibile sia da persone che dalle macchine. XML definisce quindi nuove prospettive e possibilità d'uso espresse in un linguaggio da considerare un futuro standard per lo scambio di dati semistrutturati e strutturati attraverso sistemi computerizzati.

1.2 – I dati sul Web e nei database

Il Web fornisce un semplice, universale standard per lo scambio di informazione. Ruota attorno al principio di decomposizione dei dati in unità che possono essere trasmesse e alle quali sono stati assegnati dei nomi: i documenti. Il successo del Web è strettamente legato allo sviluppo di HTML (Hypertext Markup Language), il linguaggio con cui vengono scritti i documenti visualizzati dai browser. HTML descrive sia la struttura interna del documento ovvero la disposizione ed il formato del testo, sia la struttura per il

collegamento con altri documenti (link)⁸. L'introduzione di http⁹ come standard e l'uso di HTML per la composizione, costituiscono la radice dell'universale accettazione del Web come mezzo per lo scambio di informazione.

1.2.1 – La cultura database

Coloro che utilizzano i sistemi database usano un linguaggio legato a schemi per database relazionali e diagrammi entità-relazione per descrivere le strutture. Il meccanismo che permette di condividere l'informazione è molto differente: in particolare si usano linguaggi di query per accedere all'informazione e meccanismi per il controllo della concorrenza e di recovery per preservare l'integrità della struttura dei dati. Inoltre la vista logica, o astratta, del database è separata dalla sua implementazione fisica. La prima è necessaria per comprendere ed interrogare i dati. La seconda è importante per l'efficienza.

La rigidità imposta nei DB dell'uso di schemi, per cui prima di usare e popolare il DB ne deve definire la struttura, è stata riconosciuta come un problema che ne limita la flessibilità. A questo proposito sono state formulate proposte di organizzazione dei dati quali quelle dei dati semistrutturati nella quale viene eliminato il concetto di schema. I dati sono memorizzati unitamente alle informazioni che ne permettono l'utilizzo. Inoltre il formato dei dati è reso più flessibile permettendo di avere attributi ai quali è assegnato più di un valore.

1.2.2 – La possibilità di una convergenza

Un primo passo per la convergenza tra questi due approcci per lo scambio di informazione è rappresentato da XML che è come HTML un

⁸ Un ipertesto è un testo elettronico in cui alcune parole o insiemi di parole, detti "link", sono collegati ad altri testi, cui si accede cliccando con il mouse sul link. Se il link collega file di diversa natura, cioè non solo testi ma anche file audio, video e immagini, allora l' ipertesto è in realtà un "ipermedia". Un link può collegare una parola a un' immagine, un' immagine a un brano musicale o a un video e così via. Il termine "ipertesto" è stato coniato dall' americano Ted Nelson all' incirca nel 1965. Il programma per consultare un ipertesto viene definito "browser" (dall' inglese "to browse", che significa curiosare).

⁹ HyperText Transfer Protocol è il protocollo per trasferire sulla rete i documenti ipertestuali.

linguaggio di markup ma nel quale i tag non sono predefiniti, cioè non hanno una semantica assoluta. Il primo scopo di XML è di permettere la strutturazione di documenti. La visione dell'informazione come da una parte contenuta in documenti Web, dall'altra in database, ha portato allo sviluppo di strumenti differenti. Questi due approcci hanno portato ad obiettivi differenti con la possibilità di convergenza:

Il Web ha portato con sé le seguenti caratteristiche, metodologie, linguaggi:

- 1) Una infrastruttura globale ed un insieme di standard per supportare lo scambio di documenti.
- 2) Un formato di presentazione per ipertesti (HTML).
- 3) Interfacce utente per il caricamento e la visualizzazione di questi documenti.
- 4) Un nuovo formato, XML, per lo scambio di dati strutturati.

La tecnologia database ha invece portato:

- 1) Tecniche per la memorizzazione e l'interrogazione per l'accesso a grandi quantità di dati altamente strutturati.
- 2) Modelli di dati e metodi per la strutturazione dei dati.
- 3) Meccanismi per il mantenimento dell'integrità e della consistenza dei dati.
- 4) Un nuovo modello di dati semistrutturati che riduce le restrizioni dei sistemi di basi di dati altamente strutturati.

E' attraverso la convergenza del punto 4 di entrambi gli elenchi, ovvero *XML* e *dati semistrutturati*, che sta emergendo una *tecnologia per i dati sul Web*.

Vi è una sostanziale differenza tra l'architettura Client/Server dei normali database riportata in *figura 1* e quella per lo scambio di dati sul Web di *figura 2*. Nella prima il Client, che può essere una persona o un programma, formula una query che viene processata, compilata in codice ottimizzato ed eseguita, in attesa di una risposta inviata dal Server. Nell'approccio Web il contesto è più articolato: il gradino più basso è costituito dai fornitori di dati, i Server che possono essere convenzionali Server per basi di dati oppure file system o ancora qualsiasi applicazione in

grado di produrre dei dati. La sorgente traduce i dati in un comune modello logico e formato, ad esempio XML. Dall'altra parte nell'architettura vi saranno i Client, che consistono in interfacce utente o applicazioni come package di analisi che utilizzano i dati.

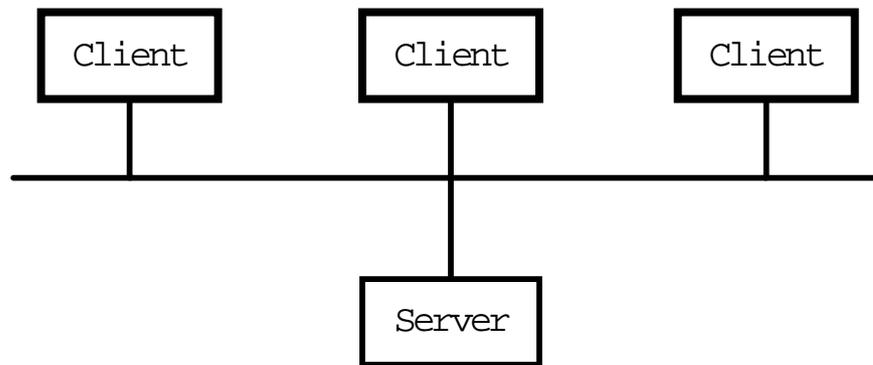


Figura 1 - Tradizionale architettura database Client/Server

Sempre in figura 2, tra i Client ed i Server vi può essere una intera collezione di intermediari denominati *middleware*, il software che trasforma, integra o aggiunge ulteriore valore ai dati.

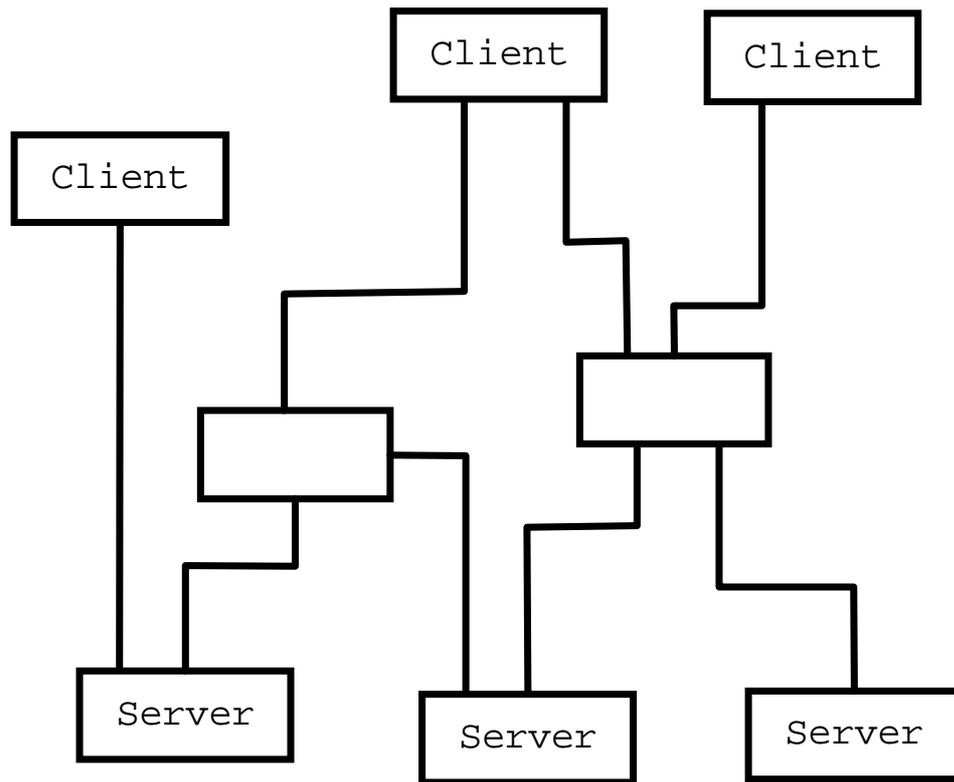


Figura 2 - Architettura basata sul Web

Nello scenario più semplice non vi è lo strato middleware e l'iterazione avviene direttamente tra i Client ed i Server. I dati scorrono dai Server ai Client mentre le interrogazioni (query) percorrono il cammino opposto. L'analisi della query da parte del Server consiste nel tradurre l'interrogazione nel modello di dati interno del Server, processarlo grazie al motore interno di analisi delle query e restituire il risultato.

Negli oltre 30 anni di sviluppo dei database, sono stati identificati principalmente tre livelli di astrazione per rappresentarne l'implementazione e la funzionalità. Il primo è il *livello fisico* dei dati che descrive come i dati sono memorizzati a livello hardware e quali indici sono disponibili. Sopra a questo vi è il *livello logico* che definisce ad esempio la validità delle interrogazioni. Infine vi è il *livello esterno* che fornisce un insieme di viste e di interfacce che gli utenti ottengono dai dati.

Allo stato attuale la distinzione tra gli strati logico e fisico non è ancora stata riconosciuta per i dati Web e non vi è ragione perché dovrebbe esserlo, in quanto la principale funzionalità del Web consiste nel trasmettere brevi file

di ipertesto. Ma questa visione potrebbe cambiare perché in sintesi possiamo pensare all'XML come ad una *rappresentazione fisica* (un formato dati) oppure come ad una *rappresentazione logica*. Il secondo modo di concepire l'XML è probabilmente più consono alle caratteristiche implicite del linguaggio che consentono la rappresentazione di dati semistrutturati.

XML si presta a diventare un formato di dati intermedio per la comunicazione tra applicazioni che non possono essere riprogettate per supportare lo stesso formato dei dati.

1.2.3 – Il formato dei documenti

E' possibile identificare due tipologie di formato nei documenti: quelli *orientati al contenuto* e quelli *orientati alla presentazione*. I primi sono solitamente proprietari e l'informazione viene memorizzata in maniera strutturata al fine di facilitarne l'elaborazione. Il documento è analizzabile, consultabile, modificabile tramite specifiche applicazioni in grado di interpretarne il formato. Viceversa i formati di documenti orientati alla presentazione, tra i quali HTML e PostScript, mantengono esclusivamente quella parte di informazione utile alla resa del documento. Entrambi queste tipologie hanno dei limiti: i formati orientati al contenuto portano alla proliferazione di software proprietario ed impediscono la creazione di strumenti, come ad esempio i motori di ricerca, in grado di lavorare in modo uniforme sui vari documenti. I formati orientati alla presentazione rendono difficoltosa o impossibile la successiva rielaborazione anche se presentano il vantaggio di essere standard aperti che ne facilitano la diffusione e portano alla creazione di un gran numero di programmi in grado di utilizzare questo formato. La necessità di dotare i formati orientati al contenuto delle stesse qualità di quelli orientati alla presentazione ha portato alla creazione dell'SGML (Standard Generalized Markup Language). Dal 1986, anno in cui venne proposto come standard internazionale, non riscosse un grande successo, anche a causa delle consistenti risorse fisiche di cui necessitava, che all'epoca non erano disponibili come oggi. Nonostante il suo scarso successo

e adozione, due suoi discendenti l'HTML e DocBook hanno riscontrato una diffusione planetaria.

I *linguaggi di markup* si dividono tipologicamente in base alle caratteristiche della loro semantica e possono essere divisi in due classi:

1. Linguaggi procedurali: in letteratura vengono indicati come “*Specific markup language*” e la semantica è per essi un processo computazionale di trattamento del testo. Esempi noti sono *TeX* sviluppato da Donald Knuth nel 1984 e particolarmente adatto alla preparazione di testi con formule matematiche; *RTF (Rich Text Format)* è un formato sviluppato da Microsoft per consentire lo scambio interpiattaforma di documenti formattati.
2. Linguaggi dichiarativi o descrittivi: definiti anche come “*Generic markup language*”, in cui la semantica indica l'appartenenza di una porzione di testo ad una certa classe di caratteristiche testuali. Il più potente di questi linguaggi è proprio l'SGML.

1.3 – Lo Standard Generalized Markup Language

Il linguaggio SGML è un sistema per la rappresentazione dell'informazione testuale su supporto digitale. Per le proprietà di cui gode è attualmente uno dei più potenti e versatili sistemi per la creazione, archiviazione, gestione e trasmissione di testi in formato digitale. Le sue caratteristiche fondamentali sono espresse nella sua stessa denominazione:

- Markup Language: SGML è un metalinguaggio o più precisamente un metalinguaggio per la codifica testuale basato su di un sistema di etichette che vengono associate al contenuto verbale di un testo per esplicitarne le caratteristiche.
- Generalized: perché si tratta di un sistema di codifica dichiarativo fortemente astratto e generalizzato e al tempo stesso altamente flessibile.
- Standard: perché costituisce uno standard formale sviluppato dalla *International Standardization Organization* e pubblicato ufficialmente nel 1986 con la sigla ISO 8879.

SGML è un linguaggio con alcune caratteristiche fondamentali di indipendenza:

1. Indipendenza dal sistema informatico: è completamente autonomo dal tipo di piattaforma hardware e software su cui viene utilizzato.
2. Indipendenza dai dispositivi: per cui può essere usato per archiviare testi in formato digitale su qualsiasi tipo di supporto attualmente esistente (dischi magnetici, dischi ottici, ambienti distribuiti, nastri, carta, ecc.).
3. Indipendenza dalle applicazioni: la sintassi SGML permette la rappresentazione di qualsiasi tipo di testo e di qualsiasi caratteristica testuale, indipendentemente dalle finalità e dai processi computazionali per i quali il testo è stato memorizzato e codificato. Esso non è pensato esplicitamente per la stampa, per l'archiviazione digitale, per l'analisi testuale, ma può servire ottimamente a tutti questi scopi di trattamento testuale.
4. Indipendenza dalle lingue e dai sistemi di scrittura: SGML è assolutamente indipendente dalle lingue nazionali e dai relativi sistemi di scrittura.

Queste caratteristiche fanno di SGML il sistema di codifica testuale più adeguato per tutte quelle applicazioni di gestione testuale in cui la salvaguardia e la riusabilità o portabilità delle informazioni sono condizioni indispensabili.

SGML introduce il concetto di *tipo di documento*, applicando ai dati testuali una nozione che caratterizza anche altre strutture di dati informatiche. Un tipo di documento identifica una classe di tutti i documenti che presentano caratteristiche analoghe strutturali: ad esempio il tipo di documento “lettera commerciale” oppure “testo narrativo” o ancora “antologia poetica”. Ogni tipo è caratterizzato da un insieme finito di elementi costituenti che sono soggetti a determinate relazioni posizionali: ad esempio una antologia è costituita da una sequenza di poesie, ogni poesia è a sua volta costituita da una sequenza di versi e così via. La definizione formale di tipo di documento viene effettuata in una particolare struttura dati denominata *Document Type Definition* (DTD). Una DTD, analizzata più esplicitamente nelle pagine

seguenti, è costituita da un elenco di dichiarazioni che seguono la sintassi SGML e che identificano gli elementi di un testo (element) attraverso un identificatore generico (generic identifier costituito da una stringa di caratteri) ed il loro modello di contenuto (content model), ovvero l'insieme di sottoelementi e/o caratteri che ciascun documento può contenere, con i relativi rapporti di ordine e ricorrenza. Ad ogni elemento possono inoltre essere associati uno o più attributi che ne specificano ulteriori caratteristiche o funzioni non strutturali: ad esempio l'aspetto fisico di un titolo o di un segmento testuale enfatizzato, il tipo di una divisione testuale, l'attribuzione di un identificatore unico. Una volta definita una DTD è possibile codificare un documento come istanza (document instance) di quel tipo di documento. L'associazione di un documento viene effettuata mediante una esplicita dichiarazione che va collocata all'inizio di ogni documento SGML valido.

Comunque, come accennato, l'SGML non ha mai suscitato particolare interesse tra gli sviluppatori Web, probabilmente a causa della sua complessità e delle difficoltà che l'adozione e l'utilizzo di questo linguaggio comportano.

1.3.1 – La necessità di un nuovo linguaggio per il Web

Non avrebbe oggi senso parlare di internet se non in stretta associazione con il linguaggio HTML nato da una DTD SGML negli anni ottanta elaborato da Tim Berners-Lee durante la realizzazione di un'applicazione ipertestuale. HTML è propriamente un linguaggio laddove invece SGML è un metalinguaggio: questo significa che mentre nel primo caso lo sviluppatore è vincolato all'uso di marcatori definiti nelle specifiche, nel caso di qualsiasi metalinguaggio, per mezzo della sintassi definita nelle specifiche, è possibile definire nuovi tag (marcatori). Nel primo caso ne derivano vantaggi di semplicità perché l'attività dello sviluppatore si limita alla stesura del testo del documento, spesso attraverso l'uso di tool

WYSIWYG¹⁰, mentre nel secondo caso derivano vantaggi enormi di possibilità nel trattamento di dati, attività che risulta estremamente vantaggiosa in caso di applicazioni Web complesse che comunicano con altre o con basi di dati, permettendo la definizione di tag in cui è possibile controllare e vincolare anche il contenuto.

Nuovi servizi però necessitavano di un'ulteriore passo avanti che con il solo HTML non erano direttamente realizzabili. La semplice staticità degli standard ha permesso la diffusione da parte degli utenti di quantità enormi di dati grazie alla creazione di pagine html: non più un unico Server centrale, ma tanti Server connessi tra di loro costituiscono la rete che contiene questo patrimonio di informazioni in via di continua espansione. Con il progressivo aumento della mole di documenti, la complessità è diventata tale che, quella stessa semplicità che permette a chiunque di diventare "editore", da fattore vincente rischia di diventare una caratteristica che rallenta un ulteriore sviluppo per il quale è richiesta maggiore affidabilità del mezzo e duttilità degli standard. I problemi più impellenti caratterizzano la crisi dei motori di ricerca con il conseguente aumento dei "dead link" o di risultati non eccellenti alle nostre interrogazioni. Questi problemi sono da imputarsi ad alcuni limiti principalmente insiti nel linguaggio che ha fatto dilagare il Web: l'HTML.

Il successo di questo linguaggio si deve principalmente alla sua semplicità e all'assenza nei browser di un meccanismo di convalida sintattica. Ma questa caratteristica manifesta nel tempo i suoi aspetti negativi in virtù del fatto che molti sviluppatori, concentrati essenzialmente sulle esigenze estetiche, hanno usato i marcatori senza tenere troppo in considerazione il loro significato gerarchico. Informazioni di rappresentazione¹¹ e di struttura¹² si sono alternate con troppa disinvoltura, senza una precisa delimitazione tra le

¹⁰ Applicazioni visuali che utilizzano il paradigma WYSIWIG = What you see is what you get. Rendono immediata la stesura dei documenti attraverso un'approccio visuale di realizzazione pratica del file.

¹¹ Un *marcatore di rappresentazione* consiste in un identificatore per la formattazione del documento, ad esempio per il grassetto.

¹² Un *marcatore di struttura* identifica ad esempio il paragrafo <P>.

due. Solo da successive specifiche del consorzio W3C¹³ è stata inserita la possibilità di usare i fogli di stile (CSS - Cascading Style Sheet) ovvero file separati nei quali definire la rappresentazione, in modo da lasciare il documento HTML vero e proprio libero di recuperare il suo valore di struttura. Tuttora la rappresentazione può essere definita all'interno del file HTML e persistono i problemi legati all'assenza di vincoli del controllo formale da parte del browser. Nonostante tutto ciò appare evidente la chiarissima direzione del Consorzio verso una separazione più puntuale.

In questo modo si perdono molte delle potenzialità dei motori di ricerca, costretti a trovare informazioni su tutto il testo a parità di importanza, proprio nel momento in cui la mole dei documenti in rete diventa tale da richiedere un meccanismo più efficiente. Una divisione corretta fra rappresentazione e struttura associate ad un controllo formale, garantiscono invece a qualsiasi software che analizzi il documento non solo di individuare nel testo eventuali ricorrenze, ma anche di stabilirne la rilevanza in base alla posizione nella gerarchia testuale. Purtroppo però la confusione tra tag di struttura e tag di rappresentazione non permette questa possibilità ed obbliga a ricerche che spesso restituiscono migliaia di documenti senza indicare nulla della rilevanza del termine da individuare.

HTML, linguaggio nato per un publishing elementare, non rispondeva più alle esigenze degli sviluppatori di applicazioni Web e si trovava a dover assicurare potenzialità impensabili al momento della sua nascita: lasciava spazio allo sviluppo di tecnologie parallele che potessero assicurarne la sopravvivenza. Tra nuovi linguaggi come Javascript e plugin come Shockwave o Adobe Acrobat Reader, HTML è diventato piano piano un assemblatore di tecnologie piuttosto che un linguaggio vero e proprio e ciò ha comportato anche problemi di portabilità delle applicazioni Web perché tutte queste tecnologie non sono standard, bensì soluzioni proprietarie più o meno

¹³ Il termine significa "World Wide Web Consortium". Il W3C è un consorzio di aziende del settore informatico che si occupa di stabilire standard di riferimento per il Web. Questo consorzio, in altri termini, studia i sistemi ed i linguaggi per la trasmissione di dati attraverso il Web e ne ufficializza l' utilizzo attraverso raccomandazioni definitive. Al W3C si devono gli standar di HTML, XML, CSS e altri ancora. Il sito ufficiale del consorzio è raggiungibile da: <http://www.w3c.org>.

diffuse che necessitano di software specifici e di particolari versioni. Tutto ciò è ben lontano dall'iniziale filosofia ed anche alla tendenza che si sta affermando in questo periodo di costruire ambienti standard in grado di permettere la costruzione di applicazioni portabili a prescindere dal sistema operativo o dall'ambiente di sviluppo.

Le pagine HTML hanno inoltre come scopo principale quello di essere visualizzate da un browser, infatti si dice che i dati nell'HTML sono orientati al video. Risulta molto difficoltosa un'eventuale elaborazione successiva delle informazioni contenute nelle pagine HTML.

Se internet è ormai un mezzo di comunicazione di massa, nasce l'esigenza dell'introduzione di correzioni che consentano di evitare questa frammentazione che è controproducente per un uso semplice e capillare di internet stessa.

1.4 - XML: il nuovo linguaggio per il Web

Un limite del Web prima di XML era rappresentato dalla dipendenza ad un tipo fisso di documento, HTML. I metalinguaggi di markup offrono rispetto ad HTML essenzialmente tre vantaggi:

- 1) *L'estensibilità* che permette la definizione di set personalizzati di marcatori consentendo a gruppi di persone o ad organizzazioni di creare il proprio linguaggio di markup, specifico per il tipo di informazione che trattano.
- 2) La salvaguardia degli elementi strutturali definiti in un file esterno chiamato *Document Type Definition (DTD)*.
- 3) La *validazione* per la quale ogni documento passa attraverso un validatore che ne attesta la conformità alle regole definite nella DTD.

Già SGML include queste caratteristiche ma, come accennato, è stato considerato un linguaggio inadeguato a causa della sua eccessiva complessità che ne ha frenato la penetrazione. Per questo il W3C ha optato per la definizione di un linguaggio che, pur mantenendo tutti i vantaggi sopra elencati, fosse più semplice. Tre membri del W3C produssero nell'agosto del 1997 un primo draft di XML: Tim Bray, James Clark e Michael Sperlberg-

McQueen. Dopo diversi mesi di lavoro, nel febbraio del 1998 è stata rilasciata la versione 1.0 delle specifiche di XML. Il progetto prevede la definizione di un gruppo di specifiche anche per la gestione dei link (*XLL*) e per la rappresentazione (*XSL*).

1.4.1 – Applicazioni di XML

Lo stesso documento XML può essere mostrato in vari modi diversi: su di un monitor, attraverso audio da un cellulare, ecc. Questo vuol dire che un documento scritto secondo queste specifiche può essere veicolato attraverso device diversi, non necessariamente presi in considerazione all'atto della sua stesura. XML quindi pur essendo nato per il mondo Web, ha senso anche al di fuori da esso, comunque e dovunque qualcuno desideri produrre un documento, a prescindere dal mezzo trasmissivo.

Questo linguaggio è importante in due classi di applicazioni Web: la creazione di documenti e lo scambio di dati; i Server Web attualmente utilizzati richiedono, per essere in grado di servire documenti XML, minime modifiche di configurazione; inoltre il metodo standard di collegamento e la connessione dei documenti XML utilizza gli URL, che vengono interpretati correttamente dalla maggior parte dei software per internet.

Come accennato, può essere utilizzato come piattaforma per lo scambio di dati tra le applicazioni, come illustrato nella figura 3. Questo è reso possibile dal fatto che è orientato alla descrizione dei dati. Era sempre stato difficile trovare un formato di interscambio che potesse essere utilizzato per il trasferimento di dati tra database di fornitori differenti e sistemi operativi diversi. Quel tipo di interscambio è ora diventato una delle principali applicazioni di questo linguaggio.

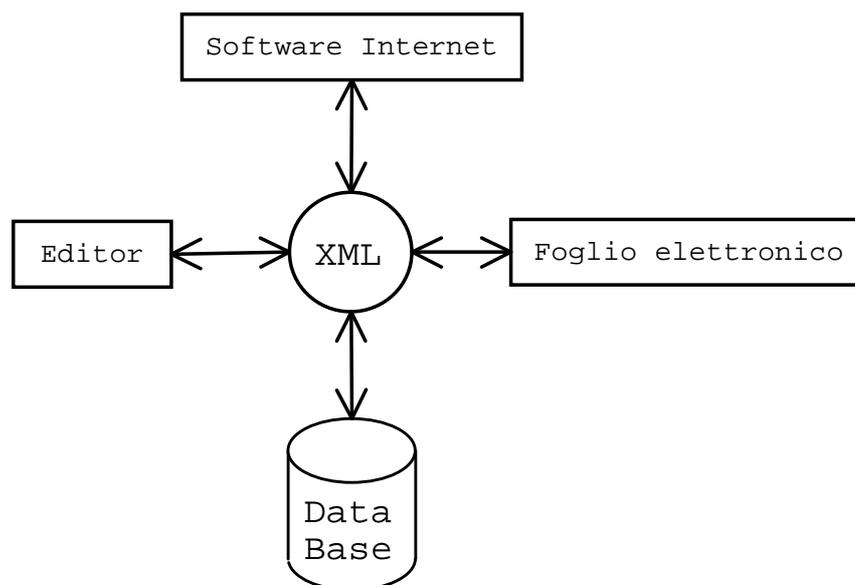


Figura 3 - XML può essere utilizzato come piattaforma per lo scambio di dati

1.4.2 – Il Parser ed il Processore

Un tool per la lettura di un documento XML consta di due parti:

- 1) Il *Parser* che esegue il controllo semantico e gestisce gli errori.
- 2) Il *Processor* che, utilizzando un altro file in cui è definita la formattazione dei vari tag, visualizza il documento.

Già da questa dinamica si comprende come sia evidente la separazione tra struttura e rappresentazione che, come si è visto, è uno degli aspetti chiave per la buona realizzazione di un ipertesto: XML infatti non dice nulla in riferimento alla rappresentazione (formattazione) che può essere gestita attraverso un altro linguaggio XSL, cui si farà cenno in seguito.

Il controllo da parte del Parser avviene su due livelli: valutando la conformità del documento alla DTD (illustrata in seguito) di riferimento prima e in caso di non conformità, eseguendo un altro controllo relativo alle regole generali della sintassi XML. Proprio per via di questo doppio controllo i documenti XML possono essere di due tipi: *ben formati* e *validi*. Si ha validità quando un documento XML è conforme ad una DTD specifica e ben formato nel caso sia conforme alle specifiche XML.

1.4.3 – *La Document Type Declaration*

Sia i documenti DTD che XML iniziano nella prima riga con la *Document Type Declaration* o *dichiarazione XML*, che non ha tag di chiusura, per cui potrebbe essere del tipo:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
```

dove:

- `<?xml ... ?>` è l'elemento che introduce il documento e lo dichiara come aderente alle specifiche XML.
- “version” definisce la versione delle specifiche di riferimento.
- “standalone” è l'attributo che indica se il documento è ben formato (“yes”) o valido (“no”).
- “encoding” indica il tipo di codifica. Al momento vi sono iso-8859-1, UTF-8, ecc.

Quindi, come si è visto, con la “Document Type Declaration” si dichiara l'adesione del documento alle specifiche XML e se questo fa riferimento o meno ad una precisa DTD ovvero se è *valido* o semplicemente *ben formato*.

1.4.4 – *La Document Type Definition in XML (DTD¹⁴)*

Come accennato questo linguaggio non è limitato ad un insieme fisso di tipi di elementi, ma permette di definire e utilizzare elementi e attributi personalizzati; per far questo viene fornita una sintassi con cui è possibile specificare gli elementi e gli attributi che possono essere utilizzati all'interno dei documenti. In altre parole è possibile creare un modello chiamato *Document Type Definition* che descrive la struttura ed il contenuto di una classe di documenti. XML, consentendo quindi di formalizzare il concetto intuitivo di *tipo di documento*¹⁵ attraverso le DTD. Lo stesso XML ha una

¹⁴ In generale con l'abbreviazione DTD nella tesi, ci si riferisce alla *Document Type Definition* e non alla *Document Type Declaration*.

¹⁵ Cos'è un “Tipo di documento”? Intuitivamente chiunque è in grado di distinguere senza difficoltà una lettera da un opuscolo pubblicitario o dire se un libro sia un romanzo, un elenco telefonico, un volume enciclopedico, ecc. indipendentemente dal titolo, dalla rilegatura, dai caratteri, dai colori, ecc.

propria DTD¹⁶ in cui vengono elencate le regole della specifica stessa del linguaggio. Con l'XML è anche introdotta una classe di documenti che fa riferimento alla sola DTD dell'XML. La creazione di una DTD personale non è quindi indispensabile, come mostra la figura 4.

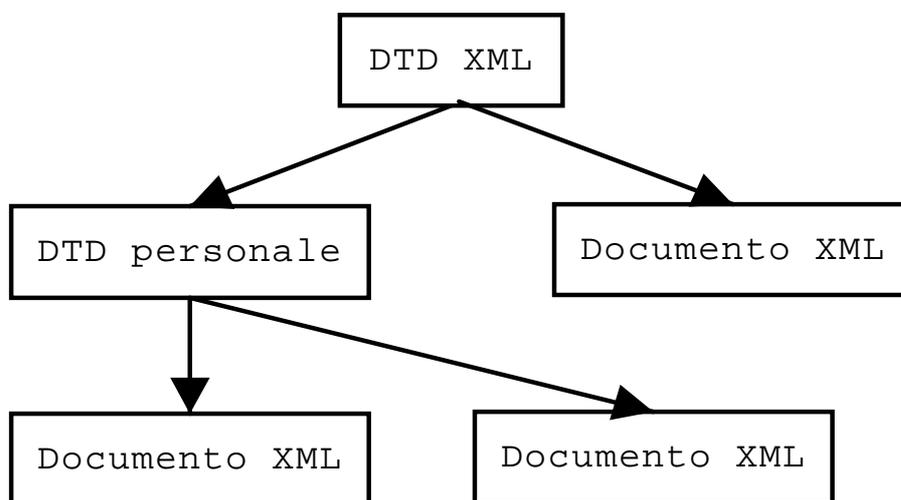


Figura 4 - Un documento XML può fare riferimento al DTD dell'XML o ad una DTD personale

Questa possibilità semplifica notevolmente l'utilizzo dell'XML rispetto all'SGML: nel linguaggio SGML infatti le DTD sono indispensabili, spesso complicate da imparare e creare e un documento deve fare obbligatoriamente riferimento ad una di esse.

La DTD contiene le regole che definiscono i tag usati nel documento e ne definisce in pratica la struttura. Sebbene non sia obbligatoria, per chiarezza può essere conveniente utilizzarla. XML prevede la possibilità di definire la struttura del documento non solo in un file esterno, bensì anche al suo interno, pertanto di fatto i due esempi di file XML che seguono, danno lo stesso risultato:

¹⁶ Descritto nella specifica REC-xml-19980210.

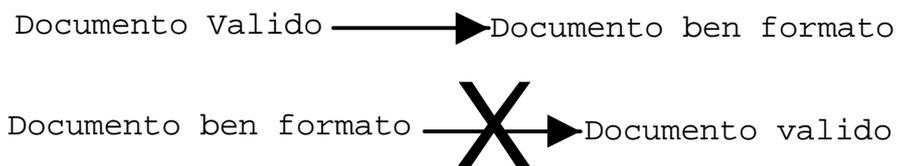
```
<?xml version="1.0"?>
<!DOCTYPE saluto SYSTEM "hello.dtd">

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE saluto [
    <!ELEMENT saluto (#PCDATA)>
]>
```

Nel primo caso per quanto riguarda la struttura, il documento fa riferimento ad un file esterno (`hello.dtd`) mentre nel secondo caso il contenuto di questo è inglobato all'interno del documento stesso. La *Document Type Definition* è sempre contenuta all'interno della *Document Type Declaration* e impostata dal marcatore `<DOCTYPE>` e quando ne è inglobata si apre con il carattere “[“ e si chiude con “]”.

La separazione della DTD dal documento riduce notevolmente la dimensione del file del documento XML e fornisce altri vantaggi: una DTD separata può essere utilizzata in altri documenti da chiunque vi abbia accesso. Un altro autore può creare un documento XML utilizzando la stessa struttura e contenuto completamente differente. Poiché il nuovo documento creato segue la DTD, potrebbe essere letto da qualsiasi applicazione in grado di elaborare la DTD.

Un documento XML è *ben formato* quando non viola nessuna regola di produzione della specifica XML mentre è *valido* se dichiara in una dichiarazione di tipo di documento di essere conforme ad una DTD, secondo le seguenti relazioni:



I documenti che non hanno una dichiarazione di tipo di documento non sono propriamente documenti non validi, poiché non violano nessuna DTD, ma non sono altresì documenti validi, perché non sono conformi a nessuna DTD.

1.4.5 – *Gli elementi*

Nel precedente esempio `ELEMENT` definisce “saluto” come un elemento che potrà essere usato come tag nel documento e che contiene del testo (PCDATA = Parsed Character Data). L’elemento è il blocco di base del documento XML. Si è visto che uno dei vantaggi di questo linguaggio sia la possibilità di definire tag propri. Questo è possibile per mezzo della dichiarazione di elemento che avviene nella DTD. Nel documento XML l’elemento è delimitato da un marcatore di apertura e da uno di chiusura, ad esempio:

```
<saluto> Ciao, mondo! </saluto>
```

In questo caso si ha l’elemento “saluto” che contiene la porzione di testo “Ciao, mondo!”. A differenza di HTML qui gli elementi sono case sensitive per cui `<saluto>` non ha lo stesso valore di `<Saluto>`.

La caratteristica di poter aggiungere informazioni semantiche al testo, consente di aggiungere conoscenza specifica al documento. Ad esempio, considerando il seguente marcatore `<Autore>Giovanni Rossi</Autore>` è immediato comprendere che il suo uso permette di semplificare la creazione di applicazioni che svolgono operazioni “intelligenti” con i documenti elettronici: un motore di ricerca sarebbe in grado di eseguire ricerche specifiche per trovare tutti i documenti di cui Giovanni Rossi è l’autore. In questo modo si può superare il limite di HTML per il quale i dati sono orientati alla rappresentazione e difficili da utilizzare per un’elaborazione successiva.

Il commercio on-line è in pieno sviluppo ma statistiche evidenziano che vi sia una certa frustrazione da parte degli acquirenti abituali via internet per la difficoltà di trovare i prodotti di cui hanno bisogno. Il problema risiede nel sistema di indicizzazione delle merci. Se diversi negozi on-line definissero uno standard comune XML dei propri cataloghi¹⁷ che elencano i prodotti disponibili, sarebbe possibile a siti di terze parti, indicizzare questi cataloghi e restituire all’utente il prodotto con le caratteristiche più vantaggiose attraverso

¹⁷ Vedere l’Appendice A.

un'analisi e un confronto dei cataloghi. Un approccio di questo tipo è già esistente in qualche negozio on-line anche se non ha ancora la penetrazione globale che dovrebbe avere.

Si potrebbe quindi definire una DTD relativa a delle aziende agricole che hanno la necessità di catalogare fiori, piante, ecc. Un esempio di documento XML in questo caso potrebbe essere il seguente dove, per ogni pianta, vengono riportati il nome italiano e il corrispettivo latino:

```
<?xml version="1.0"?>
<!DOCTYPE piante SYSTEM "piante.dtd">
<catalogo>
  <pianta>
    <comune>Adonide Primaveraile</comune>
    <botanico>Adonis Vernalis</botanico>
  </pianta>
  <pianta>
    <comune>Pino</comune>
    <botanico>Pinus Pinaster</botanico>
  </pianta>
  ...
</catalogo>
```

L'annidamento è il processo che consente di incorporare un oggetto o un costrutto all'interno di un altro. Un documento XML può ad esempio contenere elementi annidati e anche altri documenti. Mentre HTML ed il linguaggio di origine SGML consentono di omettere i tag di chiusura senza invalidare il codice, i listati XML richiedono esplicitamente che un tag di chiusura venga utilizzato per ogni elemento.

`<linea/>` è invece un esempio di elemento vuoto, che va usato nel caso di elementi che non debbano avere un contenuto ed in cui la barra "/" è posta dopo il nome dell'elemento a differenza del tag di chiusura.

Come si è visto gli elementi vanno definiti nella DTD con una sintassi del tipo: `<!Element [nome_elemento] ([elenco_sottoelementi])>`.

1.4.6 – Gli attributi

Gli attributi permettono l'aggiunta all'elemento di informazioni addizionali. Già HTML prevedeva l'uso di attributi degli elementi, per esempio nel caso di: `` dove "align" altro non è che un attributo dell'elemento IMG. Però mentre in HTML l'uso degli attributi descrive la rappresentazione dell'elemento, in XML essi costituiscono vere e proprie informazioni addizionali. Supponiamo di voler definire una DTD per documenti relativi alla catalogazione di acqua in bottiglia. Sicuramente l'elemento base è *acqua* ma la bottiglia potrebbe essere di vetro oppure di plastica o cartone. La definizione sarebbe dell'unico elemento `<acqua>` ma con attributo *bottiglia* che potrà assumere una delle tre possibilità elencate. Esempi potrebbero essere:

```
<acqua bottiglia="plastica">San Bernardo</acqua>
<acqua bottiglia="vetro">Fiuggi</acqua>
ecc.
```

Gli attributi vengono dichiarati nella sezione *attribute list* della DTD che contiene il nome dell'elemento cui gli attributi si riferiscono, il tipo di dati, la lista dei valori degli attributi stessi e il valore di default. In questo caso nella DTD si avrà un'attribute list di questo tipo:

```
<!ATTLIST acqua bottiglia (plastica|vetro|cartone) "plastica">
```

Nel linguaggio XML gli attributi vengono dichiarati nella DTD utilizzando la seguente sintassi:

```
<!ATTLIST NomeElemento NomeAttributo Tipo Default>
```

dove `<!ATTLIST>` rappresenta il tag che identifica una dichiarazione di attributo. La voce `NomeElemento` rappresenta il nome dell'elemento a cui vengono applicati gli attributi. La voce `NomeAttributo` identifica il nome dell'attributo e `Default` specifica le impostazioni predefinite relative all'attributo. Infine `Type` indica il tipo di attributo dichiarato secondo la seguente tabella:

Tipo di Attributo	Utilizzo
CDATA	Denota dei dati carattere "grezzi" che il parser non deve considerare.
ENTITY	Il valore dell'attributo deve fare riferimento a un'entità binaria esterna dichiarata nella DTD.
ENTITIES	E' equivalente all'attributo ENTITY, ma consente l'utilizzo di più valori separati da spazi.
ID	Il valore dell'attributo deve essere un identificatore univoco. Se un documento contiene attributi ID con lo stesso valore, l'elaboratore produrrà un errore.
IDREF	Il valore deve essere un riferimento a un ID dichiarato in un altro punto del documento. Se l'attributo non corrisponde al valore dell'ID specificato, l'elaboratore produrrà un errore.
IDREFS	E' equivalente all'attributo IDREF, ma consente l'utilizzo di più valori separati da spazi.
NMTOKEN	Il valore dell'attributo consiste in una qualsiasi combinazione di caratteri del token del nome, rappresentati da lettere, numeri, punti trattini, due punti o caratteri di sottolineatura.
NMTOKENS	E' equivalente all'attributo NMTOKEN, ma consente l'utilizzo di più valori separati da spazi.
NOTATION	Il valore dell'attributo deve fare un riferimento a un'annotazione dichiarata in un altro punto della DTD. La dichiarazione può anche essere costituita da un elenco di annotazioni. Il valore deve corrispondere a una delle annotazioni dell'elenco. Ogni annotazione deve avere la relativa dichiarazione nella DTD.
Enumerated	Il valore dell'attributo deve corrispondere a uno dei valori inclusi. Ad esempio: <!ATTLIST MyAttribute (content1 content2)>.

Tabella 1 - Tipi e utilizzo degli attributi

L'attributo deve avere poi dei valori che indicano al parser come comportarsi durante il controllo di conformità: #REQUIRED indica che l'attributo deve avere necessariamente un valore ogni volta che è usato l'elemento, #IMPLIED specifica che l'attributo può non avere valore, #FIXED vuol dire che l'attributo può non avere valore ma se ce l'ha deve necessariamente essere quello di default.

Nel seguente esempio viene mostrato il possibile uso degli attributi in una DTD; viene definito un libro come tag che deve contenere i marcatori figli autore ed editore:

```
<!ELEMENT libro (autore, editore)>
<!ATTLIST autore
    sesso CDATA #REQUIRED
    nascita CDATA #IMPLIED>
<!ATTLIST editore
    città CDATA #FIXED "Alessandria">
```

1.4.7 – Le entità

Un documento XML non deve necessariamente essere composto da un solo file, ma può assemblare al suo interno parti diverse che prendono il nome di *entities* o *entità*. Queste permettono di creare dei riferimenti a dati esterni, ad altri documenti o anche a porzioni di testo, a patto che ci sia una specifica dichiarazione DTD. Le entità possono essere quindi interne o esterne, a seconda di dove fisicamente si trovano i dati rispetto al documento. Pertanto, nel caso di una dichiarazione del tipo:

```
<!ENTITY commedia "Dante Alighieri, La Divina Commedia">
```

all'interno del documento avrà un'entità interna, che quindi può essere paragonata ad una vera e propria variabile, mentre in una dichiarazione come `<!ENTITY commedia "./commedia.txt">` l'entità sarà esterna perché vi è un riferimento ad un file diverso, che può essere posto anche su di una macchina remota. Il concetto chiave è che dopo la dichiarazione nella DTD, l'entità è utilizzabile in tutti i documenti che a quella DTD fanno riferimento, semplicemente per mezzo del suo nome. E' quindi evidente il vantaggio della costruzione di un documento per mezzo di entità: è possibile ad esempio definire la struttura nella DTD, lo scheletro nel documento con la possibilità di riempirlo con contenuti presi dall'esterno. Oltre alla rappresentazione (formattazione), anche il contenuto diventa in questo modo un modulo a parte e risulta maggiormente mantenibile.

1.4.7.1 – Le entità predefinite

Si è visto che nel linguaggio XML alcuni caratteri sono utilizzati per contrassegnare il documento in modo specifico. Le parentesi angolari (<, >) e la barra (/) sono interpretate come markup e non come dati di un carattere

effettivo, ad esempio: `<pianta>Rosmarino</pianta>`. Questi e altri caratteri sono riservati per il markup, non possono essere utilizzati come contenuto e prendono il nome di *entità predefinite*. Se si desidera che questi caratteri siano visualizzati come dati, è necessario utilizzare determinati codici:

Simbolo	Sequenza di Escape	Significato
&	&	e commerciale
<	<	parentesi angolare di apertura
>	>	parentesi angolare di chiusura
'	'	apostrofo
"	"	virgolette doppie

Tabella 2 - Caratteri di Escape per documenti XML

1.4.8 – Un esempio completo di documento XML e DTD

Nel seguente esempio viene mostrato un documento XML valido relativo ad un formato di messaggio di posta elettronica con una DTD interna:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!DOCTYPE EMAIL [
  <!ELEMENT EMAIL (TO, FROM, CC, SUBJECT, BODY)>
  <!ELEMENT TO (#PCDATA)>
  <!ELEMENT FROM (#PCDATA)>
  <!ELEMENT CC (#PCDATA)>
  <!ELEMENT SUBJECT (#PCDATA)>
  <!ELEMENT BODY (#PCDATA)>
]>
<EMAIL>
  <TO>giannini@di.unito.it</TO>
  <FROM>paolo@edu-al.unipmn.it</FROM>
  <CC>bizio@unipmn.it</CC>
  <SUBJECT>Documento Xml</SUBJECT>
  <BODY>Questo e' un esempio di documento Xml</BODY>
</EMAIL>
```

La DTD del documento identifica gli elementi che possono essere presenti ed il tipo di dati che deve contenere ognuno di essi e dunque svolge un ruolo di rigido regolamento per i documenti XML. La prima dichiarazione è relativa all'elemento EMAIL:

```
<!ELEMENT EMAIL (TO, FROM, CC, SUBJECT, BODY)>
```

All'interno di questa riga di codice, tra le parentesi, è presente un elenco degli altri elementi che possono essere contenuti nel documento. Questo elenco viene definito *modello di contenuto* e identifica gli elementi secondari che l'elemento EMAIL contiene e l'ordine in cui sono elencati. Se si togliesse dalla definizione del modello di contenuto un elemento, ad esempio FROM, vi sarebbe un errore nel documento XML in quanto non seguirebbe il modello di documento specificato.

1.4.9 – Simboli relativi alla struttura di una dichiarazione

Il linguaggio XML utilizza una serie di simboli per specificare la struttura di una dichiarazione di elementi. La tabella seguente identifica i simboli disponibili, lo scopo di ogni simbolo, un esempio di come vengono utilizzati ed il loro significato.

Simbolo	Scopo	Esempio	Significato
(..)	Racchiudono una sequenza, un gruppo di elementi o una serie di alternative	(content1, content2)	L'elemento deve contenere la sequenza content1 e content2
,	Separa gli elementi di una sequenza e identifica l'ordine in cui devono essere visualizzati	(content1, content2, content3)	L'elemento deve contenere content1, content2, content3 nell'ordine specificato
	Separa gli elementi in un gruppo di alternative	(content1 content2 content3)	L'elemento deve contenere content1 o content2 o content3
?	Indica che un elemento deve essere visualizzato una sola volta o non apparire mai	content1?	L'elemento può contenere content1. Se content1 viene visualizzato, deve apparire una sola volta
*	Indica che l'elemento può essere visualizzato ogni volta che l'autore desidera	content1*	L'elemento può contenere content1. Se viene visualizzato, può apparire una o più volte
+	Indica che un elemento deve essere visualizzato una o più volte	content1+	L'elemento deve contenere content1 una volta, ma può essere visualizzato anche più di una volta
Nessun simbolo	Indica che deve essere visualizzato un elemento	content1	L'elemento deve contenere content1

Tabella 3 – Simboli relativi alla struttura di una dichiarazione

1.4.10 – I Namespace

XML è un linguaggio che si candida a migliorare lo stato della riusabilità delle applicazioni Web attraverso l'uso di un approccio modulare. Il parser ed il processore devono essere in grado di gestire eventuali ambiguità, il cui rischio è inevitabilmente insito nell'approccio a moduli. Per questo sono nati i *Namespace*, che permettono l'uso di *tag ambigui* ovvero con lo stesso nome ma in riferimento a significati e ad ambienti diversi, utilizzando costrutti con nomi non equivoci. Un esempio può essere rappresentato considerando il marcatore "Titolo" e i vari significati che può assumere: titolo di un libro o di un articolo, titolo di studio, titolo inteso come posizione all'interno di un'azienda. Grazie ai Namespace è possibile creare diversi tag `<titolo>` ed utilizzarli nel documento senza correre il rischio che il parser ed il processor li confondano. I Namespace consentono l'uso di più DTD in un singolo documento XML. Un esempio di ambiguità risolta può essere il tag `name` definito in `<person:name>` ed in `<product:name>`: i namespaces permettono quindi l'uso di associazione diretta di tag di nomi con un preciso DTD e perciò con la combinazione di multipli DTD. Gli "spazi dei nomi" costituiscono quindi una metodologia per la creazione di nomi universalmente univoci in un documento XML, identificando i nomi degli elementi con una risorsa esterna univoca.

1.4.11 – XSL: la rappresentazione

Come accennato, l'*XSL (Extensible Stylesheet Language)* definisce la specifica per la presentazione e l'aspetto di un documento XML. E' stato presentato nel 1997 al W3C da un consorzio di industrie software, tra cui anche Microsoft, affinché venisse approvato come linguaggio di stile standard per i documenti XML. XSL è un sottoinsieme del *Document Style Semantics and Specification Language (DSSSL)*, il linguaggio di stile utilizzato in ambiente SGML. Gode delle proprietà di essere estensibile, potente ma nello stesso tempo di facile utilizzo.

Con l'*XSL* è possibile creare *fogli di stile* che permettono la visualizzazione di un documento XML in qualsiasi formato (audio, video,

braille, etc.), come mostrato nella figura 5. Le regole relative alla rappresentazione non devono essere necessariamente univoche, ma possono variare in base al diverso dispositivo di output o anche in seguito all'interazione con l'utente.

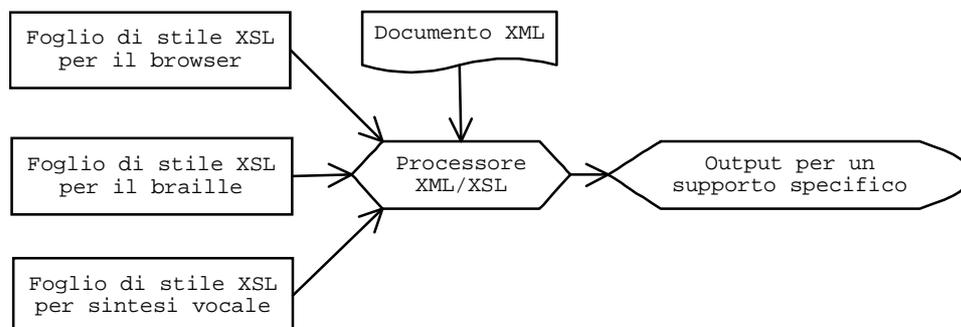


Figura 5 - L'XSL permette di creare fogli di stile per un qualsiasi formato

L'associazione tra un documento XML ed un foglio di stile avviene nel prologo del documento, per mezzo dell'istruzione “xml:stylesheet”, che ha come pseudo-attributi: href (necessario), type (necessario), title (opzionale), media (opzionale), charset (opzionale):

```
<?xml:stylesheet href="stile.xml" title="Compact" type="text/xml"?>
```

I fogli di stile sono quindi usati per applicare in modo coerente stili o formattazione ai documenti XML. Il tipo di foglio di stile più utilizzato sul Web è basato sulla specifica dei fogli di stile CSS (Cascading Style Sheets). XSL è stato inizialmente progettato con l'obiettivo di raccogliere i pregi di CSS e quelli di DSSSL, ovvero unire la semplicità alla potenza.

CAPITOLO 2

Analisi di Java & XML

2.1 – Java e XML: i linguaggi del momento

Java ed XML possono essere considerate le tecnologie del momento nell'ambito Web. Java è un linguaggio orientato agli oggetti che ha nella portabilità una delle sue caratteristiche principali. Sviluppare un'applicazione in linguaggio Java significa poterla eseguire indifferentemente su ogni sistema operativo per il quale esista una implementazione della Java Virtual Machine (JVM).

Inizialmente i programmi Java erano *applet*¹⁸ ovvero programmi di dimensione generalmente abbastanza ridotta eseguiti dalla JVM contenuta in un programma host, in genere il browser. Con il passare del tempo Java si è sempre più affermato come linguaggio per quel tipo di programmazione chiamata *server side*: sono stati sviluppati diversi *Application Server* che sfruttando la tecnologia dei *Servlet*¹⁹, hanno permesso la pubblicazione di contenuti attraverso pagine HTML create dinamicamente. Un sito Web costruito con questi applicativi non è più un insieme di pagine statiche collegate semplicemente tra loro attraverso dei link statici, ma diventa un'applicazione vera e propria, che permette di generare informazioni,

¹⁸ Il termine *applet* deriva dalla parola "applicazione" ovvero programma per computer.

¹⁹ Programma costruito in Java che viene eseguito su di un Server Web.

occupandosi al tempo stesso di come le stesse debbano essere visualizzate. Un approccio di questo tipo ha portato ad avere una prima naturale scomposizione tra quella che è la *gestione delle informazioni* e il *controllo della loro rappresentazione* ovvero visualizzazione all'utente.

La *portabilità del codice* Java non è più sufficiente per la creazione di applicazioni enterprise; è anche necessaria la *portabilità dei dati* ovvero un modo standard con cui sistemi di natura anche molto diversa possano comunicare tra loro, scambiandosi un insieme di informazioni di diversa provenienza. Questo ambizioso obiettivo è raggiunto dal nuovo linguaggio XML che si integra e si completa con il linguaggio Java: da una parte un meta-linguaggio²⁰ come XML che permette di rappresentare delle informazioni e delle relazioni che intercorrono tra di esse; dall'altro Java ovvero un linguaggio di programmazione orientato agli oggetti per la realizzazione di applicazioni portabili.

2.2 – L'utilità di Java per l'uso di XML

Si è visto che XML è un sistema per la descrizione di strutture dati attraverso l'uso di marcatori (tag). XML venne sviluppato da un XML Working Group²¹, formato sotto la supervisione del W3C nel 1996. Il gruppo era diretto da Jon Bosak della Sun Microsystems con la partecipazione attiva di un XML Specialist Interest Group²² organizzato dal W3C.

Tra gli obiettivi progettuali di XML vi era la necessità di rendere facile lo sviluppo di programmi che elaborino documenti XML. Questo perché l'adozione di questo linguaggio sarebbe stata proporzionale alla disponibilità di strumenti legati ad XML e la proliferazione di questi sarebbe stata la conferma del raggiungimento di questo obiettivo.

Le applicazioni XML consistono sia in tool che consentono di editare e visualizzare un documento XML, sia in API per la sua elaborazione. Il motivo per preferire delle API Java rispetto ad altri linguaggi come il Perl o

²⁰ XML è considerato un linguaggio "universale" in quanto gli autori sono liberi di definire nuovi marcatori e nuovi linguaggi per descrivere le strutture dei loro dati.

²¹ Conosciuto originariamente come SGML Editorial Review Board.

²² Precedentemente conosciuto come SGML Working Group.

C++, risiede nell'alta disponibilità delle stesse e nella loro portabilità. Sun ha infatti incluso un supporto standard per la manipolazione di XML per tutte le piattaforme al fine di evitare ai programmatori la necessità di costruirsi delle classi proprie. Cosa che porterebbe, ed ha inizialmente portato in effetti, alla realizzazione di implementazioni non standard, generando applicazioni pseudo-compatibili. Con l'introduzione delle API Sun, invece, esse fanno parte della piattaforma Java standard.

Se si volesse definire un paragone tra le tecnologie HTML/CGI e XML/Java, si potrebbe dire che mentre nel primo caso le funzionalità sono totalmente demandate al Server, nel secondo si permette una maggiore capacità di manipolazione dei dati a livello Client, riducendo nel contempo il traffico di rete. In quest'ottica, nel momento in cui si passano dei dati ad un browser, gli stessi dati, inviati una volta sola, possono essere trattati dal Client stesso in modo da ottenere tabelle, grafici, statistiche, report e quant'altro di utile. Se il trattamento dal lato browser venisse effettuato da un'applet Java, si sarebbe trovato un formato di scambio di dati semplice, potente, rapido e scalabile.

2.3 – L'operazione di Parsing XML in Java

Per poter manipolare i file XML attraverso Java sarebbe necessario aprire un file XML ed eventualmente il file DTD²³ associato, attraverso quest'ultimo creare una tabella dei nodi presenti e una volta creato l'albero, interpretare il file XML per estrarne i dati. Fortunatamente questo lavoro così complesso non è richiesto perché come accennato esistono delle classi standard che permettono di realizzare tutto ciò: sono i *parser*, classi in grado di interpretare la struttura ad albero del file XML, permettendoci di trattare l'oggetto risultante come qualcosa di più gestibile tramite Java.

L'operazione di *parsing* è di fondamentale importanza nell'utilizzo di un documento XML: realizzare il parsing significa scorrere il documento XML, estrapolando da esso tutte le informazioni di cui si ha bisogno. Quindi

²³ Si rammenta che il file DTD definisce la *Document Type Definition* che contiene la definizione dei vari elementi/nodi presenti nel file XML.

un parser è un modulo software che si colloca tra l'applicazione ed il documento XML.

Esistono due tipi di parser: *validanti* e *non validanti*. I primi, oltre a controllare se un documento è ben formato, verificano anche che esso sia valido, cioè fedele alle regole definite nella sua DTD. I parser non validanti invece, si preoccupano solo di controllare se il documento è ben formato.

2.3.1 – Parser DOM e SAX

Vi sono due modi per interfacciare un parser con un'applicazione: usando un'interfaccia *object-based* oppure *event-based*.

Con il primo approccio di tipo *object-based*, il parser costruisce esplicitamente in memoria un albero che contiene tutti gli elementi del documento XML. Grazie ad una serie di regole è possibile spostarsi in vari modi attraverso l'albero. Il vantaggio di questa metodologia è che ci si può riferire ad un qualsiasi nodo dell'albero a prescindere dal nodo in esame. L'aspetto negativo è che questo approccio richiede che il software mantenga in memoria l'intero albero per tutto il tempo dell'analisi, con conseguente dispendio di risorse, a discapito di velocità ed efficienza, consentendo però un controllo totale sui dati. Questo metodologia denominata DOM²⁴, mostrando al parser l'intera struttura del documento, permette anche il controllo sul fatto che le modifiche apportate siano coerenti con la struttura fino ad ora utilizzata.

Invece con il secondo tipo di approccio *event-based* (o event-driven) il parser durante la lettura del documento XML genera un evento ogni qual volta incontra un elemento, un attributo o del testo. Vi sono eventi per i tag

²⁴ DOM = Document Object Model. Il W3C DOM consiste di diversi livelli, ognuno dei quali è basato sui precedenti e ne estende le API. L'insieme di questi livelli è definita come la *specifica DOM*: vi è il *livello zero* e il W3C DOM è costruito su questa tecnologia; il *livello uno* si concentra sui modelli di documenti HTML e XML implementando funzionalità per la navigazione del documento e la sua manipolazione; il *livello due* include uno modello ad oggetti di stile, definendo funzionalità per la manipolazione delle informazioni di stile legate al documento, eventi, attraversamento della struttura di un documento. Ulteriori livelli consentiranno il modo di interagire con l'utente, manipolare il DTD e gestire un modello di sicurezza. In particolare al momento sono stati definiti i requisiti del *livello tre* che definirà, tra le altre cose, lo spostamento di un nodo fra documenti e l'ordinamento dei nodi, l'estensione del modello di eventi di DOM livello 2, content model e validation use case, caricamento e salvataggio di un documento XML, gestione delle viste e formattazione.

di apertura e di chiusura, per gli attributi, il testo, le entità, i commenti, ecc. Il concetto *event-driven* è riscontrabile nella programmazione di interfacce grafiche in cui l'applicazione risponde all'utente a seconda dell'evento che questi ha scatenato. Esistono programmi che permettono di definire quali azioni intraprendere quando il parser incontra un certo tag durante l'analisi. Il vantaggio di questo approccio è quello di essere veloce e semplice da sviluppare, mentre lo svantaggio è che il processo vede solo un ristretto insieme di informazioni del documento sottoposto ad analisi.

La scelta di utilizzare un parser DOM anziché SAX dipende fondamentalmente dal tipo di applicazione che lo utilizza. Generalmente l'uso di un parser DOM (e più in generale object-based) è più appropriato in applicazioni dove è richiesta una manipolazione di documenti XML, come ad esempio in editor²⁵, browser, processori XSL, ecc. e nel caso in cui vi sia bisogno di analizzare il file XML in modo da poter accedere a qualsiasi parte dell'informazione. Mentre l'uso di un parser SAX o di altri event-based è più consono nei casi in cui l'applicazione non richieda la memorizzazione dei dati in XML ma in un altro formato, come ad esempio un tool per importare documenti XML in un database: in questo caso il formato dell'applicazione è la struttura del database e non quella del documento XML.

2.3.2 – Introduzione ai Parser disponibili

Come discusso, per evitare le difficoltà legate al parsing di dati XML, quasi tutti i programmi necessitano di usare un parser XML per la lettura del documento. Vi sono dozzine di parser XML disponibili, ognuno con una specifica licenza che lo accompagna. A meno che si abbia bisogno di specifiche particolarità, è piuttosto bassa la possibilità che il singolo utente possa scrivere un parser migliore rispetto a quelli prodotti da Sun, IBM, Apache XML Project e numerosi altri già realizzati. Per quanto riguarda le strategie SUN, Java 1.4 è la prima versione di Java ad includere un parser XML come standard nella distribuzione del JDK. Nelle precedenti versioni di Java vi era la necessità di scaricare un parser dal Web e di installarlo nel

²⁵ Un editor come XMLStudio ad esempio.

modo tradizionale, tipicamente inserendo il file `.jar` nella directory `jre/lib/ext`. Però, pur disponendo di Java 1.4 si potrebbe desiderare rimpiazzare il parser standard con uno differente che fornisca ulteriori proprietà o semplicemente che sia più veloce nell'uso sui documenti.

2.3.3 – *Come scegliere un Parser?*

Quando si tratta di scegliere una libreria per il parsing, diversi fattori entrano in gioco per determinare la scelta più opportuna. Questi includono quali caratteristiche ha il parser, quanto costa, quali API implementa, quanti bug presenta e quanto velocemente effettua l'operazione di parsing. Le specifiche di XML 1.0 permettono ai parser una certa tolleranza in riferimento a come implementano le specifiche stesse.

Esistono sia *parser validanti*, sia altri che leggono l'istanza di un documento ma non realizzano tutti i controlli legati alla caratteristica di essere ben formati. Tecnicamente questi parser non sono permessi dalle specifiche XML, ma ve ne sono tutt'ora in circolazione.

Se il documento che si desidera processare ha una DTD allora la scelta dovrebbe ricadere sull'uso di un *parser validante*. Non è necessario attivare la verifica di validità se non lo si desidera. Comunque XML è stato progettato in modo che non si sia realmente sicuri del contenuto del documento XML senza la lettura della sua DTD. In alcuni casi può essere notevole la differenza di processamento tra un documento la cui DTD è stata esaminata e lo stesso documento senza la suddetta analisi. Per esempio, un parser che legga la DTD restituirà valori di attributi di default ma uno che non la leggesse, non potrebbe farlo. In linea di massima, anche se oggi non è strettamente consigliabile, si potrebbe scegliere di adottare un parser non validante nel caso si fosse certi che nessuno dei documenti da processare abbia la DTD.

Di seguito verranno elencate alcune caratteristiche degne di menzione che permettono la distinzione tra i vari parser:

- Allo stato attuale tutti i parser degni di nota supportano i *namespaces*. Attualmente solo Xerces e Oracle supportano la validazione degli *schema*²⁶ anche se altri parser aggiungeranno questa caratteristica.
- La maggior parte dei parser supporta sia *SAX* che *DOM*. In ogni modo vi sono alcuni di essi che supportano solo SAX e almeno una coppia che implementano delle API proprietarie. Se si desidera utilizzare DOM e SAX, bisogna essere certi che il parser possa manipolarle: Xerces e Crimson sono due parser, illustrati nelle prossime pagine, che lo consentono.
- Una delle caratteristiche spesso trascurate nella scelta di un parser è la *licenza* sotto la quale esso è pubblicato. La maggior parte dei parser sono *free* e siccome sono essenzialmente delle librerie di classi collegate dinamicamente al proprio codice (tutte le librerie Java lo sono) ed i parser sono spesso rilasciati sotto una licenza aperta, spesso può convenire adottare l'uso della GPL usata per esempio da Xerces. Secondo essa, eventuali cambiamenti apportati al parser devono essere donati alla comunità. Vi sono pochi parser *non free* e uno di essi è quello di IBM; comunque quelli gratuiti e liberi risultano essere generalmente più che adeguati per i compiti da svolgere.
- Un altro aspetto spesso trascurato nella scelta di un parser è la *correttezza*: quante delle specifiche rilevanti sono implementate e come? Alcuni parser risultano essere certamente più affidabili di altri anche se la proprietà che devono meglio eseguire è il verificare se un documento è ben formato: per essere seriamente considerato un parser deve svolgere perfettamente il suo lavoro in quest'area. Errori di validità non sono così importanti anche se sono comunque significativi: diversi programmi ignorano la validità e conseguentemente i bug di validazione del parser.

²⁶ Con il termine *XMLSchema* si intende un documento XML che dovrà sostituire, con numerosi miglioramenti, il DTD. Quindi vi è differenza tra il verificare la validità o meno di un documento XML dotato di DTD dal controllo di validazione di un documento XML descritto attraverso uno *XMLSchema*.

- L'ultima considerazione per la scelta riguarda *l'efficienza*: quanto veloce risulta essere un parser e quanta memoria utilizza. Questa proprietà dovrebbe avere rilevanza ridotta rispetto alle precedenti: sino a che si utilizzano API standard e si mantiene il codice dipendente dal parser al minimo, risulta sempre possibile cambiare il sottostante parser se quello che si è scelto risulta essere inefficiente. La velocità del parsing è strettamente legata alla modalità in cui avviene l'I/O: se il documento XML è trasmesso sulla rete, è possibile che la velocità con cui i dati si muovono formino un collo di bottiglia non dipendente dal parser XML. In altre situazioni dove invece il file XML è letto dal disco, il tempo speso per leggere i dati può essere significativo. In ogni caso, sia che si leggano i dati dalla rete o dal disco, risulta opportuno bufferizzare gli stream usando `BufferedInputStream` oppure a livello di carattere con un `BufferedReader`.

2.3.4 – I principali Parser sul mercato

Xerces-J (e IBM XML4J): è un parser sviluppato dall'Apache XML Project ed è reperibile all'indirizzo <http://xml.apache.org/xerces-j>. E' un parser molto completo che permette la validazione dei documenti e risponde in piena conformità alle specifiche XML 1.0 e Namespaces. Supporta completamente le API di SAX2 e DOM2 come anche JAXP. E' altamente configurabile e adattabile alle principali esigenze di parsing. L'Apache XML Project ha pubblicato Xerces-J sotto una licenza molto libera e aperta. Gran parte dello sviluppo legato a Xerces-J è stato realizzato da diversi programmatori IBM in quanto è basato sul parser IBM per Java²⁷. Tecnicamente non vi sono significative differenze tra i parser *Xerces-J* e *XML for Java* di IBM. La reale differenza è insita nel fatto che se si lavora per una grande compagnia può essere più utile acquistare il software da qualcuno come IBM che può essere ritenuto responsabile dello sviluppo del

²⁷ Disponibile all'indirizzo <http://www.alphaworks.ibm.com/tech/xml4j>

progetto del parser. Altrimenti la scelta può convenientemente ricadere su Xerces-J.

Crimson (o Java Sun Project X): Crimson è il parser che Sun ha deciso di allegare con il JDK 1.4. Sun ha inizialmente donato Crimson all'Apache XML Project, ma in un secondo momento ha ritirato la donazione, per cui il codice sorgente non è disponibile anche se i binari sono gratuitamente scaricabili. Sun è tutt'ora il principale sviluppatore di Crimson, nello stesso modo in cui IBM sviluppa Xerces. Malgrado tutto la maggior parte del progetto Apache sembra gravitare attorno a Xerces piuttosto che a Crimson. Crimson supporta all'incirca le stesse API e specifiche di Xerces e quindi SAX2, DOM2, JAXP, XML 1.0, Namespaces in XML, ecc. Allo stato attuale Crimson è più soggetto a bug rispetto a Xerces: qualche esempio dimostra che documenti ben formati sono segnalati come malformati mentre Xerces esegue il loro parsing senza problemi. Ma questi sono problemi di giovinezza legati a Crimson in quanto gli ingegneri di Sun hanno deciso di implementare una realizzazione differente rispetto agli sviluppatori di IBM: Crimson è stato progettato per essere significativamente più veloce di Xerces-J, anche se allo stato attuale non vi sono dimostrazioni effettive. Quando Sun ha inizialmente rilasciato Crimson ha affermato che le sue prestazioni in termini di velocità erano diverse volte superiori rispetto al concorrente Xerces. IBM ha allora deciso di eseguire gli stessi benchmark e ha ottenuto esattamente il risultato opposto: essi hanno affermato che Xerces era diverse volte più veloce di Crimson. Dopo un paio di settimane di discussioni piuttosto accese in varie mailing-list, la verità è venuta a galla: Sun ha pesantemente ottimizzato Crimson per la Sun Virtual Machine e naturalmente eseguivano i loro test sulla Virtual Machine di Sun. Allo stesso modo IBM ha pubblicato la propria Java Virtual Machine e anche loro ottimizzavano i benchmark su di essa. Senza grosse sorprese le ottimizzazioni di Sun non venivano eseguite molto bene su Virtual Machine non-Sun e allo stesso modo le ottimizzazioni di IBM non erano molto efficienti su Virtual Machine non-IBM. Attualmente le prestazioni sia di Xerces-J che di Crimson sembrano essere egualmente veloci su hardware

equivalenti non curando la VM. Il beneficio sostanziale di Crimson è dato dal fatto che è allegato con il JDK 1.4. Esso non lavora con Virtual Machine precedenti a Java 1.1. Così se si sa di poter lavorare su di un ambiente Java 1.4 o successivo, non vi è da preoccuparsi in riferimento all'installazione di librerie Jar extra per realizzare il parsing di documenti XML. Si può tranquillamente scrivere il proprio codice per le classi standard SAX e DOM. Comunque, se si desidera usare un parser differente da Crimson è sempre possibile farlo: è sufficiente installare i file Jar per esempio di Xerces-J o di qualche altro parser e rendere esplicito il suo caricamento.

2.3.5 – Confronto tra i vari Parser esistenti

La qualità di un parser è ampiamente definita dalla sua conformità allo standard XML. Una suite per il test è stata definita²⁸ da OASIS²⁹ e consiste in un insieme di più di 1000 documenti validi e non validi, definiti per verificare la bontà dei parser e le loro effettive capacità e prestazioni. Questo test consisteva nel verificare che i parser accettassero i documenti effettivamente validi e rifiutassero gli altri.

Ad ottobre 2000, data in cui è stato realizzato il test in esame, solo il parser di Sun superava tutti i test; XML4J rifiutava alcuni documenti validi contenenti caratteri speciali UTF-16: questo problema potrebbe portare difficoltà nel processare documenti scritti in lingue diverse dall'inglese.

La versione Java del parser XP di James Clark, pur essendo ancora allo stato di beta, ha superato la maggior parte dei test e rileva se un documento è ben formato, non se è valido. XP è stato progettato in Java con particolare riguardo all'essere conforme al 100% alle specifiche XML. I suoi sviluppatori si sono anche concentrati nel renderlo un parser particolarmente veloce ad elevate prestazioni.

²⁸ I test possono aver preso in considerazione il tempo di costruzione del documento, il tempo di attraversamento dell'albero, il tempo di modifiche al documento XML, l'uso di memoria da parte del documento, ecc.

²⁹ OASIS è l'Organization for Advancement of Structured Information Systems. E' un consorzio internazionale senza fini di lucro che tra le altre cose dovrebbe decidere in una tavola rotonda tra i vari produttori software e ricercatori, gli schemi che tutti i documenti testuali di un certo ambito dovrebbero seguire.

Il parser di Oracle sembra essere basato su codice SGML e accetta alcuni documenti che sono illegali in XML ma sono perfettamente legali nella più flessibile definizione SGML.

Il parser di Microsoft MSXML verifica anche la validità ed è scritto in Java: sembra però svolgere un'analisi non approfondita analizzando documenti illegali.

	IBM XML4J Apache Xerces	Sun Project X	Microsoft MSXML	Oracle XML Parser for Java	James Clark XP
Controllo documento ben formato	+	+	+	+	+
Controllo documento valido	+	+	+	+	-
XML-Schema	+	-	-	-	-
Namespace	+	+	+	+	-
XSL-T	con LotusXSL	-	+	+	-
Java	+	+	-	+	+
Win32	in Java	in Java	in Java	+	-
SAX 1.0	+	+	+	+	+
SAX 2.0	+	-	+	-	-
DOM Livello 1 1.0	+	+	+	+	-
DOM Livello 2 1.0	+	-	-	-	-
Open Source	+	-	-	-	+
Nome download	<u>XML4J Xerces</u>	<u>ProjectX</u>	<u>MSXML</u>	<u>ORACLE</u>	<u>XP</u>

Tabella 2 - Analisi comparativa dei diversi parser disponibili

Il download dei vari parser è possibile ai seguenti indirizzi:

- XML4J Xerces: <http://alphaworks.ibm.com/tech/xml4j>
- Sun Project X: <http://java.sun.com/products/xml/index.html>
- MSXML: <http://msdn.microsoft.com/xml/default.asp>
- ORACLE Parser for Java: http://technet.oracle.com/tech/xml/parser_java2
- James Clark XP: <http://jclark.com/xml/xp/index.html>

Tutti i parser menzionati costituiscono dei buoni candidati per costruire un'applicazione XML e quando si utilizzano API standard come

SAX o DOM, dovrebbe essere possibile sostituire il parser utilizzato senza modifiche sostanziali al codice.

2.3.6 – La scelta di Sun Project X (o Crimson) per XMLStudio

J2SE 1.3 è stato rilasciato a Maggio del 2000 mentre J2SE 1.4 all'inizio del 2002. La versione 1.5, con nome in codice Tiger, verrà rilasciata probabilmente verso la metà del 2003. Versioni intermedie, ovvero 1.4.x definite di “mantenimento” verranno distribuite prima della 1.5 per fissare dei bug o migliorare le prestazioni.

Il JDK 1.4 non è considerato una “major release” nel senso che non cambia sostanzialmente il modo di lavorare in Java come il salto rappresentato dal JDK 1.1 a Java 2 diversi anni fa. Bensì porta con sé ridotti miglioramenti atti a rendere il JDK 1.4 più robusto.

Come discusso, con la release 1.4 del JDK sono state rese standard diverse caratteristiche tra cui il parsing XML che ha brillantemente superato i test OASIS come accennato: il supporto per XML DOM e SAX è quindi standard e garantito. Vista però la velocità con cui cambia il mondo dell'XML, non vi è modo per includere le ultime API nel JDK se non attraverso alcuni pacchetti JAX aggiuntivi che rappresentano le *Java Api for XML*.

Come risultato del processo tra Sun e Microsoft del 2001, quest'ultima esclude da Windows XP il supporto a Java. Questa forte decisione è stata presa, secondo ciò che ufficialmente afferma Microsoft a seguito del contratto che legava in Windows il JDK e la sua distribuzione limitata alla versione 1.1 solo sino al 2008. Microsoft ha dunque scelto di non supportare Java in nessuna sua forma. I browser che utilizzano Java 2 richiedono un Java Plug In pari a 5Mb che consente ai browser Internet Explorer o Netscape di eseguire le applet di Java 2. Attualmente solo il browser Netscape 6.x ha insita in fase d'installazione l'opzione per includere questo Plug In.

Un punto cruciale di sviluppo del JDK 1.4 è stato rappresentato dall'implementazione dell'interfaccia utente Swing: essa permette agli

sviluppatori di creare “Rich user interfaces” ovvero interfacce utente ricche e complete di tutti gli elementi atti a realizzare applicazioni di qualità. Grazie all’uso del *PLAF*³⁰, disponibile in Java e nelle Swing, le applet e le applicazioni possono avere un’interfaccia utente in stile Windows, Solaris, Motif e Macintosh. Il “*look and feel*” di default in Java 2 si chiama Metal e presenta delle innovazioni nel J2SE. Naturalmente Sun potrebbe fornire nuove PLAF in futuro, eventualmente per adattarsi alla nuova impostazione grafica di Windows XP, rimanendo al passo con la concorrenza anche dal semplice punto di vista estetico in quanto le interfacce Swing consentono di realizzare tutto ciò che in ambiente Windows è possibile.

Java 2SE include anche il supporto per le espressioni regolari³¹, ingrediente di base per esempio del linguaggio Perl: mentre sino ad ora esse erano tipicamente rese disponibili in Java come add-on solitamente *free* da terze parti, ora sono uno standard.

Grazie a questa breve panoramica si evince che Java³² rappresenta, in virtù di queste caratteristiche e della sua portabilità, una piattaforma di sviluppo che rimane allo stato dell’arte per la realizzazione di applicazioni Web e standalone come XMLStudio.

³⁰ Pluggable look and feel: XMLStudio ha opzioni per modificare il suo aspetto secondo questo standard.

³¹ Utilizzate in XMLStudio grazie al *package java.util.regex*.

³² A giugno del 2001 il Java 2 SDK è stato scaricato 5 milioni di volte da utenti di tutto il mondo. Si stima che gli sviluppatori Java siano 2,5 milioni e che dovrebbero crescere a 4 milioni entro il 2003 in riferimento a dati IDC.

CAPITOLO 3

L'editor integrato XMLStudio

3.1 – L'idea alla base di XMLStudio

XMLStudio è un'applicazione scritta in Java e sviluppata utilizzando il JDK in versione 1.4. XMLStudio permette di decorare file di testo (ed XML) con tag, effettuare il controllo di validità rispetto ad una DTD, inferire un file DTD. L'idea di questa applicazione parte dall'analisi dei documenti RFC.

3.1.1 – Che cos'è un documento RFC

I documenti denominati *Request for Comments* (richiesta di osservazioni), la cui sigla è *RFC*, sono note di lavoro della comunità di ricerca e di sviluppo Internet. Riguardano qualsiasi argomento legato alla trasmissione dati tra elaboratori e spaziano dai resoconti di riunioni alla specifica di uno standard (normativa). Le Request For Comments vengono presentate all'RFC Editor Jon Postel (rfc-editor@isi.edu).

La maggior parte delle RFC sono descrizioni di protocolli o di servizi di rete e spesso forniscono procedure e formati dettagliati per la loro implementazione. Altre presentano il risultato di studi su certe politiche, oppure riassumono i lavori dei comitati o dei seminari tecnici. Se non viene specificato altrimenti, tutte sono di dominio pubblico.

Le RFC, pur non essendo pubblicazioni di riferimento, ricevono spesso un esame tecnico da parte di altre task forces, di singoli esperti, oppure dell'RFC Editor, a seconda dei casi. Attualmente la maggior parte degli standard internet sono pubblicati sotto forma di RFC, sebbene non tutte le RFC specificano degli standard. Chiunque può sottoporre un documento perché venga presentato come RFC. La presentazione deve avvenire attraverso la posta elettronica indirizzata all'RFC Editor.

Quando un certo documento ha ricevuto uno specifico numero di RFC che lo identifica³³ e viene pubblicato, non viene più sottoposto a revisione, né viene ripubblicato con lo stesso numero. Non esiste il problema della versione più aggiornata di un certo RFC; tuttavia è possibile che un protocollo come per esempio *ftp* venga perfezionato e nuovamente documentato molte volte in più RFC.

Gli RFC sono documenti testuali e un mirror del loro archivio è anche presente in Università all'indirizzo <http://star.mfn.unipmn.it/rfc>.

Questi file vengono studiati da alcuni ricercatori della nostra Università del Piemonte Orientale e dagli studiosi di tutto il mondo che si occupano di internet, dei suoi protocolli, della sua evoluzione. Reperire e trovare informazioni significative ricercando all'interno dell'archivio dei documenti RFC non è sempre immediato: i motori di ricerca, come discusso nel primo capitolo, individuano le singole parole senza tenere conto del significato semantico delle stesse, restituendo spesso dei risultati non significativi. Introducendo delle parole chiavi come "internet", "protocol", "security", il numero dei risultati sarà sicuramente molto elevato e ben difficilmente i primi individueranno ciò che si stava effettivamente cercando.

3.1.2 – Aggiungere conoscenza ai documenti RFC

L'idea originaria era inizialmente quella di realizzare un'applicazione che permettesse la trasformazione di documenti RFC testuali in XML in modo che il ricercatore leggendoli potesse marcare dei paragrafi di testo

³³ Ad esempio 210, 1017, 1538, 2536, ecc.

attraverso dei tag XML da lui stesso definiti, al fine di aggiungere al documento della conoscenza. In questo modo, ad esempio, marcando un paragrafo riguardante la sicurezza con il tag <sicurezza>, sarebbe possibile effettuare ricerche più accurate perché si sarebbe certi che quel paragrafo contiene del contenuto importante relativo a questo argomento. Oppure, per esempio, un marcatore del tipo <autore> andrebbe ad identificare i nomi degli autori dei singoli RFC con la possibilità di restituire in un'eventuale ricerca, tutti e soli i documenti RFC di cui una persona ne è l'autore. In questo modo verrebbero automaticamente scartati i documenti nei quali il soggetto viene citato con nome e cognome ma dei quali non ne è l'autore.

3.1.3 – JRfc: prima bozza di XMLStudio

XMLStudio nasce come un'applicazione Java che utilizza le librerie Swing come interfaccia utente. Inizialmente sono state implementate le funzioni di base che ogni editor testuale dovrebbe avere: operazioni di apertura e salvataggio dei file testuali, funzioni di taglia, copia, incolla, ricerca del testo, annullamento e ripristino dell'ultima operazione effettuata, modifiche all'editor quali il cambio del font e dei suoi colori, dello sfondo e del *look and feel*, stampa dei file, ecc.

Un primo nome adatto all'applicazione ho ritenuto essere JRfc in relazione al suo sviluppo in linguaggio Java e al suo uso nell'elaborazione e trasformazione dei documenti testuali RFC nel formato XML. JRfc implementava già quasi tutte le funzioni oggi disponibili in XMLStudio ad eccezione della inferenza di una DTD, della creazione di un foglio di stile CSS e del controllo di validità di un documento XML dotato di DTD.

3.1.4 – Il progetto Xdidattica

Terminato lo sviluppo di questa prima parte dell'editor, nell'effettuare ricerche in internet in riferimento a Java, XML e tecnologie affini, ho trovato un sito internet, non ancora indicizzato dai motori di ricerca, relativo ad un progetto³⁴ legato all'applicazione di XML in problemi di analisi e

³⁴ Il sito internet ufficiale del progetto è <http://www.ebookpertutti.com>

costruzione di documenti testuali nell'ambito di una sperimentazione da parte del Liceo Scientifico Lussana, dell'Università di Milano (Crema), dell'Ufficio Scolastico Provinciale di Bergamo, della Casa Editrice "La Scuola", con la partecipazione di docenti di varie scuole e studenti universitari. L'obiettivo di questo progetto, tutt'ora in corso, consiste nel verificare l'utilità dell'introduzione di concetti legati all'impiego di XML nella didattica. La sperimentazione avviene grazie ad una mailing-list internet e all'uso di un software specifico Visual_XML³⁵, utilizzato su esercizi specifici da consegnare. L'obiettivo consiste nell'individuare nei testi elementi di struttura per aggiungere informazione utile ai documenti al fine di utilizzarla con fini archivistici, di ricerca, di comprensione, di didattica:

"Qui non si intende anticipare probabili sviluppi (in realtà già nei laboratori). Tuttavia va segnalato che senza una forte consapevolezza su questi aspetti, il futuro della gestione umana sarà sotto il controllo della Intelligenza Artificiale (si noti che non ho affermato delle tecniche della Intelligenza Artificiale). La consapevolezza della necessità di comprendere l'evoluzione della comunicazione umana nella forma più significativa (i testi scritti) per la evoluzione dell'uomo e della società... è quindi una delle molle che ha proposto la sperimentazione."³⁶

3.1.5 – Le possibilità d'impiego di XMLStudio

Anche alla luce del progetto Xdidattica, si è constatato che al termine della realizzazione dell'editor JRfc, esso consentiva non solo l'elaborazione XML dei file testuali Rfc ma anche l'analisi e la possibilità di marcare qualsiasi altro documento. Si è quindi deciso di modificare il nome e la finalità d'uso dell'editor per non limitarne la sua applicabilità ad un ambito ristretto viste le vaste possibilità d'impiego. XMLStudio in sintesi può essere

³⁵ L'editor Visual_XML verrà preso in considerazione e analizzato nel Capitolo 4 della tesi.

³⁶ Parte dell'introduzione relativa alla sperimentazione Xdidattica all'indirizzo internet:
<http://www.ebookpertutti.com/xmldidattica/indice.htm>

definito come un *editor integrato XML*: integra cioè al suo interno diversi componenti che consentono di svolgere diverse funzionalità sui file XML.

XML è stato adottato perché può essere considerato “il formato dei formati universale”³⁷, scaturito in parte dal buon accordo dei produttori software che adottandolo hanno la possibilità di accordarsi su quali schemi utilizzare secondo il tipo di file.

XMLStudio lascia l'utente completamente libero di utilizzare file XML definiti da altri come base per l'inserimento dei tag XML ma consente anche di creare i propri o modificare gli esistenti.

3.2 – Installazione ed uso di XMLStudio

Come già discusso XMLStudio per essere eseguito necessita dell'uso di Java 2 in versione 1.4. Risulta quindi indifferente per il suo utilizzo scaricare l'intera suite di sviluppo SDK oppure il solo JRE (Java Runtime Environment)³⁸.

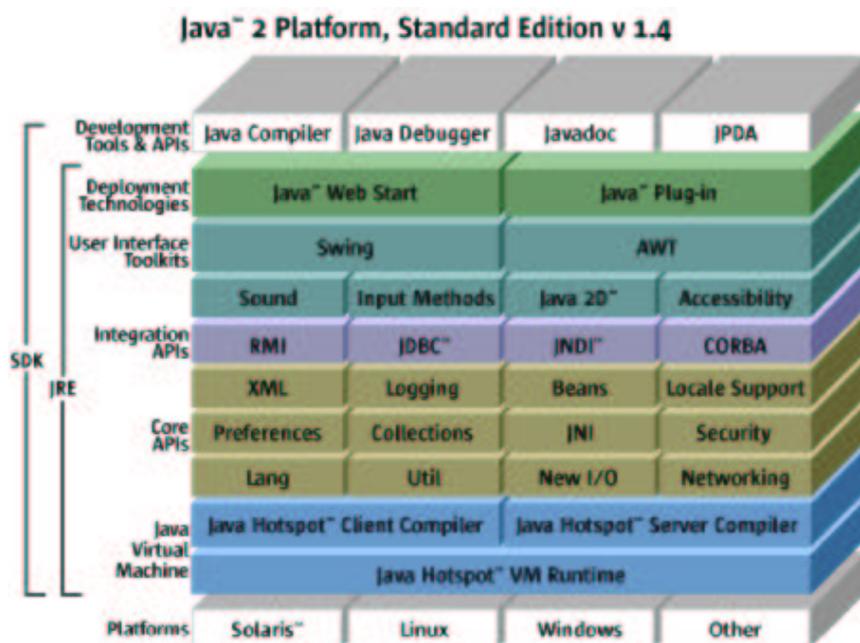


Figura 1 - Struttura di Java 2, Standard Edition v. 1.4

³⁷ Un possibile utilizzo di XML dovrebbe consentire agli utenti di tutto il mondo l'uso di un comune formato testuale in chiaro per lo scambio di documenti o anche l'adozione di un formato dati universale nel commercio elettronico.

³⁸ Entrambi sono scaricabili gratuitamente dall'indirizzo <http://java.sun.com/j2se/1.4/download.html>

3.2.1 – L'interfaccia di XMLStudio e le sue componenti

L'editor XMLStudio, come accennato, è costituito da un'interfaccia realizzata grazie all'uso delle librerie Swing. Essa è composta da quattro elementi: la barra dei menù, un'area relativa all'elaborazione del testo, un pannello relativo ad un albero che mostra un' altra vista del documento XML ed infine un pannello che consente di definire e modificare i tag e i relativi attributi o valori secondo una visione ad albero in un formato file anch'esso XML.

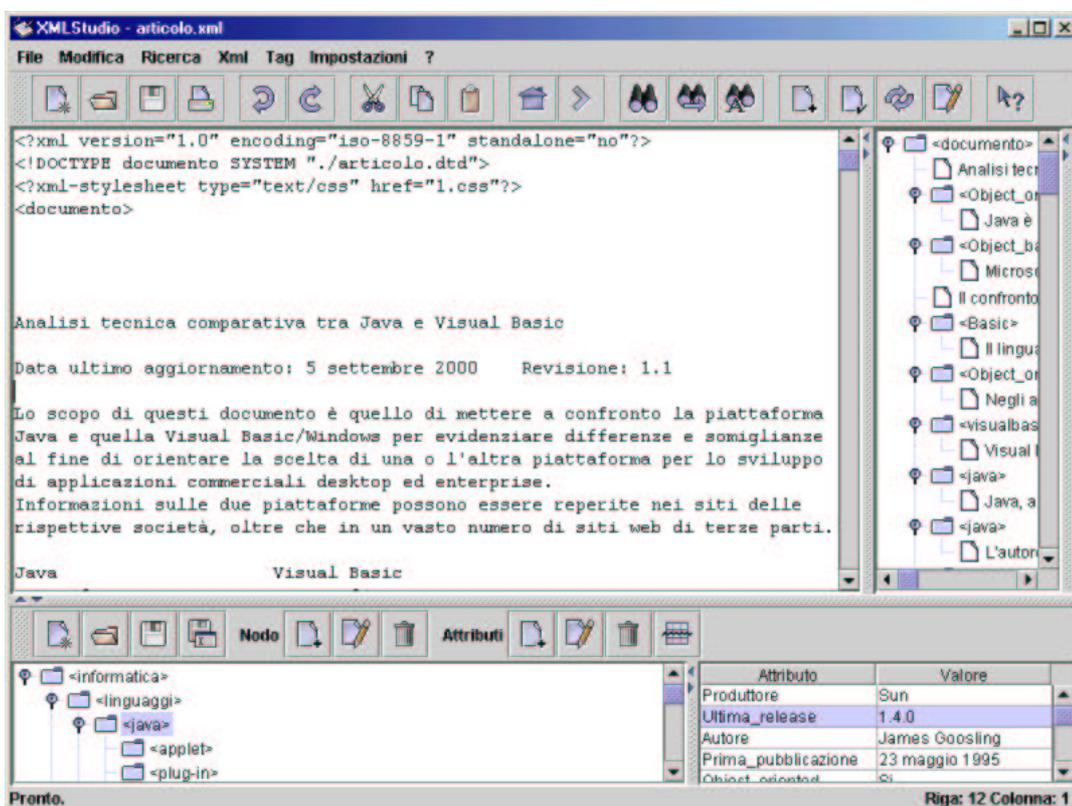


Figura 2 - L'interfaccia grafica di XMLStudio utilizza le librerie Swing

3.2.1.1 – La barra dei menù

Quest' oggetto rappresenta la *barra dei menù* ad accesso rapido dell' applicazione. Posizionando il cursore del mouse su ogni icona, compare una breve spiegazione della funzionalità svolta. Ogni comando mostrato

dalle icone di questa barra è anche presente nei menù a tendina di XMLStudio.



Figura 3 - La barra dei menù

3.2.1.2 – L'area di elaborazione del testo

E' l' area che visualizza il documento testuale caricato sul quale lavorare; è possibile selezionare il testo tramite il mouse e anche agire sui delimitatori laterali dell'*area di testo* per modificarne la dimensione. Premendo il pulsante destro del mouse su quest' area, compare un menù che mostra sei funzioni applicabili al testo selezionato:

- Tag testo, Seleziona paragrafo tag, Elimina tag testo.
- Taglia, Copia, Incolla.

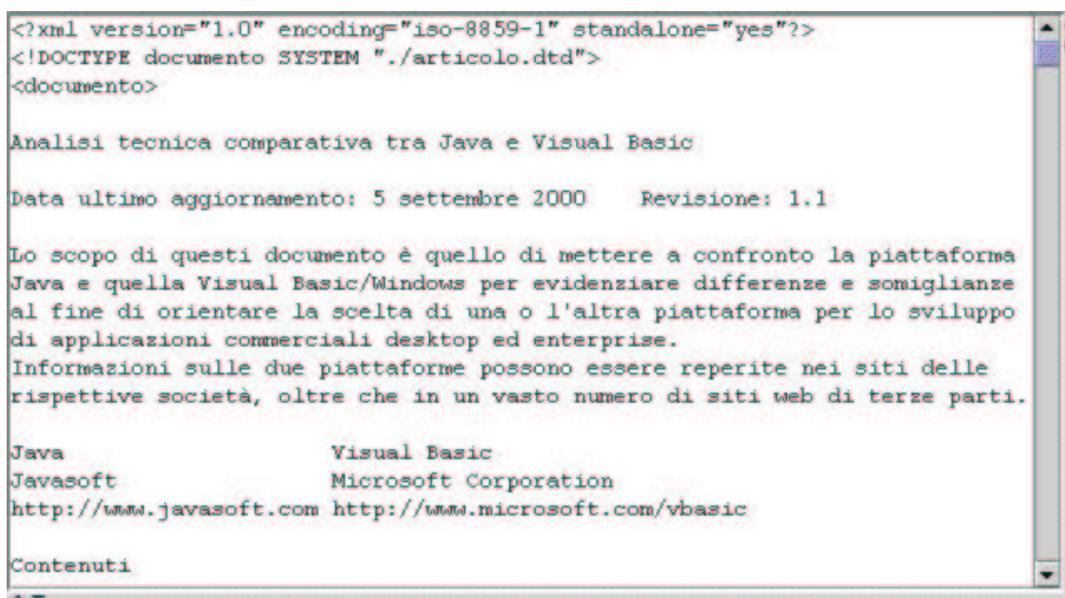


Figura 4 - L' area testuale di elaborazione del documento XML

3.2.1.3 – L'albero che mostra il documento XML

L'*albero XML* mostra un' altra vista di tipo strutturale del documento XML visualizzato nell' area di testo. E' possibile spostare i delimitatori per

modificare le dimensioni dell' area visibile e far comparire gli attributi e i valori relativi ai nodi dell' albero.



Agendo sul comando  per l'aggiornamento dell'albero XML, vengono eseguite le seguenti operazioni: viene salvato il file XML sul quale si sta lavorando, viene verificato che il file XML sia ben formato ed in caso affermativo viene aggiornato l'albero XML relativo al documento aperto.

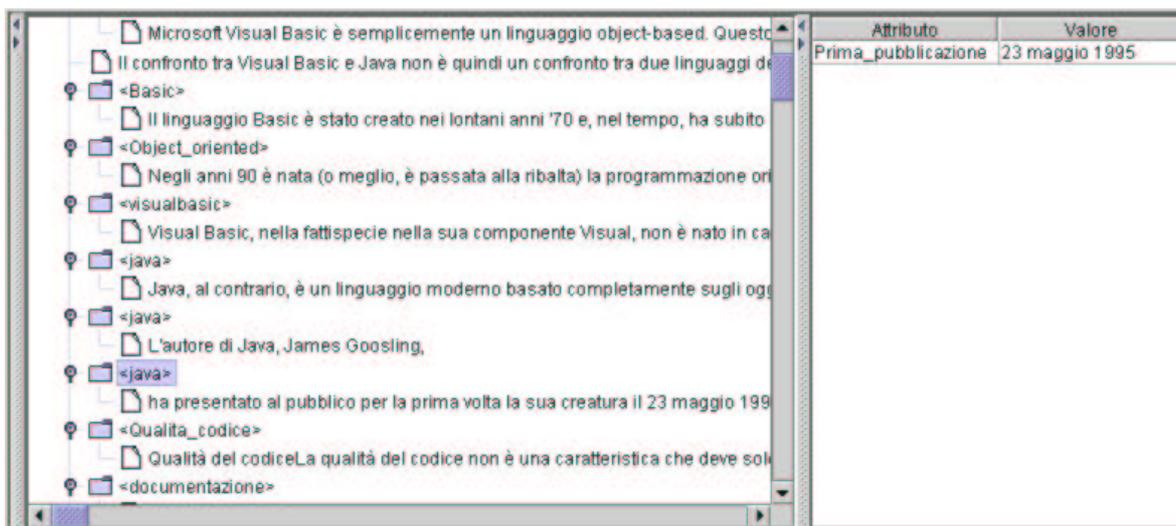


Figura 5 - L'albero che mostra la vista strutturale del documento XML

3.2.1.4 – Il pannello che consente di definire e modificare i tag

Questo pannello è composto da: un albero che mostra i marcatori (tag) inseriti dall' utente, una tabella che visualizza i rispettivi attributi e valori dei nodi (tag) dell' albero, una barra che mostra i comandi applicabili ai nodi dell' albero, agli attributi e valori della tabella e alla creazione, apertura e salvataggio del file XML relativo ai marcatori.

E' possibile impostare il file XML dei tag caricato al momento del lancio di XMLStudio attraverso il comando *File tag di default* del menù *Impostazioni*. I file XML dei marcatori vengono generalmente salvati nella directory *Tag* di XMLStudio. Viene sempre realizzato nel file *TagXmlBackup.xml* posto nella directory *Tag*, un backup dell' ultimo file dei marcatori salvato.



Figura 6 - Il pannello relativo ai marcatori (tag) e ai rispettivi attributi e valori

3.2.1.5 – I menù a tendina

Nella parte superiore dell'editor sono presenti i classici *menù a tendina* che permettono l'esecuzione di molti comandi che non compaiono nella *barra dei menù*. E' in particolare possibile realizzare le classiche operazioni che ogni editor consente sui file (apertura, salvataggio, stampa, richiamo rapido degli ultimi file aperti, ecc); modificare il documento XML secondo le classiche funzioni di taglia, copia, incolla; individuare e sostituire delle parole o spostarsi ad una determinata linea; rendere il documento ben formato e valido attraverso sostituzioni delle entità predefinite (caratteri di escape) e la creazione e inclusione del file DTD associato al file XML; creare un foglio di stile CSS associato al file XML per la visualizzazione nel browser; modificare nell'area testuale di elaborazione del documento XML il carattere, il colore, lo sfondo, la directory di lavoro, il file tag di default caricato al momento dell'avvio dell'editor, le dimensioni e l'aspetto della finestra, ecc.

3.2.2 – L'uso di SAX in XMLStudio

Il supporto per le API SAX (Simple API for XML) si trova nel package `org.xml.sax`; in particolare il package `org.xml.sax.ext` fornisce una classe accessoria usata in XMLStudio per rilevare un'eventuale eccezione nel momento in cui si chiede di verificare se un documento XML è ben formato: se esso non è ben formato il parser restituisce una `SAXException`.

3.2.3 – L'uso di DOM in XMLStudio

Il *Document Object Model* DOM, come discusso nel secondo capitolo, rappresenta la base per descrivere documenti strutturati. DOM identifica un modo per rappresentare gli elementi e gli attributi di un documento come una struttura logica utilizzabile attraverso la comune programmazione.

Nell'applicazione XMLStudio DOM è utilizzato sia per il parsing e la visualizzazione strutturata ad albero del documento XML elaborato nell'area testo, sia nella gestione relativa al file XML dei tag.

E' importante osservare che DOM non specifica un linguaggio di programmazione: gli elementi sono organizzati secondo relazioni simili a ciò che avviene nei comuni modelli di programmazione orientati agli oggetti.

Il modo più semplice per capire come DOM elabora un documento è osservare il *diagramma ad albero* relativo agli elementi, attributi e alle loro reciproche relazioni. *L'albero* rappresenta solo una metafora per la rappresentazione perché la reale implementazione di DOM potrebbe non rappresentare i dati in questo modo. Ma le modalità di manipolazione della struttura del documento si basano su questo tipo di visualizzazione, per cui risulta comodo concepirlo.

Le interfacce che DOM utilizza sono un *insieme di metodi* per lavorare sulla rappresentazione strutturale del documento dopo la sua parsificazione. Il processo di parsing crea la mappa strutturale del documento: in HTML le strutture sono note e comprese in anticipo. Il documento avrà certamente una radice di nome HTML che conterrà un elemento HEAD che a sua volta includerà alcuni insiemi limitati di possibili sotto-elementi con i rispettivi attributi. A loro volta questi elementi conterranno dati che saranno figli degli elementi e dei loro attributi secondo una codice di questo tipo:

```

<HTML>
  <HEAD>
    <TITLE>Un documento</TITLE>
  </HEAD>
  <BODY>
    <H1>Questo è un documento</H1>
    <P>Il body ha del testo.</P>
  </BODY>
</HTML>

```

Che potrebbe essere rappresentato in una struttura ad albero come la seguente:

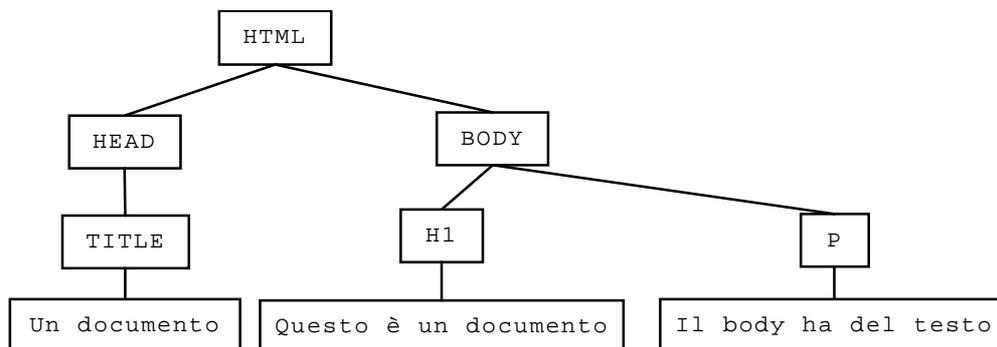


Figura 7 - Rappresentazione strutturale ad albero del codice HTML

A differenza di ciò che accade in HTML, in XML non vi sono elementi o attributi predefiniti e noti, per cui *il processo di parsing* rappresenta effettivamente il modo per costruire una rappresentazione strutturale tra i vari elementi del documento letto. Un esempio di codice che descrive un cliente, potrebbe essere il seguente:

```

<CLIENTE>
  <NOMINATIVO>
    <NOME>Mario</NOME>
    <COGNOME>Rossi</COGNOME>
  </NOMINATIVO>
  <INDIRIZZO>
    <STRADA>Via Roma, 12</STRADA>
    <CITTA>Alessandria</CITTA>
  </INDIRIZZO>
</CLIENTE>

```

Che può essere rappresentato nel seguente modo:

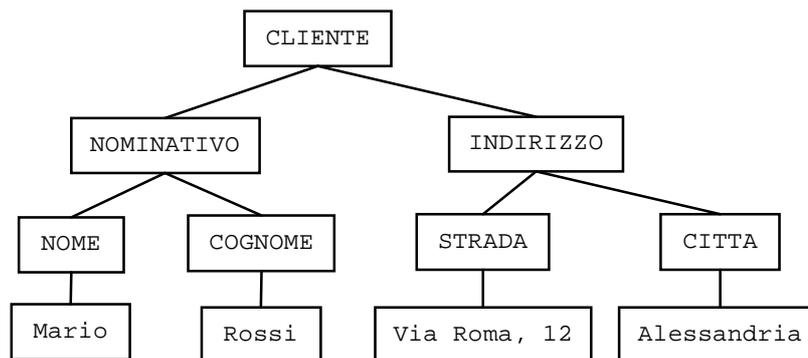


Figura 8 - Rappresentazione strutturale ad albero del codice XML

La radice **CLIENTE** è l’oggetto di livello più elevato che contiene un elemento **NOMINATIVO** che ha due figli **NOME** e **COGNOME** i cui valori sono “Mario” e “Rossi”. **CLIENTE** inoltre contiene un elemento di nome indirizzo che anch’esso ha due figli: **STRADA** e **CITTA** che hanno rispettivamente i valori di “Via Roma, 12” e “Alessandria”.

L’essenza di DOM riguarda le modalità di gestione di strutture come questa ed in particolare basa la sua elaborazione sull’iniziale azione di parsing dell’intero documento XML, restituendo un oggetto di tipo **Document** che rappresenta la struttura gerarchica dei marcatori e di altri elementi quali attributi, blocchi testuali, ecc.

Document è una interfaccia che rappresenta il documento XML e dispone di API per la manipolazione di un albero virtuale di oggetti di tipo **Node**. Quando il parsing è stato completato, è quindi possibile utilizzare metodi inclusi nelle API di **Document** per accedere al contenuto dell’albero XML. DOM consente di esaminare in qualsiasi momento il contenuto di ogni nodo dell’albero, manipolare l’albero, scrivere l’albero, passare l’albero ad altri moduli software che comprendano le interfacce DOM, ecc.. Il package DOM non consiste di classi, bensì contiene interfacce: DOM è proprio una specifica di interfacce tra componenti software.

La figura 9 mostra il grafo di ereditarietà relativo all’interfaccia DOM di livello 1. Com’è possibile osservare, qualsiasi cosa in una struttura di documento ad albero è di tipo **Node**. DOM definisce un documento come un albero di oggetti che implementa le interfacce del package DOM. Tutti

questi oggetti implementano **Node**, in quanto tutte le interfacce DOM sono sottointerfacce di **Node**, che rappresenta quindi in un certo senso l'interfaccia più importante. Per esempio **Element**, che rappresenta un elemento in un documento XML (o HTML), eredita il metodo **Node** come anche metodi aggiuntivi necessari per rappresentare un singolo tag nella struttura di documento.

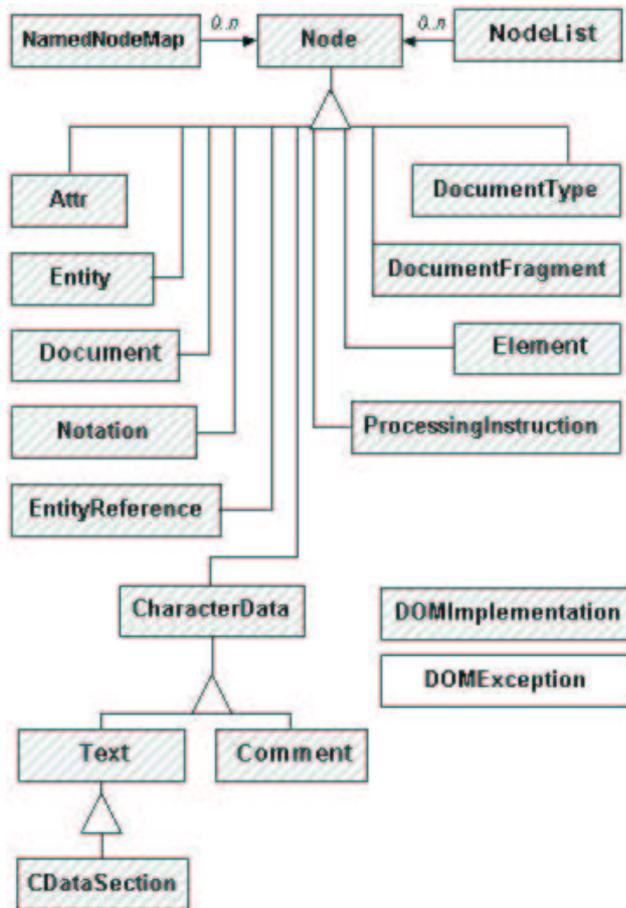


Figura 9 - Grafo di ereditarietà relativo all'interfaccia DOM di livello 1

- **Element**³⁹: questa interfaccia rappresenta il singolo tag nel documento XML ed eredita i metodi implementati dall'interfaccia

L'interfaccia **Node** identifica quindi il generale nodo in un albero DOM: per ogni nodo specifico l'interfaccia dispone di metodi per accedere ai nodi figli e genitori, alla radice del nodo che identifica il **Document**. **Elements**, **Comments**, **Text** e altri rappresentano tipi di **Node**, come mostrato in figura 9. Le principali sottointerfacce di **Node** che costituiscono l'albero del documento sono le seguenti:

³⁹ Si consideri il seguente esempio:

```

<elementoroot>
  <elemento1>
  </elemento1>
  <elemento2>
    <sottoelemento>
    </sottoelemento>
  </elemento2>
</elementoroot>

```

Node. Aggiunge ulteriori metodi per la manipolazione degli attributi di **Element** e per accedere a tutti i sotto-**Element** con un nome di marcatore specifico.

- **CharacterData:** rappresenta una sequenza di caratteri all'interno del documento e le sue sottointerfacce sono **Text**, **CDATASection** e **Comment**. Questa interfaccia dispone di metodi per aggiungere, cancellare, inserire, manipolare dati testuali nei nodi.
- **Text:** questa sotto-interfaccia estende **CharacterData** e rappresenta il testo contenuto all'interno di un elemento o di un attributo. Il testo all'interno di un nodo di tipo **Text** non contiene markup. Qualsiasi entità, commento o altro testo che dovesse contenere dei markup, verrebbe memorizzato in altri nodi: se il testo esaminato non contiene marcatori allora è compreso all'interno di un unico nodo **Text**, altrimenti ne viene eseguito il parsing e lo si inserisce in una lista di nodi.
- **CDATASection:** è anch'essa una sottointerfaccia di **Text** che può contenere markup che, inclusi all'interno di **CDATASection**, non vengono interpretati dal parser XML. Questo semplifica la creazione di testo nel documento che contiene diversi caratteri che potrebbero essere mal interpretati come markup. Essa inizia con il codice `<![CDATA[` e termina con `]]>`. Così, ad esempio la seguente **CDATASection:** `<![CDATA[Markup & Tesi]]>`, che include il testo `Markup & Tesi`, rappresenterebbe fuori dal contesto di **CDATASection:** `Markup & Tesi`.
- **Attr:** questi nodi contengono gli attributi dei tag. Sebbene ereditino i metodi dalla super-interfaccia **Node**, essi non esistono al di fuori del contesto di un particolare tag e non hanno una loro identità: è un caso particolare di nodo perché non viene considerato dal DOM come nodo figlio dell'elemento che

describe. Si ha quindi che i metodi che accedono ai nodi dell'albero (come `getChildNodes()`) ritornano `null`.

- **Comment:** estende `CharacterData` e rappresenta il contenuto di un commento contenuto fra i caratteri `<!--` e `-->`.
- **Entity:** rappresenta una entità in un documento XML. Ad esempio `&`; sopra riportato identifica una entità e per analogia risulta essere molto simile al costrutto `#define` utilizzato nei programmi C.
- **ProcessingInstruction:** le istruzioni di processamento compaiono in cima al documento XML. La principale istruzione di processamento in XML è la dichiarazione di file: `<xml version="1.0">`.
- **DocumentType:** il nodo di tipo di documento identifica un'interfaccia che contiene una lista di entità e notazioni definite nel documento.

Queste sono le interfacce principali ma vi sono anche altre interfacce nel package DOM. La figura 8, riportata nelle pagine precedenti, evidenzia un esempio di possibile albero DOM che riflette la struttura del rispettivo listato XML riportato.

XMLStudio utilizza il parser Crimson di Sun, per il quale esiste una classe fondamentale di nome `DocumentBuilderFactory`⁴⁰ che consente all'applicazione di ottenere da un documento XML, un oggetto organizzato secondo una struttura ad albero di tipo DOM. In poche righe di codice avviene la lettura del file XML, il controllo se è ben formato, la costruzione di questa complessa struttura gerarchica ad oggetti DOM i cui elementi principali sono come discusso di tipo `org.w3c.dom.Node`. La porzione di codice essenziale per svolgere questo lavoro è riportata nel listato seguente

⁴⁰ Inclusa nel package `javax.xml.parsers` che consiste in classi utilizzate per processare documenti XML. Il DOM è invece definito nel package `org.w3c.dom` ed è costituito da API che permettono l'accesso alle strutture e ai dati dei documenti XML. Esempi sono `Document`, `Element`, `Node`, `Attr`, ecc. discussi nelle pagine precedenti.

grazie al quale alla fine in `documentoXml` sarà presente un oggetto di tipo `Document` che rappresenta l'intero documento XML in struttura DOM. L'interfaccia `Document` fornisce inoltre i metodi factory per creare gli altri oggetti che compongono il documento come elementi e nodi di testo.

Una volta realizzata l'operazione di parsing è possibile attraversare l'albero che rappresenta il documento XML, aggiungere e cancellare nodi, aggiornare i loro valori, leggere o impostarne i rispettivi attributi: in sintesi effettuare qualsiasi operazione si desideri su questo albero.

```
try
{
    // DocumentBuilderFactory definisce API che consentono ad applicazioni di ottenere un parser che produce oggetti DOM da documenti XML
    DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance(); // crea una nuova istanza di DocumentBuilderFactory
    // DocumentBuilder e' usato per ottenere un'istanza di documento DOM da un documento XML
    DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder(); // crea una nuova istanza di DocumentBuilder
    // parsifica il contenuto di un dato file come un documento XML e restituisce un nuovo oggetto documento DOM
    documentoXml = docBuilder.parse(fileCorrente);
    JOptionPane.showInternalMessageDialog(getContentPane(), "Il documento " + fileCorrente.getName() + " e' ben formato: e' possibile creare l'albero di
    parsing.", "Informazione", JOptionPane.INFORMATION_MESSAGE);
    stato.setText("Il documento " + fileCorrente.getName() + " e' ben formato.");
    benformato = true;
} catch (SAXException e) {
    JOptionPane.showMessageDialog(this, "Informazione", "Il documento " + fileCorrente.getName() + " non e' xml ben
    formato." + e + " Si consiglia di apportare le modifiche per renderlo ben formato." + azioneTrovaCaratteriNonValid());
    stato.setText("Il documento " + fileCorrente.getName() + " non e' ben formato.");
}
}
```

Nella classe `XMLStudio` non vengono realizzate modifiche al documento XML in fase di elaborazione dell'area testo: DOM è necessario solo per verificare che il documento sia ben formato e per presentarlo secondo la rappresentazione strutturale ad albero⁴¹. Invece nella classe `xmlTag` vengono effettuate operazioni di inserimento, aggiornamento, cancellazione sui singoli nodi, attributi e valori della struttura ad albero che mostra i marcatori che su disco costituiscono il file XML dei tag. Per ulteriori dettagli implementativi, si rimanda al codice contenuto nei listati di XMLStudio⁴² e ai relativi commenti scritti nei vari file Java.

⁴¹ Entrambi gli alberi in XMLStudio utilizzano il componente Swing *JTree*.

⁴² Vedere la *Documentazione di XMLStudio*.

3.2.4 – Alcune caratteristiche del codice sorgente

I file java che costituiscono XMLStudio sono 18. Nelle pagine seguenti, per ognuno di essi, verrà riportata una breve descrizione:

- **AboutBox.java:** definisce la classe che permette di creare una finestra *about*, estensione di una *JDialog*, per mostrare alcune informazioni sul programma.
- **CopiaCorretta.java:** definisce una classe per la scrittura del file XML privato delle linee vuote⁴³. Viene utilizzata solo nella classe *XmlTag* e dopo aver scritto il file dei marcatori su disco, ne realizza una copia di backup nel file `./tag/TagXmlBackup.xml`.
- **CreaDialog.java:** definizione di una classe per la creazione di diverse finestre *Dialog* per l'iterazione con l'utente: spostamento ad una linea, sostituzione di parole, impostazione della dimensione della finestra dell'editor XMLStudio.
- **DTDGenerator.java:** definizione classe per la generazione del file DTD a partire dal documento XML elaborato nell'area testuale; il codice originale⁴⁴ di questa classe è di Michael H.Kay in versione 7.0, ripreso secondo il principio di riusabilità di codice esistente.
- **FiltroFile.java:** definizione della classe per impostare i filtri nella finestra per la scelta del file da aprire o da salvare.
- **Guida.java:** definizione classe per l'apertura di una finestra *JDialog* che mostra una breve guida testuale contenuta nel file `XMLStudio.guida` e che riporta alcune utili informazioni⁴⁵ per l'installazione del programma con *JavaHelp*.

⁴³ In pratica elimina le linee vuote inutili contenenti solo i caratteri `tab`, `\t`, `\n`.

⁴⁴ *DTDGenerator* è un programma Java che, dato in input un documento XML ben formato, produce una Document Type Definition (DTD) come output che viene salvata in un file con estensione `.dtd`. In XMLStudio il nome del file DTD associato al documento XML è lo stesso di quest'ultimo, cambia solo l'estensione.

⁴⁵ L'installazione di *JavaHelp* è requisito essenziale affinché il programma funzioni correttamente se si desidera, oltre che eseguirlo, anche compilarlo, modificarlo, ecc. *JavaHelp* supporta un motore di ricerca interno per trovare informazioni all'interno dei file che costituiscono la guida. Questo per funzionare utilizza un database che risiede nella directory *JavaHelpSearch* che contiene dati pre-generati basati sui file `.html` che compongono la guida. Conviene non installare mai XMLStudio in una directory che nel nome ha al suo interno spazi vuoti, altrimenti il motore di ricerca interno di *JavaHelp* non funzionerà e potrebbe causare dei problemi come crash o blocchi dell'applicazione.

- ***ImpostaFont.java***: definizione della classe per impostare, tramite una `JDialog`, il font usato nell'area testo di elaborazione del documento XML.
- ***JOptionPaneEsteso.java***: tramite questa classe viene definito un `JOptionPane` che consente di specificare e di mostrare a video all'utente dei messaggi d'errore più lunghi rispetto a quelli normalmente definibili nei normali `JOptionPane`.
- ***PopupListener.java***: questa classe definisce del codice che consente di rilevare quando il mouse è stato premuto o rilasciato.
- ***RenderAlbero.java***: Quando un albero analizza e mostra ogni nodo, nè il `JTree` e neppure il suo look-and-feel contengono informazioni relative alla *visualizzazione del nodo*. Invece l' albero utilizza il codice incluso in questa classe di rendering per mostrare ogni nodo. Per esempio, per mostrare una foglia dell' albero che ha un determinato valore stringa, l' albero interroga il `Cell Renderer` in modo che restituisca un componente che possa visualizzare un nodo foglia con quello specifico valore di stringa. Un `Cell Renderer` consente soltanto la visualizzazione, non gestisce eventi.
- ***StampaGrafica.java***: definizione di una classe per effettuare la stampa del documento grazie a `PrintJob`, utilizzando la modalità grafica.
- ***StoriaFile.java***: definizione della classe per memorizzare i cammini degli ultimi file aperti. A livello grafico questi compaiono nel menù *File* di XMLStudio.
- ***TavolaAttributi.java***: questa classe astratta fornisce un' implementazione di default per la maggior parte dei metodi dell' interfaccia `TableModel` in `javax.swing.table`. Si occupa di gestire i listener e fornisce alcuni strumenti per generare `TableModelEvents` e inviarli agli opportuni listeners.
- ***UndoableTextArea.java***: per realizzare le operazioni di annullamento e ripristino dell'ultima operazione effettuata sull'area testo di

elaborazione del file XML è necessario implementare la `JTextArea` con `UndoableEditListener`.

- ***XmlNodo.java***: questa classe estende `DefaultMutableTreeNode` per rappresentare i nodi XML nella rappresentazione che mostra l' albero del documento XML parsificato: sia il documento XML elaborato nell'area testo, sia il file XML relativo ai marcatori.
- ***XMLRoutine.java***: classe per la scrittura del file XML.
- ***XMLStudio.java***: rappresenta la classe principale del programma che estende un `JFrame` e carica tutte le impostazioni di default utilizzando la classe `java.util.Properties` ed il file `XMLStudio.default` contenuto nella directory `Risorse`. Crea i menù, le icone, i vari oggetti dell'interfaccia grafica di XMLStudio utilizzando le altre classi e li pone nella finestra del programma. Si veda il manuale utente a run-time `JavaHelp` per un'analisi delle funzionalità che XMLStudio implementa come applicazione.
- ***XmlTag.java***: questa classe consente di definire l' oggetto costituito da un albero `JTree` che permette l' inserimento, la modifica, la cancellazione dei tag all' interno dei documenti testuali e da una `JTable` relativa ai rispettivi attributi e valori dei marcatori. All'avvio dell'applicazione viene precaricato il file XML relativo ai tag di nome `D:\XMLStudio\Tag\XMLStudio.xml`. Questo può essere salvato modificandone eventualmente il nome⁴⁶.

3.3 – *JavaHelp come manuale utente*

`JavaHelp`⁴⁷ è un insieme di API e allo stesso tempo un help system scritto interamente in Java, creato appositamente per questo linguaggio. Quest'ultima caratteristica di `JavaHelp` lo rende particolarmente interessante: l'essere stato completamente progettato e realizzato secondo la caratteristica

⁴⁶ Il file XML dei marcatori precaricato all'avvio di XMLStudio può essere impostato dal menù *File tag di default (.xml)*.

⁴⁷ Le classi che consentono l'uso di `JavaHelp` sono incluse nel package `javax.help.*`. Nell'applicazione XMLStudio vengono distribuiti due file di nomi `jhall.jar` e `jsearch.jar` che consentono il corretto funzionamento dell'Help.

che fa di Java un linguaggio portabile, gli permette di ereditarne questa proprietà: è infatti possibile pensare di realizzare help e distribuirli tramite il World Wide Web, disinteressandosi completamente della piattaforma di destinazione, su qualsiasi ambiente che disponga di una JVM (Java Virtual Machine) o del JRE (Java Runtime Environment).

Il sistema JavaHelp permette la realizzazione di un help in linea con la capacità di navigare, cercare e visualizzare le informazioni, rendendo più semplice, per l'utente finale, la comprensione dell'applicazione e delle sue funzionalità. I componenti della GUI (Graphic User Interface) di JavaHelp seguono le metodologie di progettazione Swing. Questo sistema di help utilizza un'intelligente combinazione di HTML 3.2 e XML per rendere semplice ed immediato l'accesso alle informazioni d'aiuto, per creare una TOC (Table Of Contents), indici e modalità di ricerca di un argomento attraverso l'inserimento di parole chiave.

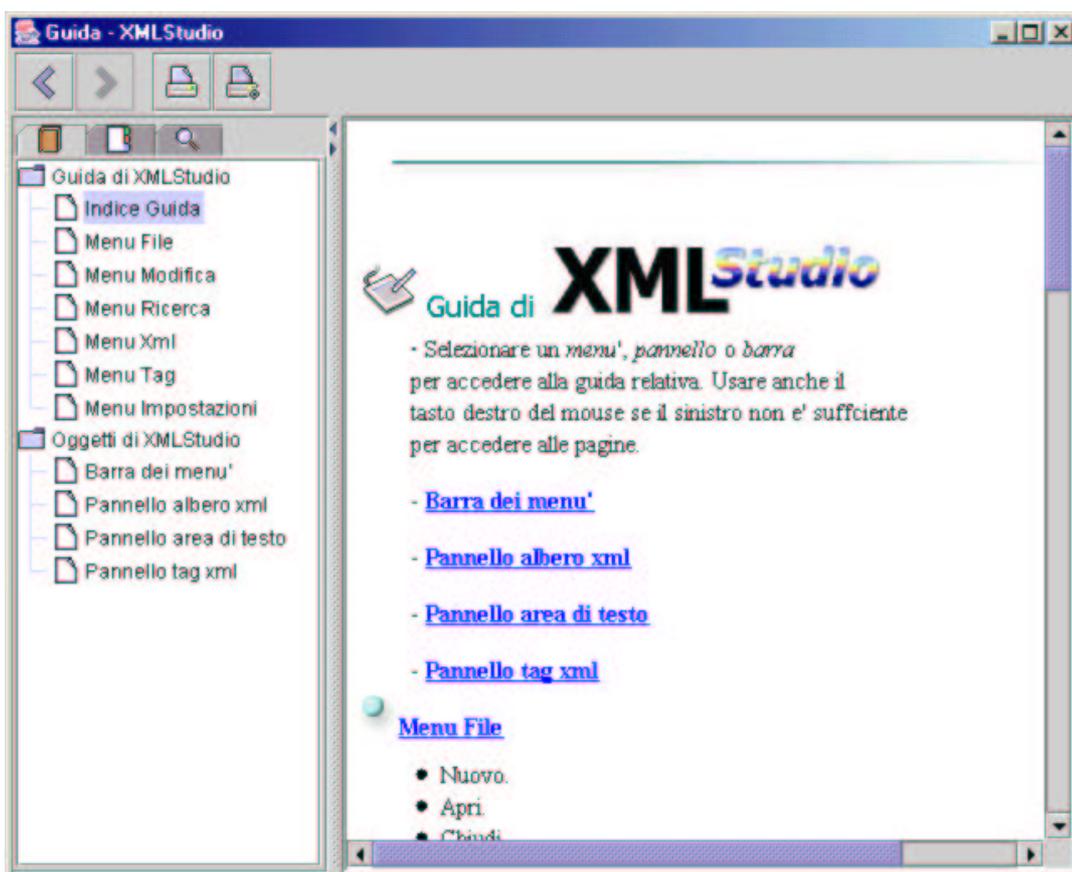


Figura 10 - La finestra di JavaHelp utilizzata da XMLStudio per fornire l'aiuto in linea

3.3.1 – Le caratteristiche di JavaHelp

JavaHelp è composto da una finestra che consiste in una *ToolBar*, in un *ContentPane* ed in un *NavigationPane*. Il *ContentPane* utilizza HTML 3.2 come formato per la visualizzazione delle informazioni. Il *NavigationPane* consiste di un'interfaccia grafica che permette all'utente di spostarsi dal "Contenuto della guida", a quello che è definito "L'indice della guida", al "Motore di ricerca interno".

In XMLStudio è anche possibile richiamare JavaHelp secondo una modalità *sensibile al contesto*. Posizionando il mouse sull'icona  riportata sulla barra dei menù, il cursore assume una nuova forma a "freccia e punto di domanda": se posizionato su di un componente di XMLStudio produce l'apertura automatica di JavaHelp che fornisce una spiegazione specifica delle funzionalità implementate da quell'oggetto.

3.4 – L'uso pratico di XMLStudio

Una volta lanciato XMLStudio è necessario caricare un file testuale per la sua trasformazione in XML e successiva elaborazione. Dopo averlo caricato tramite il comando *File/Apri*, viene richiesto se creare automaticamente il documento XML e se si risponde affermativamente viene presentata la finestra per salvare il file in formato XML. Vengono aggiunte in cima e in coda al documento le seguenti righe di codice:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<documento>
...
</documento>
```

ISO 8859, valore dell'attributo `encoding` è una serie di oltre 10 insiemi di codifica di caratteri a singolo byte (8bit). Include ad esempio [Latin1](#) (West European), [Latin2](#) (East European), [Latin3](#) (South European),

[Latin4](#) (North European), [Cyrillic](#), [Arabic](#), [Greek](#), [Hebrew](#), [Latin5](#) (Turkish), [Latin6](#) (Nordic). Questo insieme di caratteri non è così completo come [Unicode](#) ma offre alcuni miglioramenti rispetto al classico 7bit [US-ASCII](#).

A questo punto il documento ha estensione XML ma è necessario verificare che sia ben formato premendo l'icona  prima di poter creare l'albero di parsing. Una finestra avvisa l'utente nel caso il documento in esame non sia ben formato e suggerisce alcuni caratteri da sostituire. All'interno dei documenti XML, come discusso nel primo capitolo, è possibile includere qualsiasi carattere dell'insieme US-ASCII⁴⁸ a 7 bit ad eccezione dei caratteri speciali che devono essere sostituiti con la rispettiva sequenza di escape. Da prove effettuate, anche se i caratteri relativi ad entità predefinite che renderebbero non ben formato il documento XML sono cinque (&, <, >, “, ‘), in realtà solo i simboli & e <, sono da sostituire per rendere il documento ben formato.

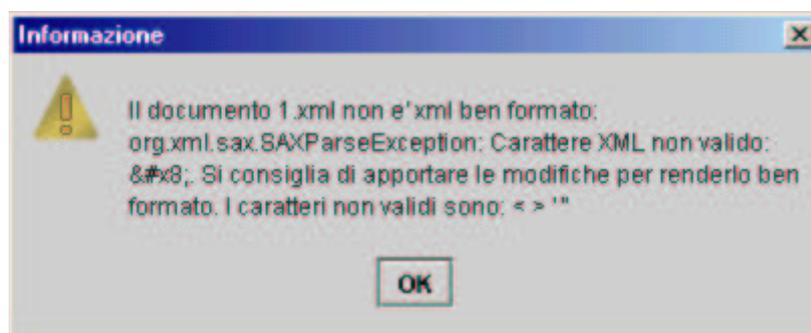


Figura 11 - Finestra che evidenzia che il documento XML non è ben formato

Nel caso in cui il documento risultasse essere non ben formato, per renderlo tale, sarebbe opportuno sostituire secondo una modalità *manuale* o *automatica* principalmente i caratteri⁴⁹ & e < dall'intero documento. Il

⁴⁸ Gli insiemi di caratteri UTF-8 e ISO8859-1 (Latin Alphabet #1) sono essenzialmente identici con US-ASCII sino alla posizione 127. Per i caratteri successivi, quelli accentati, UTF-8 utilizza sequenze di escape multi-byte.

⁴⁹ Per XML il carattere < segna sempre l'inizio di un tag ed il carattere & segna sempre l'inizio di un riferimento a entità. Questo comporta problemi quando è necessario inserire del testo contenente < e &. XML ben formato richiede che, quando non siano utilizzati come parte di un tag o di una entità, < e & siano scritti come *<* ed *&*; compresi i ; finali.

comando per effettuare questa modifica si trova nel menù *Xml/Modifica per documento ben formato/Sostituisci &* (oppure <). La modifica manuale consente all'utente il pieno controllo su ogni sostituzione mentre l'automatica realizza le sostituzioni senza l'intervento umano.

Come mostrato in figura 12, dopo queste sostituzioni il documento sarà stato reso ben formato.

A questo punto risulta possibile leggere il documento e marcare i paragrafi di interesse con i tag scelti. E' sempre possibile aggiungere, modificare, cancellare i tag come anche i rispettivi attributi e valori, salvando se lo si ritiene necessario il file XML dei marcatori.

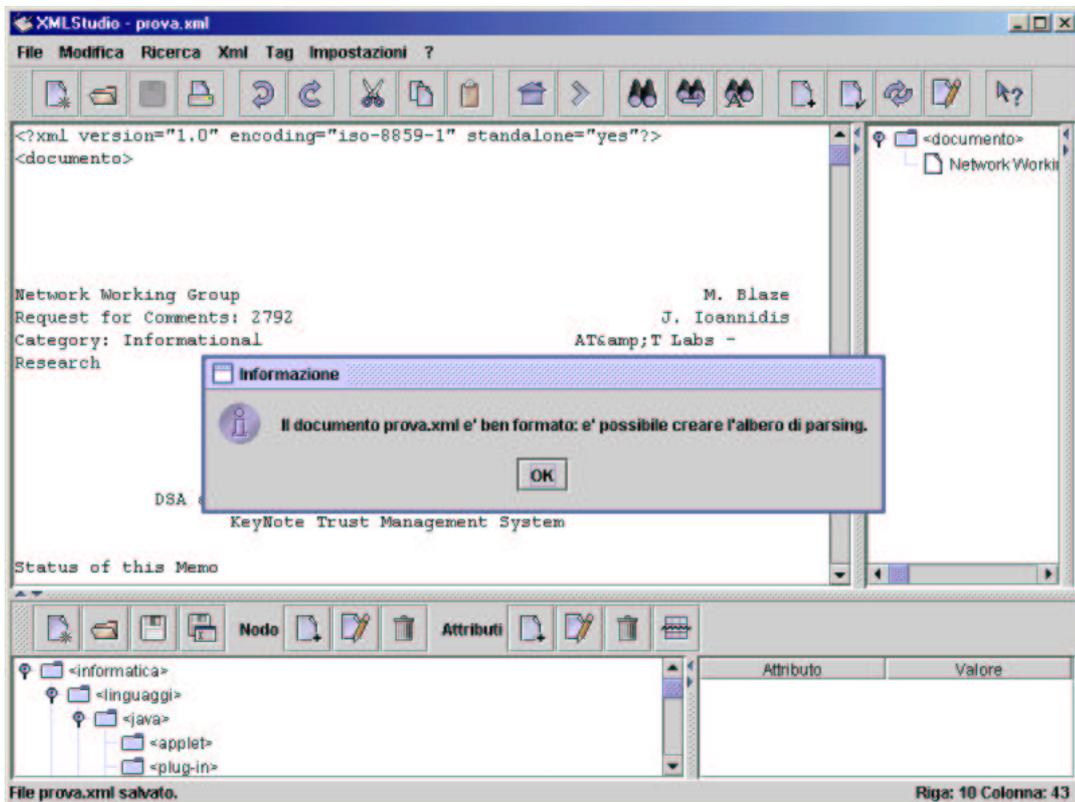


Figura 12 - L'editor XMLStudio segnala che il documento XML è ben formato

Per marcare una porzione di testo è sufficiente evidenziarla, scegliere un marcatore ed uno o più attributi (e valori) e premere l'apposita icona



. Sarà comunque sempre possibile togliere il tag inserito o manualmente oppure premendo il pulsante destro del mouse sul documento

XML per far comparire un menù ad accesso rapido che riporta alcune funzioni, come mostrato nella figura 13.

Premendo il tasto  verrà nuovamente parsificato il documento e aggiornato l'albero che mostra la suddivisione strutturale dello stesso secondo i tag definiti dall'utente.

E' sempre possibile agire sui controlli posti tra i pannelli (JSplitPane) per modificare l'aspetto dell'editor al fine di visualizzare meglio ciò che si sta elaborando, come mostrato nelle figure 14 e 15.

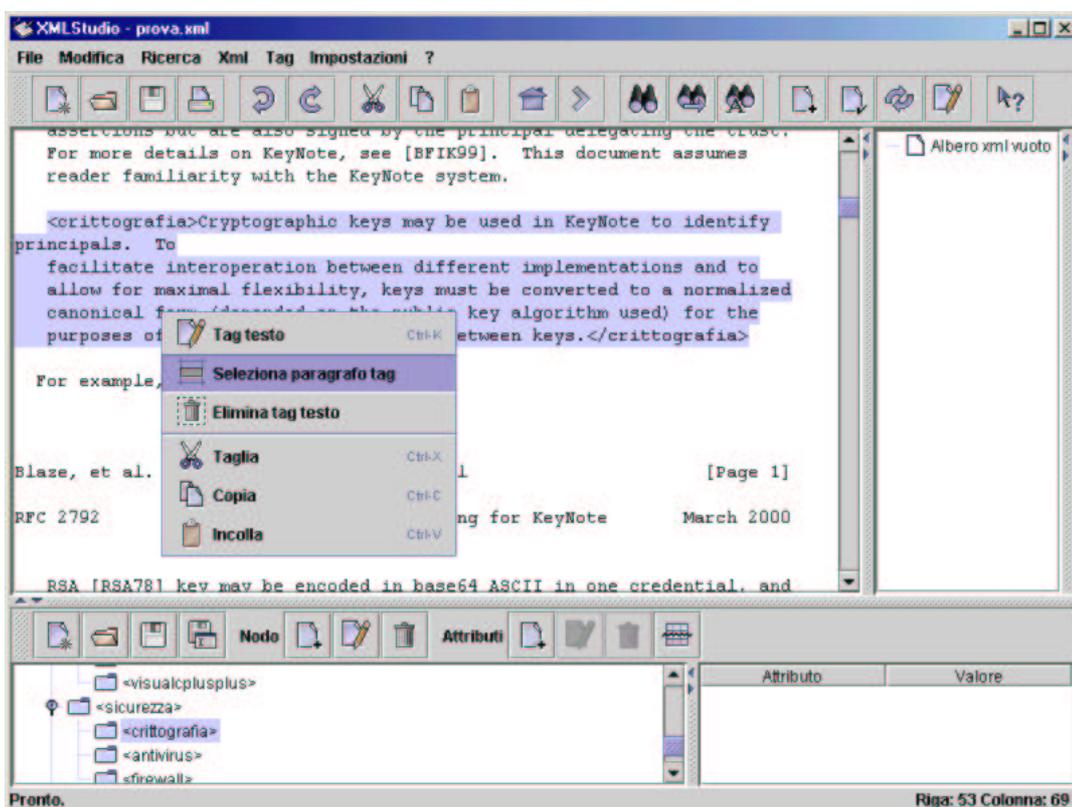


Figura 13 - Premendo il pulsante destro del mouse compare un menù

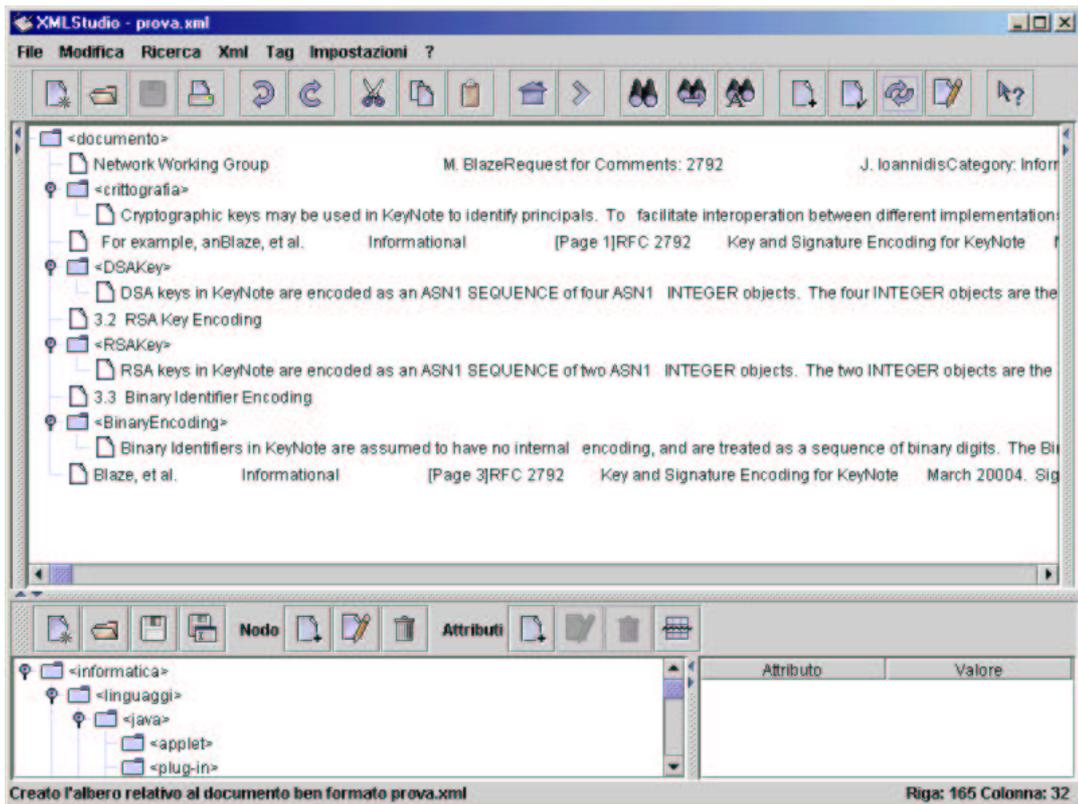


Figura 14 - Albero che mostra la suddivisione strutturale del documento XML

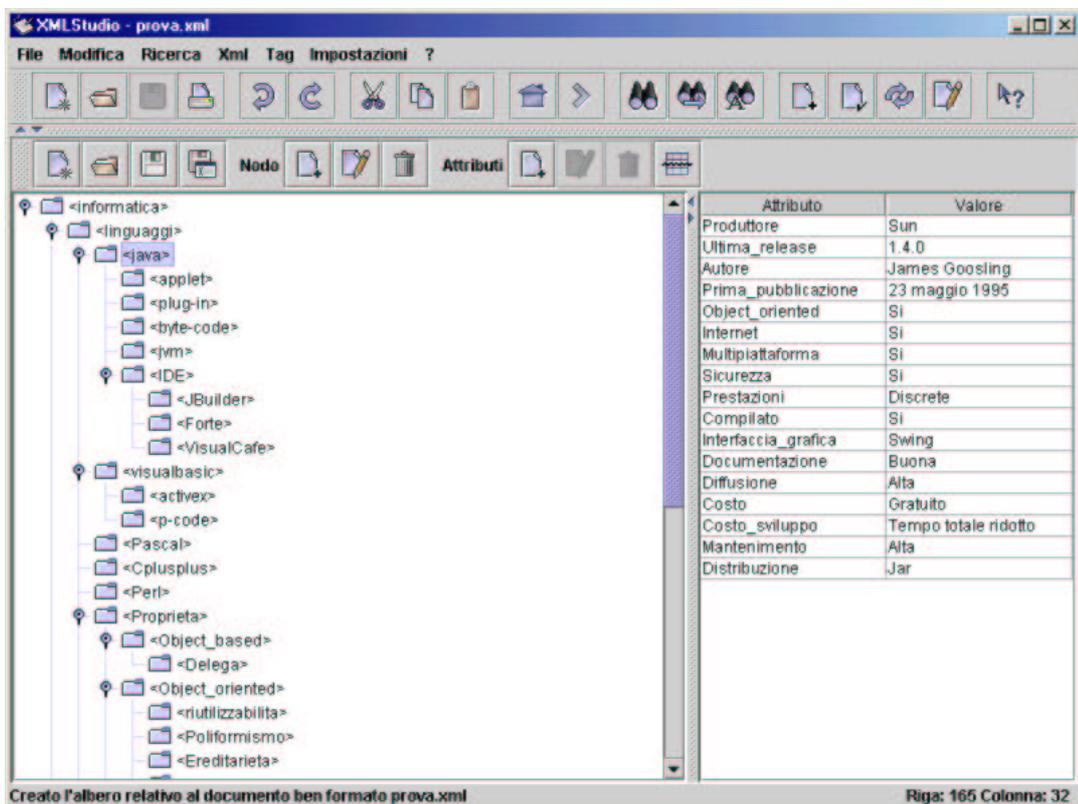


Figura 15 - Vista per modificare i marcatori da utilizzare sul documento

Giunti a questo punto il documento XML elaborato risulta essere *ben formato*, ma non ancora *valido*. Per renderlo tale è necessario creare il DTD associato *prova.dtd* grazie al menù *Xml/Crea dtd e documento xml valido*. Questo DTD viene incluso all'interno del file XML attraverso una modifica alle linee di codice d'intestazione del file:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE documento SYSTEM "./prova.dtd">
<documento>
...
```

3.4.1 – Creazione di uno specifico foglio di stile CSS associato

XML definisce la struttura e la semantica del documento ma non il formato: se si desidera visualizzare XML si può utilizzare un *foglio di stile* per definire le modalità con le quali deve essere presentato.

I file CSS sono fogli di stile per i file XML e contengono codice che descrive lo stile da usare per rappresentare il documento, come ad esempio i margini, i tipi di carattere, i colori, particolari visualizzazioni di una informazione contenuta tra certi tag, ecc. Sono file testuali composti da istruzioni che prendono il nome di regole che identificano, nel caso di XMLStudio, un elemento del documento XML e determinano come presentare questo elemento. Ogni regola è composta da due parti: un *selettore* e una *dichiarazione* come nel seguente esempio *java {color:green}*. In questo caso *java* rappresenta il selettore e specifica a quale tipo di elemento del documento deve essere applicata la dichiarazione. Mentre la dichiarazione è rappresentata da *color: green* il cui senso è indicare che nel documento XML i paragrafi delimitati dai marcatori di nome *java* devono essere di colore verde. Ogni dichiarazione è infatti composta da due parti, una proprietà (ad esempio *color*) e un valore da essa assunto (ad esempio *green*). La proprietà è una caratteristica che il selettore possiede mentre il valore rappresenta un'indicazione precisa di come, relativamente alla proprietà indicata, deve essere realizzata la presentazione.

Nei file CSS realizzati da XMLStudio avviene un raggruppamento di più dichiarazioni per ogni selettore nel modo seguente:

```

documento { color : #000000; font-family : Arial Black; }
crittografia { color : #3399ff; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
DSAKey { color : #ff0033; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
RSAKey { color : #009999; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
BinaryEncoding { color : #993300; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }

```

L'utente, lanciando il comando *Xml/Crea css associato al file xml*, può associare ad ogni marcatore definito nel documento che desidera, uno specifico colore di sua scelta attraverso una finestra *JColorChooser* (vedere figura 16). Nell'esempio riportato, viene scelto un colore differente per i selettori *crittografia*, *DSAKey*, *RSAKey*, *Binary Encoding* ed al termine viene generato un file di nome `prova.css`: il foglio di stile è quindi inserito in un file di testo separato con estensione `.css`.

Per fare in modo che il foglio di stile influenzi la presentazione del documento XML viene collegato il file `prova.css` utilizzando l'elemento `link` relativo ad un foglio di stile esterno: il collegamento viene effettuato all'interno dell'intestazione del documento XML:

```

<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE documento SYSTEM "./prova.dtd">
<?xml-stylesheet type="text/css" href="prova.css"?>
<documento>
...

```

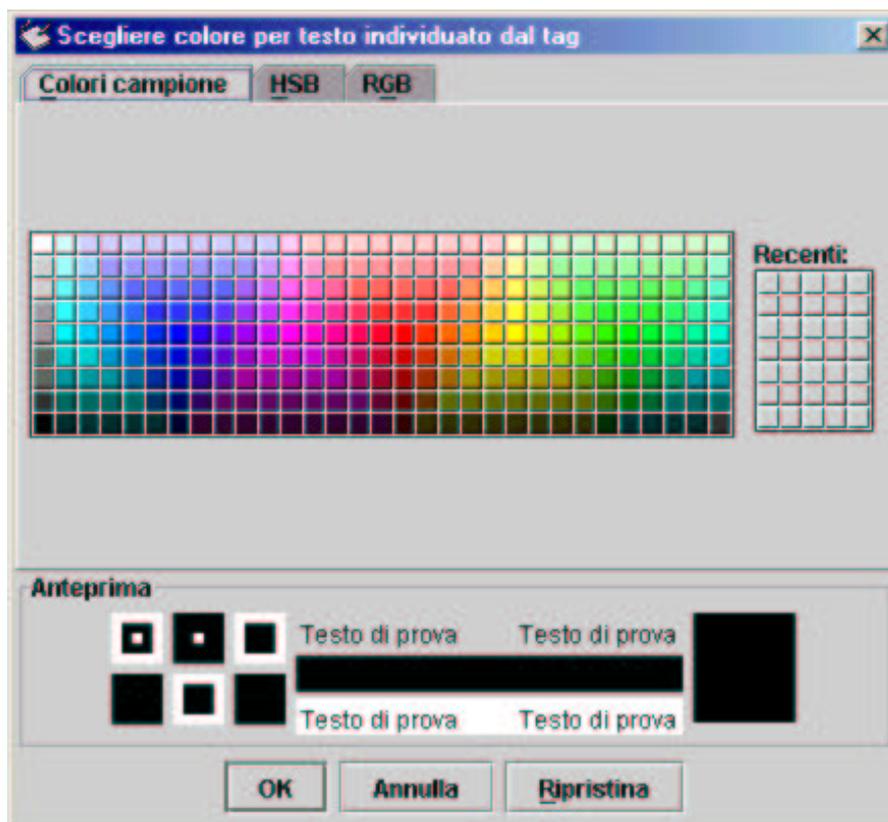


Figura 16 - Finestra JColorChooser per la scelta dei colori associati ai marcatori

Giunti a questo punto, creato il file XML, il DTD e CSS associati, sarà possibile mostrare il risultato tramite il browser, come illustrato nella seguente figura 17.

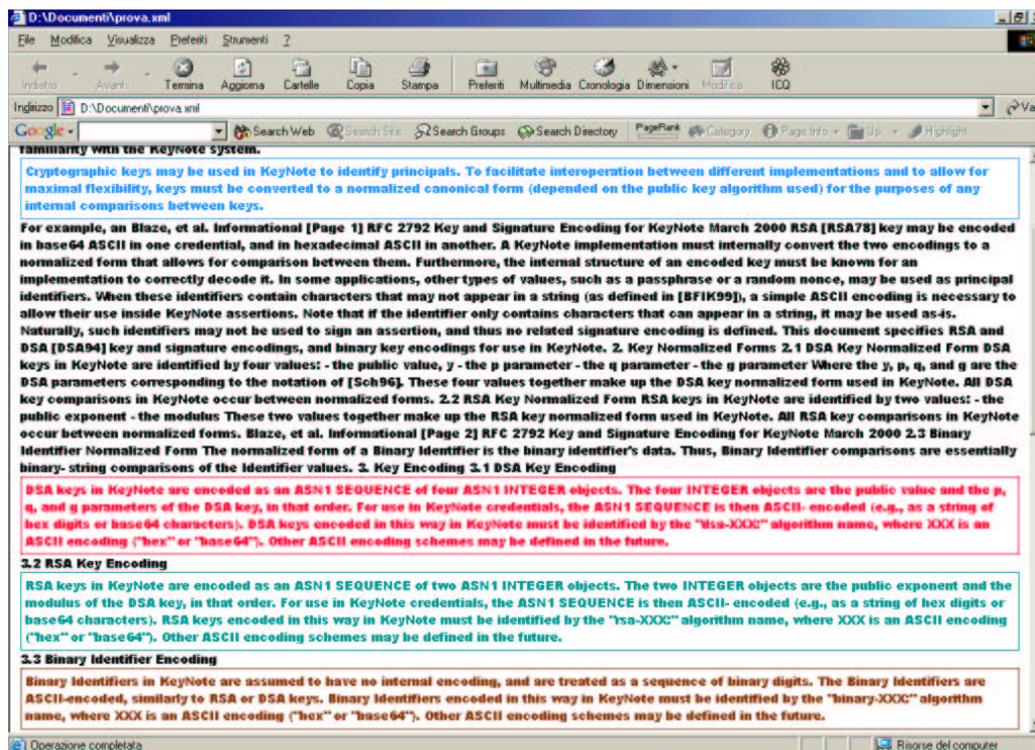


Figura 17 - Visualizzazione nel browser del file XML valido
con un foglio di stile CSS associato

3.5 – Esempi d'uso di XMLStudio

Oltre al testing dell'applicazione realizzato dal dott. Fabrizio Tambussa, ho utilizzato il programma su 22 documenti testuali RFC per verificarne il corretto funzionamento. Allegati al programma⁵⁰ vi sono inoltre quattro esempi d'uso di XMLStudio⁵¹: partendo da articoli in formato testuale, presi casualmente in internet, di genere informatico, medico, storico e uno specifico RFC, grazie ad XMLStudio sono stati trasformati in file XML, sono stati inseriti i marcatori per definirne la struttura e sono stati creati i DTD e CSS associati.

3.6 – Organizzazione dei file e directory

Il programma viene lanciato con il comando:

```
java -jar XMLStudio.jar
```

⁵⁰ Gli esempi sono riportati nell'Appendice B della tesi. Inoltre nella directory *esempi* della distribuzione di XMLStudio, sono inclusi i file XML, DTD, CSS; i rispettivi file dei marcatori sono invece inclusi nella directory *tag*.

⁵¹ Una sintetica descrizione è riportata nell'appendice.

Nella distribuzione del programma vi sono i file:

- **XMLStudio.jar** (contiene al suo interno tutte le classi dell'applicazione).
- **jsearch.jar** (indispensabile per il corretto funzionamento dell'applicazione e di JavaHelp).
- **jhall.jar** (indispensabile per il corretto funzionamento dell'applicazione e di JavaHelp).

e 3 directory:

- **Help** (contenente tutti i file necessari all' help).
- **Risorse** (contenente le icone ed i file di configurazione del programma, scritti e letti in fase di esecuzione).
- **Tag** (contenente il file XMLStudio.xml dei tag XML di default che viene precaricato all' avvio e la copia di backup del file corrente dei tag; se cancellata questa directory fa sì che il programma semplicemente non carichi il file XML dei marcatori di default e lo segnali; la directory contiene anche altri file XML dei marcatori di esempio).
- **Esempi** (contenente file di esempio con estensione TXT, XML, DTD, CSS. Questa directory non contiene file indispensabili al corretto funzionamento dell'applicazione, ma solo di esempio).

Il file **Leggimi.txt**, incluso nella directory principale del programma, contiene informazioni aggiuntive relative all'installazione, compilazione, esecuzione di XMLStudio in ambiente Unix e Windows.

3.7 – Diagramma delle classi di XMLStudio

Per comprendere meglio la struttura logica dell'applicazione, il suo contenuto e le relazioni tra i vari elementi, si è creato un *diagramma delle classi UML* utilizzando il programma JVision 1.2.1 e 2.1. Questo diagramma mostra la struttura statica di XMLStudio costituita dai collegamenti esistenti tra le varie classi per via delle relazioni di ereditarietà o dei contenuti delle classi.

Si è volutamente preferito non includere nel rettangolo descrittivo delle singole classi l'elenco degli attributi e delle operazioni perché, visto

l'elevato numero degli stessi, si rimanda alla *Documentazione di XMLStudio* per visionare l'elenco completo riportato nella documentazione prodotta con Javadoc e nei listati.

Il diagramma presentato suddivide le varie classi in tre colori: con il colore *verde* viene evidenziata la classe principale del programma ovvero XMLStudio; con il colore *giallo* vengono mostrate tutte le altre classi dell'applicazione; con il colore *blu* vengono indicate alcune classi di default di Java.

Inoltre una seconda suddivisione delle classi è espressa grazie all'uso di due rettangoli tratteggiati:

1. ***Classi relative alla parte computazionale***: si occupano di gestire varie operazioni di elaborazione e di input/output di XMLStudio.
2. ***Classi relative all'interfaccia grafica***: riguardano l'uso delle librerie Swing e l'implementazione di metodi per la creazione di finestre e oggetti per l'interazione con l'utente.

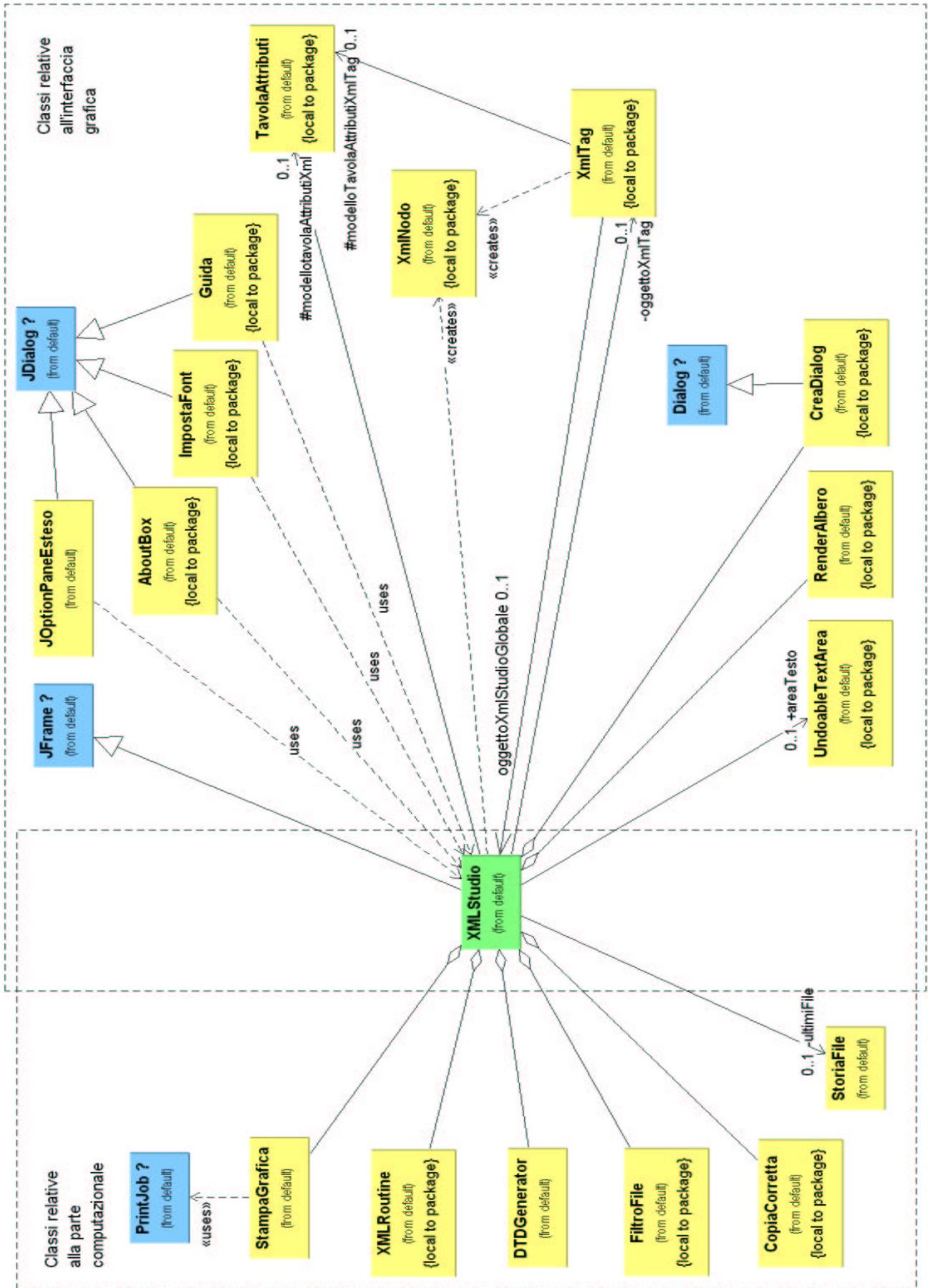
Il diagramma in figura mostra diverse relazioni tra le classi:

- ***Generalizzazione (o ereditarietà)***: è una relazione tassonomica fra un elemento più generale (il genitore) ed uno più specifico (il figlio) che eredita dal primo metodi e proprietà, aggiungendovi informazioni addizionali. Nel diagramma sono state riportate le relazioni di generalizzazione della classe XMLStudio che estende la classe JFrame di Java, CreaDialog che estende la generica classe Dialog e JOptionPaneEsteso, AboutBox, ImpostaFont, Guida che estendono la classe JDialog.
- ***Aggregazione***: sono indicate da un rombo vuoto posto all'estremo dell'associazione, verso la classe alla quale le altre classi sono aggregate. Questa relazione evidenzia in certo qual modo che la classe aggregata, ovvero quella con "il diamante" che la tocca, rappresenta l'insieme principale e l'altra classe nella relazione fa parte di quell'insieme. In figura diverse classi tra cui StampaGrafica, XMLRoutine, DTDGenerator, FiltroFile, CopiaCorretta,

RenderAlbero, CreaDialog hanno una relazione di aggregazione verso la classe principale XMLStudio.

- **Dipendenza:** indica una relazione semantica più debole tra due o più classi del modello che sono messe in relazione tra di loro ed è rappresentata da una freccia tratteggiata con l'elemento puntato dalla freccia che indica quello dal quale dipendono gli altri. In Booch94 è anche stata definita *relazione "using"* e difatti nel diagramma le classi JOptionPaneEsteso, AboutBox, ImpostaFont, Guida rappresentano delle finestre JDialog che svolgono un ruolo di interazione con l'utente ma dipendono dalla classe principale XMLStudio che identifica la finestra dell'applicazione di tipo JFrame. Un'altra relazione di dipendenza è evidenziata in modo automatico da JVision, che mostra che entrambe le classi principali del programma, XMLStudio e XmlTag, utilizzano la classe XmlNodo come estensione di DefaultMutableTreeNode per rappresentare ed utilizzare i nodi XML. Anche la classe StampaGrafica realizza un'associazione di dipendenza utilizzando la classe PrintJob per la stampa del testo.
- **Associazioni:** le associazioni di tipo generico sono evidenziate come linee che connettono le classi coinvolte ed una freccia viene riportata su di uno degli estremi dell'associazione per indicare che la navigazione è supportata nella direzione indicata. Nel diagramma viene mostrato che XMLStudio utilizza ad esempio le classi StoriaFile ed UndoableTextArea creando uno specifico oggetto per ognuna di esse (molteplicità 1) nella singola istanza del programma in esecuzione. Inoltre XMLStudio crea un oggetto di tipo XmlTag e nel crearlo passa se stesso (this) questo per far sì che la classe XmlTag possa a sua volta utilizzare, direttamente al suo interno, dei metodi della classe XMLStudio attraverso l'uso di un oggetto globale di nome oggettoXmlStudioGlobale definito nella classe XmlTag. Altre due relazioni di associazione sono infine definite tra le classi XMLStudio, XmlTag e TavolaAttributi, classe che fornisce un'implementazione di default per la maggior parte dei metodi

dell' interfaccia TableModel per l'uso delle tabelle JTable degli attributi e valori XML.



CAPITOLO 4

Confronto con altri strumenti e progetti

4.1 – Gli editor XML sul mercato

L'adozione di un determinato linguaggio è proporzionale alla disponibilità di strumenti che ne semplificano l'utilizzo. Le specifiche di XML sono caratterizzate da una notevole semplicità e ciò ha permesso la rapida proliferazione di programmi che consentono di elaborare file XML: alcuni di essi sono gratuiti, altri a pagamento e questi ultimi implementano generalmente più funzionalità. Nelle pagine seguenti verranno analizzate alcune delle caratteristiche di sette editor gratuiti per mostrare quali sono le principali finalità di applicazioni attualmente disponibili per l'elaborazione di file XML.

4.1.1 – Microsoft XML Notepad

E' un programma gratuito di modeste dimensioni: Microsoft probabilmente ne ha abbandonato lo sviluppo, mantenendo freeware questa versione e concentrando i propri sforzi legati ad XML nella realizzazione del nuovo ambiente di progettazione programmi di nome *Visual Studio .Net*. Le funzionalità che questa applicazione mette a disposizione sono essenziali e consentono il semplice editing dei file XML.

L'interfaccia grafica prevede due pannelli sincronizzati: il primo mostra *la struttura gerarchica del documento*, il secondo visualizza parallelamente a ciascun elemento *i dati* in esso contenuti. Una grave limitazione del software è data dall'impossibilità di modificare direttamente il codice: per apportare cambiamenti al documento XML è necessario agire esclusivamente sulla struttura ad albero. Caricando documenti non ben formati, benché venga segnalato e visualizzato il tipo di errore e la relativa posizione, per correggerli è necessario modificarli manualmente con un editor esterno. Risulta invece particolarmente comoda la possibilità di variare le tipologie dei singoli nodi e di duplicare automaticamente sottorami, replicandone sia i valori, sia la struttura. Permette l'apertura di un solo file XML per volta, il codice sorgente non viene visualizzato colorato e come accennato, non può essere modificato direttamente. Dispone di un help che ne illustra le principali funzionalità.

» Sito Internet del programma: <http://msdn.microsoft.com/xml/notepad/intra.asp>

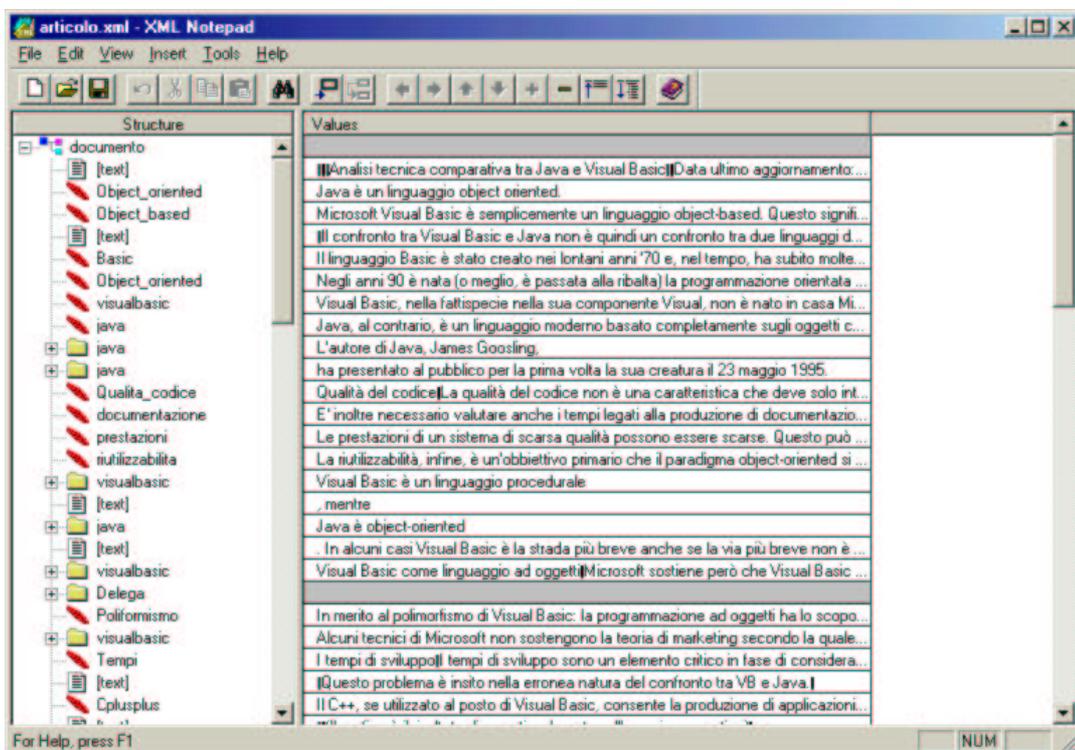


Figura 18 - Microsoft XML Notepad

4.1.2 – Peter's XML Editor

Anche questo programma, realizzato da Peter Reynolds in Delphi, è gratuito e nell'aspetto è sostanzialmente simile a Microsoft XML Notepad ma implementa alcune funzionalità che mancano al programma Microsoft. Graficamente è organizzato in due pannelli: il primo mostra le directory e i file presenti su disco per un rapido caricamento di quelli con estensione XML, mentre il secondo riporta tre viste del documento XML aperto. La prima è il *codice sorgente* i cui marcatori vengono mostrati colorati e consente di elaborare un documento XML anche non ben formato per correggerne gli errori; la seconda è *la vista strutturale ad albero* che viene mostrata solo nel caso in cui il documento sia ben formato; la terza è *l'output risultante in Internet Explorer*, anch'essa visualizzata solo nel caso in cui il documento analizzato risulti essere ben formato. Consente di lavorare su più file XML contemporaneamente e implementa le usuali funzionalità per la modifica dei nodi dell'albero e delle loro proprietà. Dispone di un help che ne illustra le principali funzionalità.

» Sito Internet del programma: <http://www.iol.ie/~pxe>

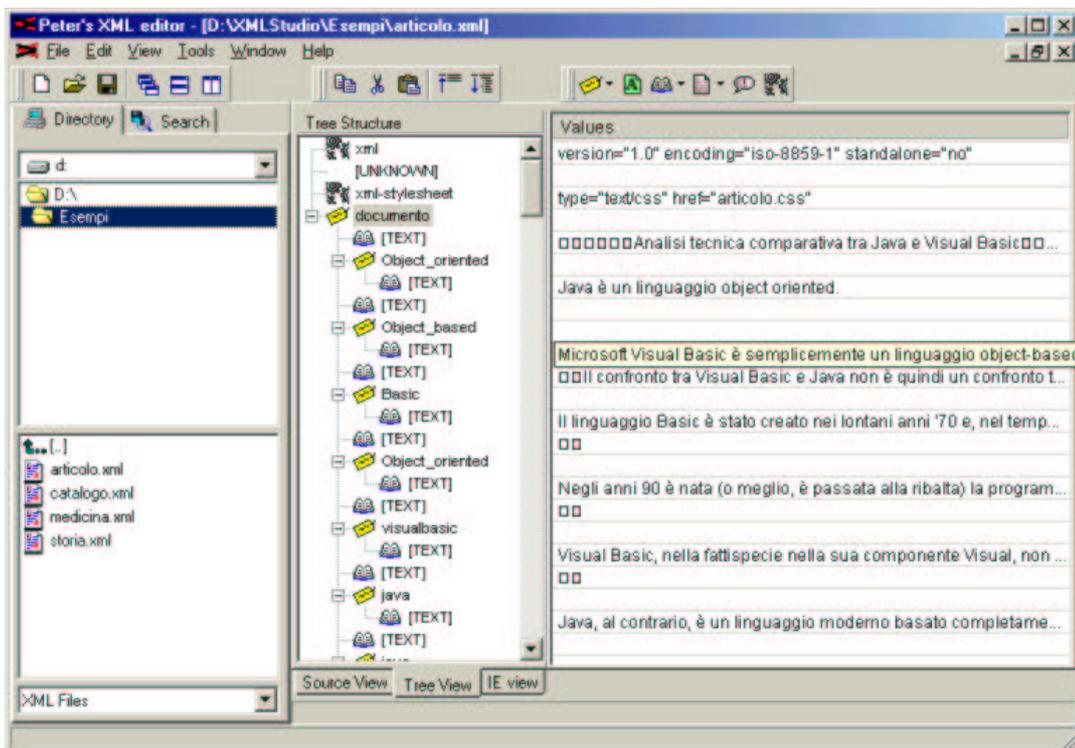


Figura 19 - Peter's XML Editor

4.1.3 – XMLEditPro

E' un programma gratuito semplice ed intuitivo sviluppato in C++ da David Levinson. Attualmente disponibile in versione 1.2 ma in questi giorni (Maggio 2002) dovrebbe uscire la nuova 2.0. La versione 1.2 è estremamente simile al Peter's XML Editor: l'interfaccia grafica è organizzata nello stesso modo, con tre pannelli, il primo che *elenca le directory*, il secondo *la struttura ad albero del documento*, il terzo che mostra *tre viste* ovvero *i nodi*, *il codice sorgente* visualizzato senza colori e *la visualizzazione finale in Internet Explorer*. Non supporta i DTD ma consente di editare i file anche via *ftp*. Supporta Microsoft XML parser 3.0 e successivi.

Consente di lavorare su più file XML contemporaneamente. Non dispone di un help che ne illustra le funzionalità.

» Sita Internet del programma: <http://www.daveswebsite.com>

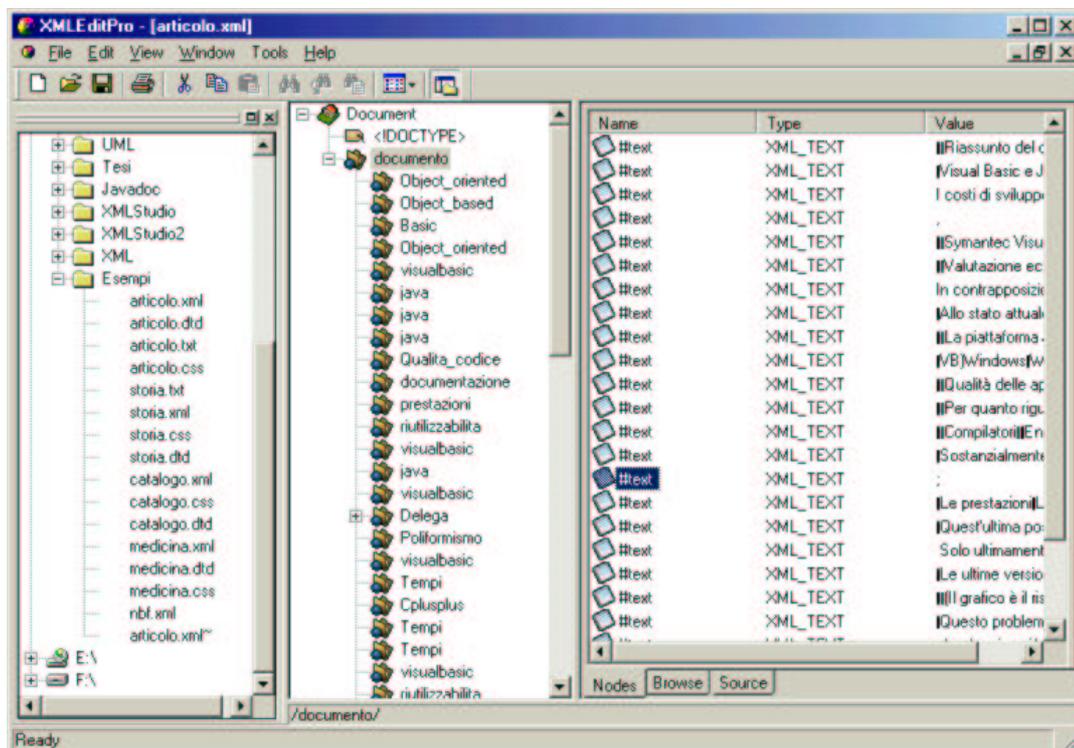


Figura 20 - XMLEditPro

4.1.4 – Cooktop

E' anch'esso un programma gratuito per scrivere e testare documenti XML, DTD e anche XPath (XML Path Language), fogli di stile XSLT (Extensible Stylesheet Language Transformations). Verifica se i documenti sono ben formati e validi. Converte file HTML in XHTML (Extensible Hypertext Markup Language) e anche file MS XSL (Microsoft Extensible Stylesheet Language) in XSLT. Può utilizzare Microsoft Word per il controllo ortografico e dispone di una chat integrata per parlare in modo anonimo con altri sviluppatori singoli o in gruppo, condividere URL e codice catalogato e riusabile con altri (chiamato Code-Bits). E' possibile visualizzare la sequenza dei marcatori utilizzati nel documento elaborato tramite un'apposita vista di nome *Structure Navigator*. E' estensibile e permette di utilizzare strumenti esterni come processori XSL, formattatori XML e altri browser. Dispone di un sistema di bookmark per memorizzare i numeri di linea per una successiva semplice identificazione e spostamento nel documento. In sintesi offre diverse caratteristiche disponibili in prodotti analoghi commerciali come il famoso XMLSpy.

» Sito Internet del programma: <http://www.xmlcooktop.com>

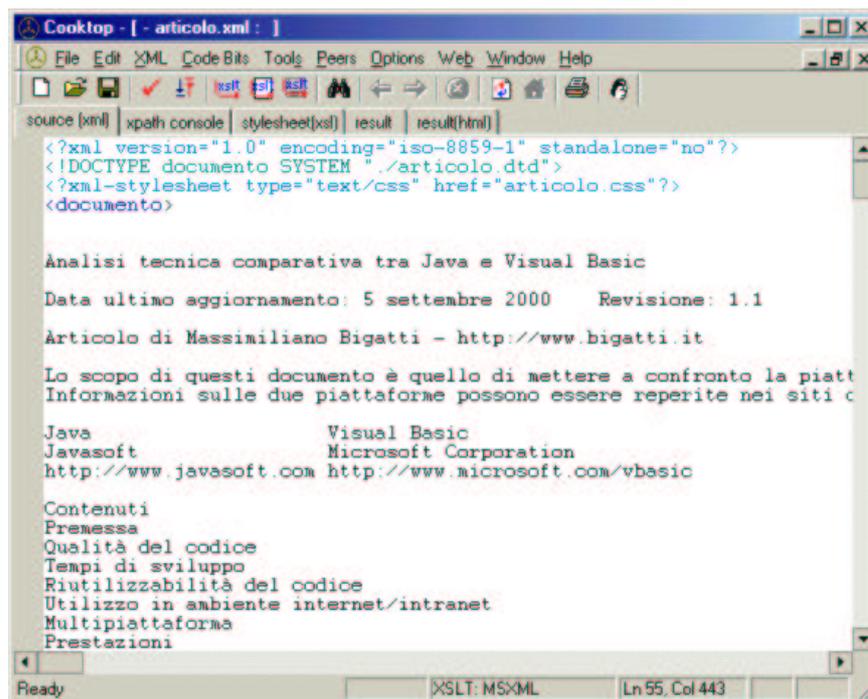


Figura 21 - L'editor XML Cooktop

4.1.5 – TXE The XML Editor

Questo programma gratuito scritto in Java per poter funzionare necessita del Java Runtime Environment (JRE) 1.2 o superiore e del parser Oracle.

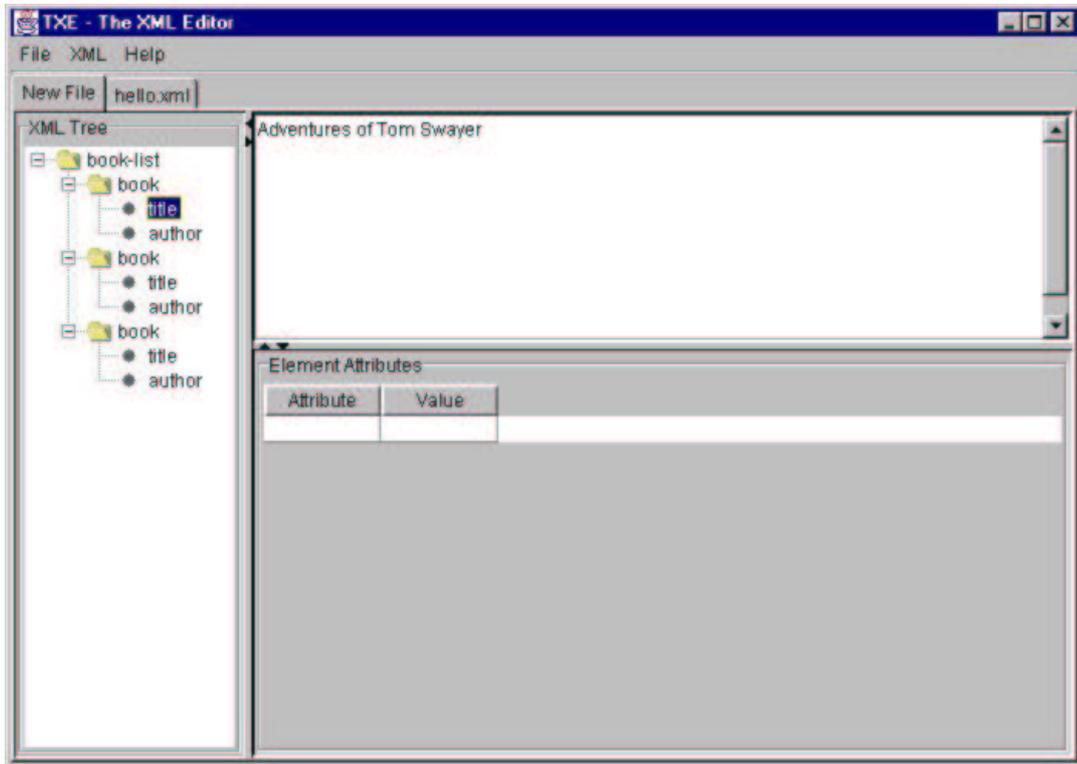


Figura 22 - TXE The XML Editor

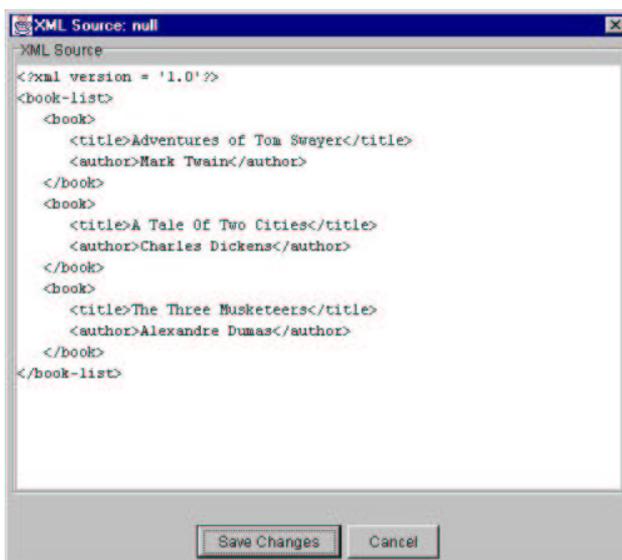


Figura 23 - Editing XML in TXE

E' un semplice programma per l'editing di file XML: è possibile aggiungere, modificare, eliminare i nodi e i rispettivi valori e attributi. Una vista consente di visualizzare e modificare direttamente il codice XML, salvando i cambiamenti effettuati. L'aspetto negativo di questo programma è che il

parser Oracle non è direttamente disponibile nella distribuzione standard di Java: va scaricato separatamente ed è necessario registrarsi.

» Sito Internet del programma: <http://www.geocities.com/shjejurkar/TXE/readme.html>

4.1.6 – XMLOperator

Questo programma, scritto in Java con licenza Open Source, consente di editare sia file XML che XHTML. Per il suo corretto funzionamento necessita l'uso di Java 2, del parser *Xerces-J* e del processore *Xalan-J*. Nella finestra del programma viene visualizzata una vista ad albero dei nodi che costituiscono il documento: quando si seleziona uno specifico nodo, questo viene mostrato in una vista dettagliata adiacente all'albero. Può editare documenti che includono o meno riferimenti a DTD. Vi sono comandi per la conversione ed il confronto dei documenti.

» Sito Internet del programma: <http://www.xmloperator.net>

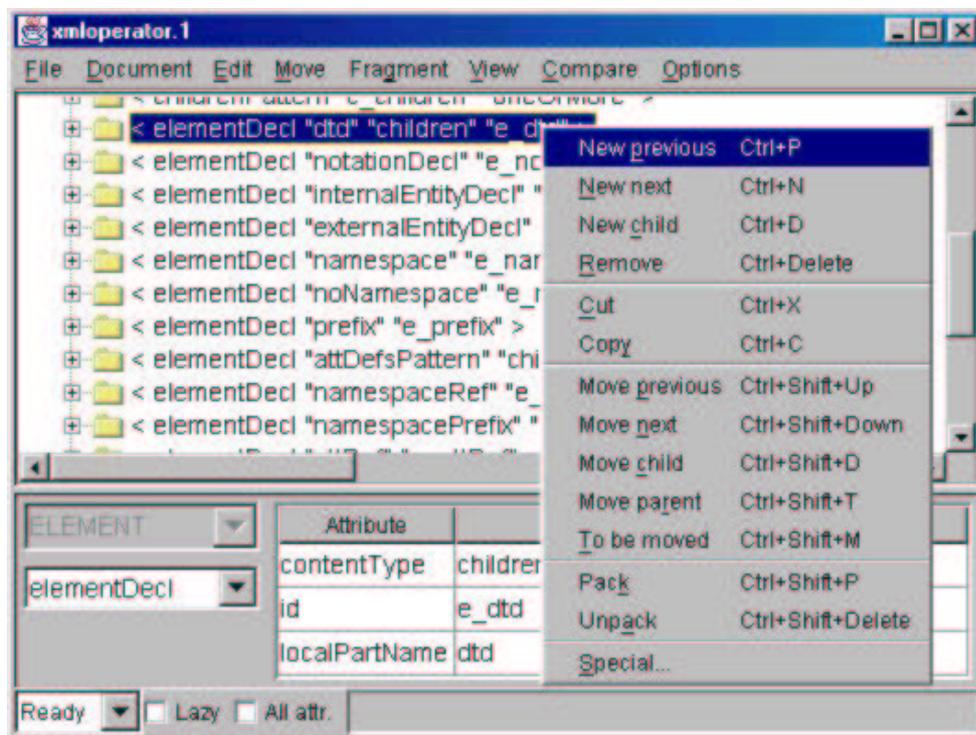


Figura 24 - L'editor XMLOperator

4.1.7 – Altri Editor XML

Un'intera tesi non basterebbe probabilmente per elencare tutti gli editor XML presenti sul mercato e per ognuno valutarne i pregi e difetti. Si

sono volutamente scelti tra di essi i principali gratuiti, liberamente scaricabili e utilizzabili da chiunque senza limiti. Chi fosse interessato ad approfondire l'argomento, eventualmente per trovare un programma adatto alle proprie esigenze, può visitare i seguenti siti internet:

- *XML Editors - Allegations of Functionality in search of reality*: articolo che recensisce in modo abbastanza esaustivo i principali editor XML.
<http://www.ivritype.com/xml>
- *PerfectXML - XML Editors*: sito internet specializzato in tutto ciò che ruota attorno al linguaggio XML.
<http://www.perfectxml.com/soft.asp?cat=6&sort=5>
- *XML Text Editor su Dmoz*: directory di ricerca che elenca i principali editor XML la cui inclusione è suggerita direttamente dagli utenti internet che a loro volta sono i responsabili dell'elenco proposto.
http://dmoz.org/Computers/Programming/Languages/Java/XML/Text_Editors
http://dmoz.org/Computers/Data_Formats/Markup_Languages/XML/Tools/Editors
- *XML Editors*: elenco riportato sul famoso sito www.xml.com dei principali editor XML disponibili. L'elenco è aggiornato al 2001.
<http://www.xml.com/pub/rq/115>
- *Scripting News*: semplice archivio di link a siti internet di editor XML.
<http://scriptingnews.userland.com/directory/1026/xmlEditors>
- *Free XML Tools and Software*: indice di applicazioni legate ad XML ordinate per nome, sistema operativo, produttore, ecc.
<http://www.garshol.priv.no/download/xmltools>

4.2 – Due approcci differenti per l'editing XML

Avendo analizzato alcuni editor per documenti XML, si è potuto osservare che l'approccio di questi programmi è generalmente differente rispetto all'idea che sta dietro ad XMLStudio: questi consentono di editare il file XML, in alcuni casi con funzionalità avanzate, permettendo di modificarne i nodi, gli attributi, i valori, ecc: quindi il loro obiettivo è

l'elaborazione da un punto di vista sintattico. XMLStudio consente invece di trasformare documenti generici in XML con il preciso intento di aggiungere ad essi conoscenza semantica data dai marcatori della sintassi di XML. Tra tutti i programmi provati, solo Visual_XML relativo al progetto Xdidattica è simile per impostazione a XMLStudio. Quindi potremmo abbozzare una classificazione di questi programmi suddividendoli in due grandi famiglie:

1. Editor generici per documenti XML, DTD, ecc. che consentono di aprire, modificare, salvare i file, modificare il contenuto dei nodi e dei loro valori ed attributi, ecc.
2. Editor che permettono di trasformare documenti in XML aggiungendo conoscenza ad essi tramite dei marcatori definiti dall'utente che grazie ad essi inserisce manualmente informazioni strutturali e semantiche al documento.

Si analizzerà a questo punto l'editor Visual_XML per evidenziare le analogie che ha con XMLStudio.

4.3 – L'editor Visual_XML

Questo programma è stato sviluppato da Davide Fiocchi ed è reperibile all'indirizzo <http://www.newgenesys.it/visualxml>. Lanciando il programma si presentano tre menù distinti che selezionati modificano l'aspetto dell'editor e delle sue funzionalità: il primo riguarda l'elaborazione di file XML, il secondo dei file XSL e il terzo dei DTD.

4.3.1 – Creazione visuale di file XML

Grazie alla sua innovativa interfaccia la creazione dei documenti XML diventa accessibile a tutti nel modo più semplice possibile, selezionando il testo ed etichettandolo con il mouse. L'interfaccia grafica è composta da un *Albero Elementi* che mostra una rappresentazione ad albero del documento XML il cui testo compare in un'area di nome *Testo* e il cui codice è presente in un'altra area di nome *Codice XML*. Indicato un nome all'etichetta, si possono assegnare dei valori o attributi per specificarlo in modo più dettagliato e meno generico. Vi è la possibilità di spostare

visualmente gli elementi all'interno del documento XML, trascinandoli nella posizione che si desidera grazie all'implementazione di una funzionalità di tipo *drag & drop*. Permette inoltre di creare una *struttura dati* che consente di unire alcuni documenti tra di loro.

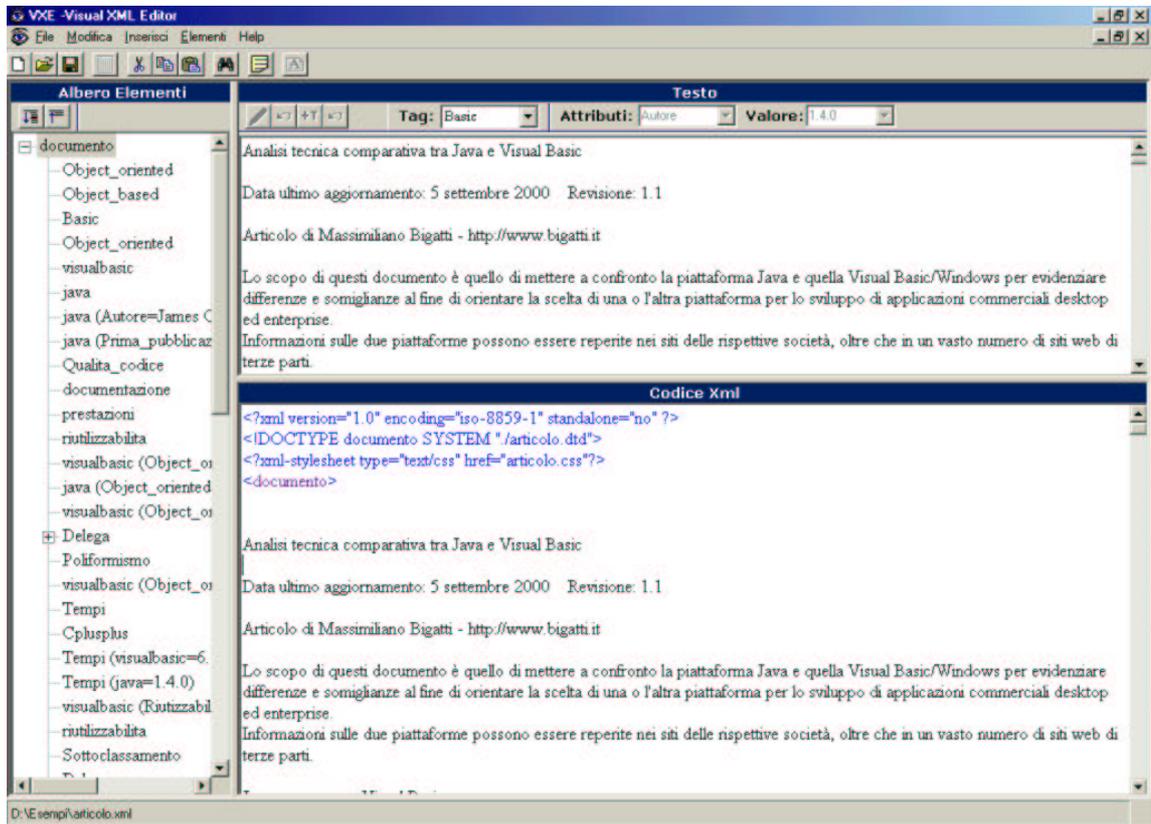


Figura 25 - Visual XML Editor, creazione di un documento XML

4.3.2 – Creazione visuale di file XSL

Grazie ad una serie di strumenti forniti dal programma, si semplifica la creazione dei fogli di stile: risulta semplice scegliere la formattazione grafica che maggiormente aggrada.

E' possibile generare automaticamente dei filtri senza conoscere la sintassi dei documenti XML: i filtri sono utilizzati per visualizzare o meno i contenuti delle etichette.

Quando si hanno più file da unire in un unico documento è possibile integrare documenti XML-XSL in strutture predefinite di frame. Altre

funzionalità consentono di creare hyperlink, collegamenti ipertestuali, inserire immagini.

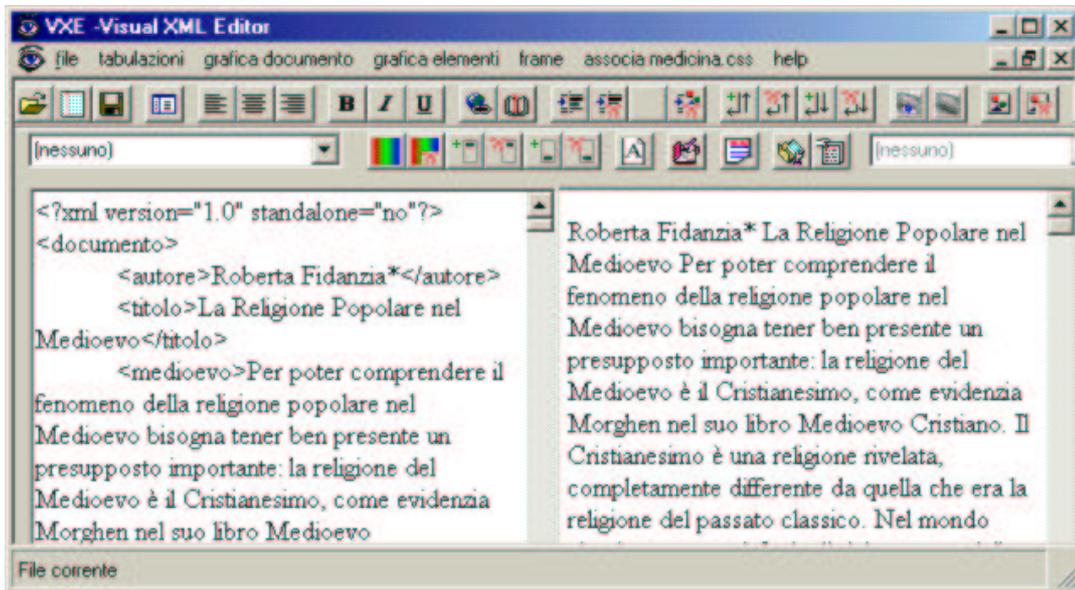


Figura 26 - Visual XML Editor, creazione di un documento XSL

4.3.3 – Creazione automatica di file DTD

Il file relativo alla *Document Type Definition* viene generato automaticamente dal file XML. Il programma si preoccuperà di controllare se future modifiche sono compatibili o meno con le regole incluse nella DTD. In questo modo si realizza un efficace strumento per il mantenimento dei documenti.

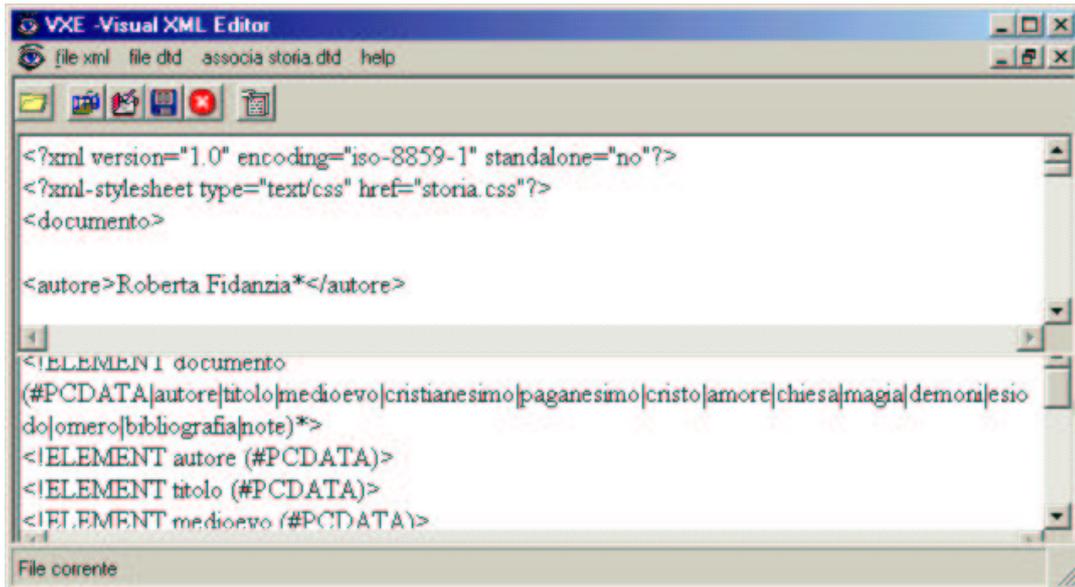


Figura 27 - Visual XML Editor, creazione di un documento DTD

4.4 – Confronto tra XMLStudio e Visual_XML

Entrambi gli editor hanno l'obiettivo di trasformare dei documenti testuali in XML associando ad essi dei file DTD, CSS, XSL e consentendo all'utente di inserire delle etichette XML per aggiungere informazioni semantiche e quindi conoscenza ai documenti.

Mentre Visual_XML dispone di più funzioni rispetto a XMLStudio, quest'ultimo essendo sviluppato in Java e non in Visual Basic presenta una maggiore portabilità consentendo di essere utilizzato su sistemi operativi differenti.

Entrambi i programmi sono semplici da usare, consentono a chiunque di comprendere la "filosofia" XML, nonché i documenti scritti in questo formato e potranno essere migliorati nel corso del tempo.

CAPITOLO 5

Conclusione e sviluppo di XMLStudio

5.1 – Lo sviluppo di XMLStudio

XMLStudio è un editor al quale potrebbe essere applicata una qualche licenza in modo da renderlo a tutti gli effetti un programma utile per essere scaricato dagli utenti. Come accade in tutti i progetti software, versioni successive potrebbero implementare nuove funzionalità. In questo capitolo esporremo alcune idee, emerse durante la realizzazione del programma, che sarebbe opportuno sviluppare.

5.1.1 – Internazionalizzazione di XMLStudio

Il programma è attualmente realizzato in lingua italiana e quindi tutti i menù, i comandi, i suggerimenti legati alle icone, l'help in linea e le finestre non sono comprensibili per utenti internazionali. Realizzare due versioni, una in italiano ed una in inglese, non è una soluzione produttiva mentre sarebbe più efficace inserire un'ulteriore *proprietà*⁵² impostata dall'utente relativa alla lingua e passare stringhe di testo differenti sia al momento della

⁵² Le proprietà di XMLStudio, caricate all'avvio del programma, sono salvate nel file *XMLStudio.default* incluso nella directory *Risorse*. Il file *XMLStudio.predefinito* contiene invece le proprietà predefinite del programma, in modo che una modifica accidentale delle stesse possa essere facilmente recuperata direttamente dall'interno del programma grazie al menù *Impostazioni/Ripristina impostazioni predefinite*.

creazione dei componenti all'avvio del programma, che nei momenti in cui risulta necessario interagire con l'utente con finestre di vario tipo.

5.1.2 – Fogli di stile XSL

Il linguaggio XSL (Extensible Stylesheet Language) è stato creato esclusivamente per l'utilizzo con i documenti XML: la sua struttura e la sua sintassi sono identiche a quelle dell'XML.

XSL è composto da due componenti: un linguaggio di *trasformazione* XSL ed una specifica di *formattazione* di un oggetto. Questi due elementi sono distinti, ma è possibile utilizzarli insieme per ottenere funzionalità di formattazione sofisticate per la visualizzazione del documento.

Il *linguaggio di trasformazione XSL* dimostra come un elaboratore può trasformare la struttura di un documento XML in un'altra struttura. L'utilizzo di questo linguaggio è quindi dato dal convertire un documento XML da una struttura semantica ad una struttura di visualizzazione, quale ad esempio la conversione di un documento XML in documento HTML può essere. In realtà questa non è l'unica possibilità dal momento che il processo di trasformazione è totalmente indipendente dal risultato finale: ciò può consentire una grande flessibilità dal momento che l'XSL potrebbe trasformare documenti in nuove strutture.

La *specifica di formattazione dell'oggetto* fornisce invece una nuova semantica di formattazione sviluppata come vocabolario XML utile per applicazioni specifiche come quelle multimediali.

XMLStudio attualmente non usa il linguaggio XSL ma solo i fogli di stile CSS: l'estensione di XMLStudio con il supporto di XSL consentirebbe l'uso di un fogli di stile più potenti, infatti gli stessi fogli di stile XSL sono documenti ben formati. Quando un pattern specificato nel documento XSL è riconosciuto nel documento XML, le regole trasformano l'XML corrispondente in qualcosa di totalmente nuovo. Sebbene i fogli CSS possano impostare solo il formato e la collocazione di elementi, XSL può riordinare gli elementi in un documento, modificarli totalmente, visualizzarne alcuni e nascondere altri, selezionare stili basati non solo

sugli elementi ma anche sugli attributi degli elementi e stili basati sulla posizione degli elementi.

5.1.3 – Fogli di stile CSS

I fogli di stile CSS vengono attualmente usati in XMLStudio per mostrare nel browser il documento XML visualizzato mettendo in evidenza porzioni di testo grazie al colore associato a determinate etichette.

Nella riga seguente è riportata una istruzione relativa alla parte dichiarativa di una regola di un file CSS generato con XMLStudio:

```
{ color : #99ffcc; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
```

Attualmente per ogni tag che si desidera è possibile cambiare solo il colore: un'idea, per un'ulteriore sviluppo, sarebbe legata alla possibilità di consentire una maggiore personalizzazione della regola e quindi del file CSS attraverso la selezione da parte dell'utente del carattere da usare, del suo colore, dello spessore dei bordi, e così via. Definire cioè un foglio di stile CSS maggiormente personalizzato in base alle esigenze di chi usa XMLStudio.

5.1.4 – Testo colorato

Operazione non immediata dovrebbe essere inserire come caratteristica la colorazione multicolore del testo e delle etichette del codice XML in quanto XMLStudio utilizza come area testo una *JTextArea* priva della proprietà di usare testo con più colori. Le alternative sarebbero o utilizzare controlli differenti come *JTextPane* o *JEditorPane* ma le modifiche da apportare al programma sarebbero considerevoli; senza poi contare che questi controlli sono in effetti più orientati ad una formattazione complessa ma non è così immediato trovare in rete materiale semplice e fruibile che permetta di far svolgere a questi funzionalità di una certa complessità come quelle attualmente implementate dalla *JTextArea* di XMLStudio. Le possibili soluzioni alternative potrebbero essere due: utilizzare (o eventualmente creare) un controllo esistente, magari commerciale che estenda la classe *JTextArea* aggiungendo ad essa la

proprietà di poter usare testo con colori multipli, oppure realizzare l’editing “colorato” in un’ulteriore finestra, un’altra vista indipendente dall’area testuale e dall’albero di parsing del documento XML. In questo secondo caso questa vista potrebbe essere rappresentata da una nuova finestra JDialog dotata di funzionalità essenziali per editare il documento. Qualche esempio a riguardo, che però non implementa funzioni avanzate su questi controlli, è allegato nella sezione demo del J2SDK.

5.1.5 – Selezione paragrafo tag

Una restrizione del programma è rappresentata dal limite di selezione automatica di un paragrafo compreso tra due etichette⁵³: questo è possibile solo se si posiziona il cursore in un qualsiasi carattere relativo ai marcatori più interni. Un’estensione del programma potrebbe consentire di selezionare in modo automatico un qualsiasi paragrafo compreso tra due tag XML del documento.

5.2 – Malfunzionamenti e problemi rilevati

Viene segnalato nel successivo paragrafo un malfunzionamento di XMLStudio la cui correzione non è possibile ma, dipendendo da un problema di esecuzione della JVM, sarà probabilmente automaticamente risolto in una successiva versione del J2SE 1.4.

5.2.1 – Problema legato al word-wrap

Il *word-wrap* sarebbe in italiano ciò che viene definito “A capo automatico” ed è una funzionalità introdotta nell’editor in quanto diversi documenti testuali senza di essa, sarebbero di difficile lettura.

Utilizzando intensamente l’editor, si è constatata un’anomalia che probabilmente dipende dal JDK 1.4.0: se si aprono consecutivamente diversi documenti testuali o XML, ad un certo punto, con una certa casualità nel verificarsi, viene mostrata la seguente eccezione:

⁵³ Il comando relativo a questa funzione viene lanciato grazie alla seguente voce dei menù:
Xml/Selezione paragrafo tag xml

```
java.lang.NullPointerException at
javax.swing.text.WrappedPlainView$WrappedLine.paint(WrappedPlainView.java:579)
...
```

Il verificarsi di questo problema è abbastanza casuale e non legato ad un particolare documento o ad una specifica sequenza di apertura dei documenti. Se viene disabilitata la funzionalità di word-wrap dell'area testuale, l'errore non si verifica. Per questo ritengo che il problema sia legato al JDK 1.4.0 e non sia risolvibile in qualche modo con righe di codice sorgente. Anche perché le linee di codice che implementano questa funzionalità sono soltanto due:

```
areaTesto.setLineWrap(wordWrapDefault.booleanValue()); // imposta se è attivo il word-wrap nell'area testo
areaTesto.setWrapStyleWord(wordWrapDefault.booleanValue()); // imposta lo stile di word-wrapping
```

escludendo quelle legate al salvataggio di questa proprietà, che non dovrebbero incidere sul corretto funzionamento del programma.

5.3 – Conclusioni

Con la crescente diffusione dei computer e di internet molti utenti sono diventati editori di documenti e la produzione di questi, con il passare del tempo, tenderà sempre ad aumentare. Da ciò si deduce e nasce l'esigenza di gestire in modo più adeguato questo enorme bagaglio di informazione che l'umanità produce. I motori di ricerca evidenziano problemi nell'indicizzare semplici pagine HTML e l'XML nasce anche con l'obiettivo di colmare questa lacuna in un futuro prossimo venturo.

XMLStudio, come progetto e applicazione, si inserisce in questa direzione, come tentativo che segue questa linea e idea. Sono molto contento per aver avuto la possibilità di sviluppare questa tesi, perché ho potuto realizzare un progetto che mi ha permesso di conoscere meglio Java ed il linguaggio XML. Spero che il lavoro svolto possa essere utilizzato da altri o per quanto riguarda un'eventuale estensione del programma XMLStudio o più semplicemente per ciò che può essere un suo utilizzo effettivo e pratico.

Appendice

Appendice A

Esempio d'uso di XML legato a pagine per siti internet inerenti il commercio elettronico.

Informazioni sul prodotto (vengono inseriti 2 prodotti):

- *Prodotto*: LEXMARK Z45.
- *Prezzo*: Euro 99,00 [Lit. 191.691].
- *Disponibilità*: Sì.
- *Garanzia*: 12 mesi.
- *Descrizione*: La nuova stampante di casa Lexmark è la Z45 Color Jetprinter una stampante a getto di inchiostro che garantisce la perfezione della qualità fotografica a 2400x1200 dpi, creata appositamente per raggiungere maggiori dettagli e sfumature di colore in fase di stampa.

Il file catalogo.xml (contenente 2 prodotti) risulta essere:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE documento SYSTEM "./catalogo.dtd">
<?xml-stylesheet type="text/css" href="catalogo.css"?>
<catalogo>

  <prodotto>
    <nome>LEXMARK Z45</nome>
    <prezzo><euro>191.691</euro><lire>99,00</lire></prezzo>
    <disponibilita>Sì</disponibilita>
    <garanzia>12 mesi</garanzia>
    <descrizione>La nuova stampante di casa Lexmark è la Z45 Color Jetprinter una stampante a getto di
    inchiostro che garantisce la perfezione della qualità fotografica a 2400x1200 dpi, creata appositamente per
    raggiungere maggiori dettagli e sfumature di colore in fase di stampa.</descrizione>
  </prodotto>

  <prodotto>
    <nome>Stampante DeskJet 656C</nome>
    <prezzo><euro>49,00</euro><lire>94.877</lire></prezzo>
    <disponibilita>Sì</disponibilita>
    <garanzia>12 mesi</garanzia>
    <descrizione>Facile da installare e da usare, consente di stampare in bianco e nero e a colori premendo un
    solo tasto. Una stampante decisamente versatile, in grado di gestire qualsiasi tipo di carta. Ideale per chi
    acquista la sua prima stampante!</descrizione>
  </prodotto>

</catalogo>
```

Il file catalogo.dtd elaborato da XMLStudio è:

```

<!ELEMENT catalogo ( prodotto+ ) >
<!ELEMENT descrizione ( #PCDATA ) >
<!ELEMENT disponibilita ( #PCDATA ) >
<!ELEMENT euro ( #PCDATA ) >
<!ELEMENT garanzia ( #PCDATA ) >
<!ELEMENT lire ( #PCDATA ) >
<!ELEMENT nome ( #PCDATA ) >
<!ELEMENT prezzo ( euro, lire ) >
<!ELEMENT prodotto ( nome, prezzo, disponibilita, garanzia, descrizione ) >
    
```

La rappresentazione strutturale ad albero di XMLStudio di catalogo.xml, risulta essere la seguente:

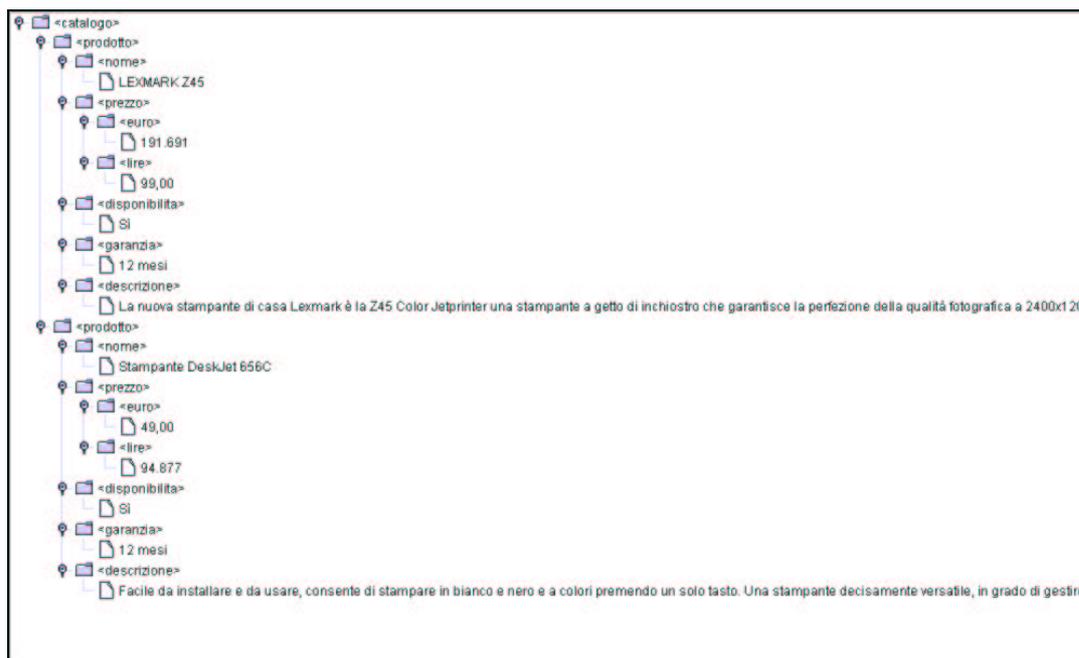


Figura 28 - Rappresentazione ad albero in XMLStudio di catalogo.xml

Per il file catalogo.xml si è creato grazie ad XMLStudio il seguente foglio di stile catalogo.css:

```

documenta { color : #000000; font-family : Arial Black; }
prodotto { color : #0033ff; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
nome { color : #ff0033; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
prezzo { color : #999900; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
lire { color : #009933; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
    
```

```
euro { color : #00cc99; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
garanzia { color : #3300cc; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
descrizione { color : #333333; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
disponibilita { color : #334444; border : solid 1px; font-family : Arial Black; padding: 4px; width: 100%; }
```

che mostrato nel browser viene così visualizzato:

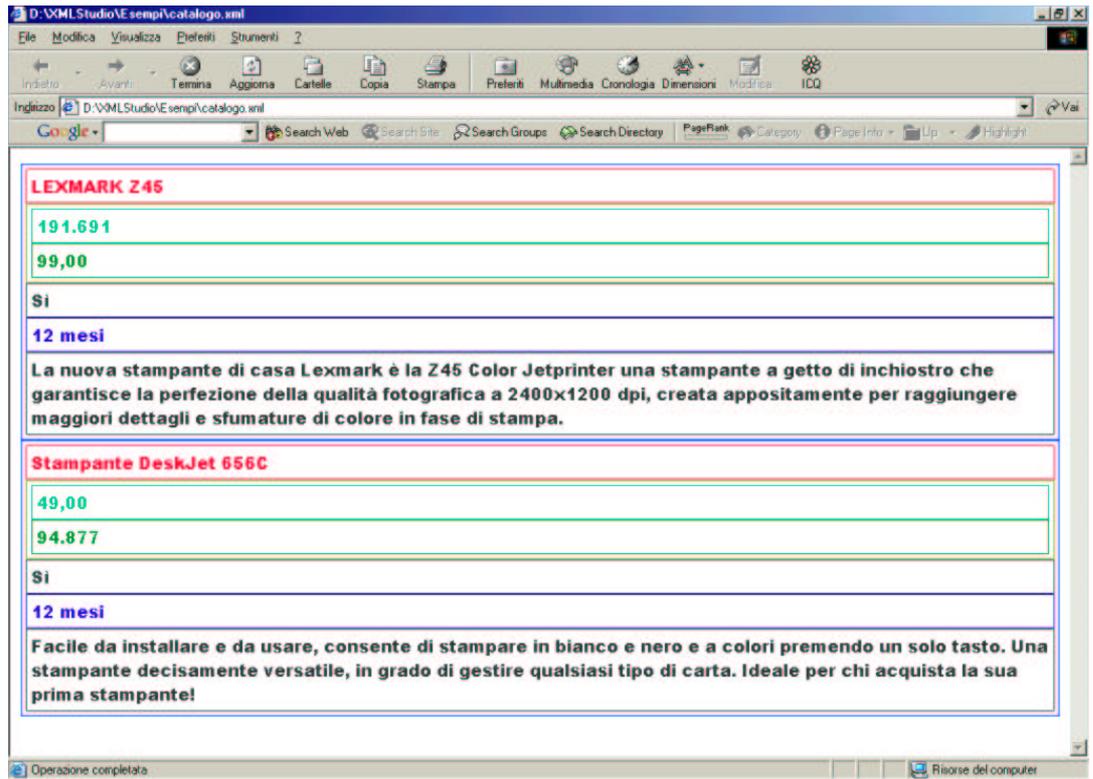


Figura 29 - Internet Explorer 6.0 mostra il file catalogo.xml della directory esempi

Appendice B

Nella directory *esempi* di XMLStudio e nelle pagine seguenti sono riportati file relativi ad esempi d'uso del programma. Gli articoli presi in considerazione sono semplici file testuali scelti in modo casuale dopo una breve ricerca con il motore di ricerca <http://www.google.it> (sono state utilizzate le parole chiave “articolo informatica”, “articolo medicina” e “articolo storia”). Per ogni articolo testuale elaborato sono stati prodotti i rispettivi file XML, DTD, CSS e XML dei marcatori. Segue la loro descrizione:

- *Articolo*: articolo informatico di Massimiliano Bigatti dal titolo: “*Analisi tecnica comparativa tra Java e Visual Basic*”, pubblicato sul sito internet personale: <http://www.bigatti.it>.
- *Medicina*: articolo medico dal titolo “L’artrosi senza cure” pubblicato sul sito internet di medicina: <http://www.medweb.it>.
- *Storia*: articolo storico di Roberta Fidanzia dal titolo: “*La Religione Popolare nel Medioevo*”, pubblicato sul sito internet che tratta di storia: <http://www.storiaonline.org>.

Esempio di articolo informatico

Articolo.xml

» Descrizione: articolo informatico trasformato in file XML al quale sono state aggiunte etichette XML riportate nel file XMLStudio.xml.

» Directory: ./XMLStudio/Esempi

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE documento SYSTEM "./articolo.dtd">
<?xml-stylesheet type="text/css" href="articolo.css"?>
<documento>
```

Analisi tecnica comparativa tra Java e Visual Basic

Data ultimo aggiornamento: 5 settembre 2000 Revisione: 1.1

Articolo di Massimiliano Bigatti - <http://www.bigatti.it>

Lo scopo di questi documento è quello di mettere a confronto la piattaforma Java e quella Visual Basic/Windows per evidenziare differenze e somiglianze al fine di orientare la scelta di una o l'altra piattaforma per lo sviluppo di applicazioni commerciali desktop ed enterprise.

Informazioni sulle due piattaforme possono essere reperite nei siti delle rispettive società, oltre che in un vasto numero di siti web di terze parti.

Java	Visual Basic
Javasoft	Microsoft Corporation
http://www.javasoft.com	http://www.microsoft.com/vbasic

Contenuti

Premessa

Qualità del codice

Tempi di sviluppo

Riutilizzabilità del codice

Utilizzo in ambiente internet/intranet

Multipiattaforma

Prestazioni

Qualità delle applicazioni realizzate

Accesso alle informazioni

Utilizzo da parte dell'industria

Valutazione economica dei costi

Citazioni

Premessa

```
<Object_oriented>Java è un linguaggio object
oriented.</Object_oriented> <Object_based>Microsoft Visual Basic è
semplicemente un linguaggio object-based. Questo significa che con
VB non è possibile sviluppare reali soluzioni orientate agli
oggetti: l'unico approccio agli oggetti che viene fornito da VB è
quello di racchiudere dati e codice in una unica entità.
Questa limitazione risulta essere intrinseca nella natura di Visual
Basic, in quanto il linguaggio che è alla base di VB è, appunto,
una forma aggiornata di Basic (vedi i cenni storici).
</Object_based>
```

Il confronto tra Visual Basic e Java non è quindi un confronto tra due linguaggi della stessa classe. Sarebbe più opportuno confrontare Java con il C++ e Visual Basic con Visual dBase (per citarne uno dei tanti).

Sebbene sussistano queste differenze, da molte parti si mettono a confronto i due linguaggi per operare una scelta sulla piattaforma da adottare per lo sviluppo di applicazioni, in special modo se legate ad internet/intranet.

Per questo motivo si è deciso di descrivere qui un confronto di questo tipo.

Questo documento confronta Java e Visual Basic in diverse aree critiche, quali:

- la qualità del codice;
- i tempi di sviluppo;
- la riutilizzabilità del codice;
- l'utilizzo in ambiente internet/intranet;
- la multipiattaforma;
- le prestazioni;
- la qualità delle applicazioni realizzate;
- l'accesso alle informazioni;
- l'utilizzo da parte dell'industria;
- la valutazione economica dei costi;
- una sezione di citazioni.

Al termine del testo, sono presenti alcune conclusioni, che tirano le somme su quanto esposto.

Cenni storici

<Basic>Il linguaggio Basic è stato creato nei lontani anni '70 e, nel tempo, ha subito molte modifiche per cercare di restare al passo con i tempi. La Microsoft stessa ha prodotto molte versioni di Basic (per PDP, Altair, Z80, 8086) spingendone lo sviluppo a livelli eccellenti per l'epoca. La versione 4.5 di QuickBasic sarà senz'altro nota a molti e, in Italia, è stata utilizzata anche per lo sviluppo di applicazioni commerciali. </Basic>

<Object_oriented>Negli anni 90 è nata (o meglio, è passata alla ribalta) la programmazione orientata agli oggetti. Alcuni linguaggi sono stati creati appositamente (come SmallTalk od Eiffel) per supportare il paradigma della programmazione ad oggetti; altri linguaggi sono stati aggiornati per cercare di sfruttare il paradigma degli oggetti. Visual Basic, insieme a Object Pascal, o al C++, è uno di questi. </Object_oriented>

<visualbasic>Visual Basic, nella fattispecie nella sua componente Visual, non è nato in casa Microsoft. Il concetto di sviluppo visuale è stato inventato da Alan Cooper. Nelle finestre di about di Visual Basic, infatti, fino alla versione 3.0 era presente l'indicazione che parte del software era prodotto da Cooper Software. Visual Basic è stato pensato con lo scopo di fornire uno strumento per poter sviluppare semplicemente sulla piattaforma Windows. Affrontare il C, o il C++, non era infatti alla portata di tutti.

Il risultato ottenuto, però, non può dirsi ottimale: se nel caso di Object Pascal, o, ancora meglio, del C++, l'integrazione con il legacy è sufficientemente buona, in caso di VB notiamo una difficoltà maggiore ad integrare in un linguaggio così datato il nuovo paradigma ad oggetti. Il problema maggiore è che non si vede il modo di far avanzare ulteriormente lo sviluppo di tale linguaggio.</visualbasic>

<java>Java, al contrario, è un linguaggio moderno basato completamente sugli oggetti che non soffre di legacy (come ad esempio ne ha sofferto il C++).

Java è stato creato ispirandosi ai linguaggi già esistenti, sia object oriented che procedurali, ed ha riassunto in una nuova entità quello che di buono risultava esserci in ciascuno di questi. Il risultato di questa progettazione è un linguaggio chiaro, elegante ed efficiente.</java>
 <java Autore="James Gosling">L'autore di Java, James Gosling,</java> <java Prima_pubblicazione="23 maggio 1995">ha presentato al pubblico per la prima volta la sua creatura il 23 maggio 1995.</java>

<Qualita_codice>Qualità del codice
 La qualità del codice non è una caratteristica che deve solo interessare lo sviluppatore ed il capo progetto. Anche a livello di management è importante valutare quale strumento produce codice di maggiore qualità, perché un codice di qualità scadente ha diversi problemi:
 i tempi di modifica si allungano;
 i tempi di documentazione si ampliano;
 le prestazioni possono deteriorare;
 la riutilizzabilità è limitata.
 Non si deve sottovalutare nessuno di questi aspetti. Sebbene il primo punto è probabilmente quello più facilmente percepibile dal management, bisogna considerare anche l'impatto degli altri problemi. </Qualita_codice>
 <documentazione>E' inoltre necessario valutare anche i tempi legati alla produzione di documentazione. L'esperienza ci insegna che un sistema software complesso e con documentazione nulla ha un impatto manutentivo molto grande. E' buona norma considerare un livello seppur minimo di documentazione. Documentare un sistema non di qualità è più difficoltoso, e di conseguenza, lungo ed antieconomico. </documentazione>
 <prestazioni>Le prestazioni di un sistema di scarsa qualità possono essere scarse. Questo può essere dovuto a scelte di progetto obbligate dalle limitazioni del linguaggio utilizzato per lo sviluppo. Generalmente i linguaggi procedurali supportano peggio dei sistemi object-oriented progetti di media complessità. </prestazioni>
 <riutilizzabilita>La riutilizzabilità, infine, è un'obbiettivo primario che il paradigma object-oriented si pone di raggiungere: nel corso dello sviluppo dei linguaggi informatici è risultato chiaro che l'approccio procedurale spinge ad una scarsa riutilizzabilità del codice. La riutilizzabilità nei linguaggi procedurali è limitata al riuso tramite librerie di funzioni. I sistemi object-oriented hanno meccanismi più avanzati di riutilizzo, quali il sottoclassamento e lo strategy pattern, in aggiunta ovviamente al mero riutilizzo del codice in stile procedurale. </riutilizzabilita>
 <visualbasic Object_oriented="No">Visual Basic è un linguaggio procedurale</visualbasic>, mentre <java Object_oriented="Si">Java è object-oriented</java>. In alcuni casi Visual Basic è la strada più breve anche se la via più breve non è quasi mai la migliore.

<visualbasic Object_oriented="No">Visual Basic come linguaggio ad oggetti
 Microsoft sostiene però che Visual Basic è un linguaggio ad oggetti. Questa pretesa è basata su una visione ridotta dell'object oriented secondo la quale un linguaggio OO è definito dai seguenti tre concetti:

può creare oggetti;
 ereditarietà;
 polimorfismo.

Microsoft dichiara che Visual Basic può creare oggetti, e questo è vero. Non possiede ereditarietà, ma questa viene simulata con la delega; il polimorfismo viene implementato tramite i parametri opzionali.

La cosa interessante è che secondo questa definizione, anche il linguaggio Clipper (compilatore per dBase, vecchia applicazione database per DOS) è un linguaggio ad oggetti, il che è quantomeno esagerato.</visualbasic>

<Delega><Ereditarieta>Per quanto riguarda la delega, questa non rappresenta un valido sostituto all'ereditarietà: si pensi ad una classe dotata di molti metodi ed a una sottoclasse che aggiunga anche solo un metodo alla classe padre. Questo è un caso frequente, ma con la delega si è costretti a riscrivere tutti i metodi nella sottoclasse richiamando i metodi relativi dell'oggetto delegato.</Ereditarieta></Delega>

<Poliformismo>In merito al polimorfismo di Visual Basic: la programmazione ad oggetti ha lo scopo di rendere il codice più chiaro. In un linguaggio OO, ogni metodo polimorfico ha differenti corpi; utilizzare i parametri opzionali per simulare il polimorfismo, oltre a non consentire la stessa libertà, costringe a mettere tutto il codice in un unico metodo di fatto complicando la chiarezza del codice.</Poliformismo>

<visualbasic Object_oriented="No">Alcuni tecnici di Microsoft non sostengono la teoria di marketing secondo la quale VB è OO. Basta consultare MSDN per vedere che, in realtà, l'implementazione OO di Visual Basic è legata al modello a componenti COM.</visualbasic>

<Tempi>I tempi di sviluppo

I tempi di sviluppo sono un elemento critico in fase di considerazione di una piattaforma tecnologica. Il management ed il responsabile di progetto sono sempre, e giustamente, orientati all'uso di una tecnologia che fornisce i minori tempi di sviluppo. Di primo acchito può sembrare che l'utilizzo di VB porti a tempi di sviluppo inferiori. In realtà un'analisi più approfondita evidenzia che ciò non è quasi mai vero. Può succedere che per alcuni casi molto particolari, Visual Basic sia la scelta migliore; nella maggior parte dei casi questo linguaggio risulta insufficiente per la produzione di applicazioni di qualità commerciale. </Tempi> Questo problema è insito nella erronea natura del confronto tra VB e Java.

<Cplusplus>Il C++, se utilizzato al posto di Visual Basic, consente la produzione di applicazioni di qualità più elevata. L'altra faccia della medaglia è che il C++ è più complesso di Java (sintassi, ereditarietà multipla) e richiede risorse più esperte ed in genere molto più tempo per lo sviluppo. </Cplusplus>

(Il grafico è il risultato di una stima basata sull'esperienza pratica)

<Tempi visualbasic="6.0">In sintesi, con Visual Basic è possibile produrre in tempi più ridotti applicazioni artigianali. Caratteristiche complesse (non presenti negli strumenti di base o nelle estensioni) richiedono un tempo maggiore per poter essere implementate rispetto a Java. </Tempi>

<Tempi java="1.4.0">Con Java il tempo di produzione di un prototipo può risultare maggiore che con VB. La produzione della versione finale è più veloce in Java che in Visual Basic. </Tempi>

<visualbasic Riutilizzabilita="Parziale">Questa differenza è dovuta al fatto che in Visual Basic non è possibile operare un riutilizzo

del codice "vivo": l'unico riutilizzo possibile è un copia/incolla del codice interessato. Questo preclude il riutilizzo di risorse quali Form o controlli personalizzati. </visualbasic>

Le ultime versioni di Visual Basic consentono una sorta di sottoclassamento dei controlli aggiuntivi con cui è possibile creare ocx personalizzati. Questo meccanismo è ancora però molto lontano dal riuso del codice in stile object-oriented.

<riutilizzabilita>Riutilizzabilità del codice

La riutilizzabilità del codice è una caratteristica basilare nel moderno sviluppo software. Con il progresso dell'informatica, il trend non è più riscrivere da zero (reinventare la ruota), ma quello di riutilizzare, assemblare, comporre. Lo sviluppo di codice a più basso livello sta diventando più raro rispetto alla composizione ad alto livello. La parte difficile nello sviluppo del software sta diventando sempre di più la capacità di conoscere un vasto ventaglio di API, specifiche e protocolli, rispetto alla capacità di concretizzare soluzioni di programmazione.

Java è un linguaggio orientato agli oggetti. La programmazione agli oggetti è un paradigma nato per risolvere il problema della riutilizzabilità. Un linguaggio orientato agli oggetti consente:

</riutilizzabilita>

<Sottoclassamento>il sottoclassamento di classi esistenti allo scopo di riutilizzarne il codice;</Sottoclassamento>

<Delega>la delegazione a classi esterne da quella in oggetto per ottenere un servizio; </Delega>

<riutilizzabilita>il riutilizzo del codice con classiche operazioni di copia/incolla.</riutilizzabilita>

<visualbasic Riutilizzabilita="Parziale">Per contro Visual Basic, come linguaggio procedurale, possiede solo il riuso del codice classico (riutilizzo di moduli e copia/incolla, per chi non sa sviluppare molto bene).</visualbasic> Solo ultimamente VB fornisce meccanismi simili al paradigma object-oriented per il riuso del codice: questo sembra però un ulteriore patchwork applicato al vecchio Basic per cercare di prolungarne la vita.

<riutilizzabilita>La possibilità di riutilizzare il codice è un elemento essenziale che deve essere valutato dal management quando considera una tecnologia. Riutilizzare significa risparmiare tempo di sviluppo e di conseguenza, abbattere i costi.</riutilizzabilita>

<java Internet="Si">Utilizzo in ambiente internet/intranet

Java

Java è la piattaforma per lo sviluppo in ambiente internet/intranet. Questa piattaforma è stata studiata per operare in ambito di networking e supporta un ampio ventaglio di possibilità per la comunicazione tramite internet:

socket;

socket UDP;

url;

Il package Java java.net, presente nella piattaforma Java standard, fornisce l'accesso a questi elementi.

Inoltre, è disponibile il supporto alla comunicazione locale, tramite porta seriale o parallela. Questo supporto è presente nell'estensione standard Java Serial Port (in precedenza Java Communication API).

Inoltre è possibile gestire l'esecuzione di codice in remoto, tramite RMI (Remote Method Invocation) o tramite il mapping a CORBA. Quest'ultimo è un robusto standard studiato da un consorzio di produttori che consente l'esecuzione di codice distribuito su server diversi (su hardware e piattaforme software eterogenee).

In particolare, nella versione 2 di Java, il supporto a CORBA è stato integrato nelle core api della piattaforma.

Java può essere presente sul server (servlet, server scritti in Java), e sul client (sia come applet che come applicazioni). Il codice è sempre composto da classi Java. </java>

<visualbasic Internet="Parziale">Visual Basic

Visual Basic non è un linguaggio nato per il networking. Abbiamo ripercorso la storia del linguaggio Basic in precedenza. All'epoca della creazione del Basic non si parlava neanche di networking locale tramite LAN. Le versioni di Basic precedenti al Visual Basic 3.0 non supportano direttamente nessun tipo di rete. Con le versioni più recenti, in particolare con le versioni 5.0 e 6.0, si può notare un'attenzione maggiore al networking, anche se tramite estensioni e aggiunte applicate in tempi successivi. </visualbasic>

<applet>Una caratteristica importante di Java, che ne ha aiutato particolarmente la diffusione, è il codice integrato in pagine HTML: le famose applet. Le applet sono piccoli programmi scaricati da un sito web ed in grado di girare dal lato client. Come tutto il codice Java, le applet possono girare su un ampio ventaglio di piattaforme (vedi multipiattaforma). </applet>

<activex>Visual Basic fornisce un supporto simile alle applet tramite ActiveX. ActiveX è una ulteriore rinominazione della tecnologia OLE presente sulla piattaforma Windows. OLE è un sistema di collegamento ed incastro (Object linking and embedding) di oggetti "pesanti" per la piattaforma Windows. Gli oggetti sono denominati "pesanti" perché richiedono molto codice per poter essere implementati. Anche un oggetto OLE "vuoto" richiede che siano implementate molte interfacce. Il codice che risulta è vasto anche in caso di oggetti OLE molto semplici. Per ovviare a questo inconveniente, che sarebbe risultato cruciale in ambito internet/intranet, dove i tempi di download sono ancora abbastanza elevati, la Microsoft ha sviluppato una versione "leggera" di OLE, denominata ActiveX. I controlli ActiveX devono implementare un numero inferiore di interfacce ed il codice che ne risulta è inferiore rispetto ad un controllo OLE.

Si noti che, comunque, il codice richiesto per un controllo ActiveX è molto maggiore rispetto a quello necessario per implementare una classe Java. </activex>

<activex>I controlli ActiveX, come <applet>le applet Java, possono essere eseguiti all'interno di un browser web. Il problema è che l'unica piattaforma supportata è Windows 95/98/NT e l'unico browser utilizzabile, è Microsoft Internet Explorer.</applet>

<visualbasic Internet="Parziale">Il supporto internet/intranet di Visual Basic è quindi largamente incompleto, in quanto alla soluzione adottata, lo standard proprietario ActiveX, funziona solo su una singola piattaforma/browser</visualbasic>. </activex> Quest'ultima possibilità esclude quindi un largo numero di utenti internet dalla base dei possibili utenti dell'applicazione.

<plug-in>Java Plug-in

Per utilizzare anche le ultimissime versioni di Java su browser che ancora internamente non le supportano, è possibile utilizzare il Java Plug-in. Questo è un controllo ActiveX che interfaccia una VM a scelta tra quelle installate nel sistema; in questo modo è possibile utilizzare, ad esempio Java2 in Netscape 3.0.</plug-in>

```
<java Multiplatforma="Si">Multiplatforma
Lo sviluppo di internet ha portato in luce un problema
relativamente nuovo per gli sviluppatori di software: la
possibilità di eseguire l'applicazione su piattaforme hardware e
software differenti.
Il mondo dei server internet è formato, per lo più, da server Unix
ed NT. I client che si collegano ad internet sono formati da
configurazioni più disparate: client unix, PC dos, PC windows
3.1/95/98/NT, PC linux, Macintosh, palmari, etc..
Lo sviluppo di applicazioni che funzionino in ambito
internet/intranet richiede che il codice sviluppato sia compatibile
con un numero disparato di piattaforme.
La piattaforma Java è presente su un un numero impressionante di
piattaforme (persino su Amiga e PalmPilot).</java> <visualbasic
Multiplatforma="No">Al contrario Visual Basic produce eseguibili
che possono funzionare solo su Windows 95/98 ed NT. Anche restando
nella piattaforma Windows, notiamo che il supporto alle precedenti
versioni di Windows è stato soppresso. La piattaforma Windows 3.1
non è infatti più supportata da Visual Basic a partire dalla
versione 5.0. Ci si può quindi aspettare che con Windows 2000 esca
una nuova versione di Visual Basic incompatibile con la precedente
e con la precedente base di installato Windows.
```

Il problema del supporto delle piattaforme di VB è curioso, specialmente se consideriamo solo la singola piattaforma Windows. Si possono considerare Windows 3.1, 95, 98, 98 second edition, NT, 2000, CE piattaforme diverse della stessa famiglia. Visual Basic supporta solo un piccolo sottoinsieme della famiglia Windows. Questo è particolarmente strano se si considera che il produttore della piattaforma e del linguaggio è la stessa società. Potrebbe sembrare che, con una accorta politica di compatibilità, il produttore promuova l'acquisto continuo di aggiornamenti dei sistemi operativi e degli ambienti di sviluppo, "costringendo" l'utente a continui e costosi aggiornamenti. (Vedi valutazione economica dei costi). </visualbasic>

```
<java Multiplatforma="Si">Per quanto riguarda il funzionamento
multiplatforma, Java risulta senza ombra di dubbio il linguaggio
più adatto a questo scopo.</java>
```

Le prestazioni

Le prestazioni sono e sono sempre state un fattore critico nelle applicazioni informatiche. Entrambi i linguaggi qui esposti non sono nativi: si appoggiano ad uno strato intermedio di codice: <p-code>in Visual Basic questo è il p-code</p-code>; <byte-code>in Java è il bytecode.</byte-code>

Sostanzialmente p-code e bytecode significa la stessa cosa. Entrambi i linguaggi sono quindi interpretati. Un linguaggio interpretato è per forza di cose più lento di uno nativo. Il codice nativo è codice composto da istruzioni che possono essere eseguite da un microprocessore specifico; il codice interpretato deve essere decodificato da un interprete che trasforma questo codice in codice nativo.

```
<java Sicurezza="Si" Prestazioni="Discrete">Java ha diverse
caratteristiche innovative ed importanti, quali la sicurezza e la
portabilità (oltre che la dinamicità). Entrambe queste
caratteristiche si basano sul fatto che Java è interpretato. La
scelta di utilizzare un'architettura interpretata (e quindi
un'ulteriore strato di astrazione) è quindi una decisione che porta
notevoli vantaggi. Il lato negativo è che questo strato aggiunto
riduce le prestazioni dell'applicazione.</java>
```

```
<visualbasic Interpretato="Si"><visualbasic Multiplatforma="No"
Sicurezza="Scarsa">Visual Basic non è portabile (come abbiamo già
visto nella sezione multiplatforma) e non è dotato di
caratteristiche di sicurezza intrinseche. La scelta di utilizzare
un'architettura interpretata è infatti più che altro dovuta a
particolari scelte di progetto ed a retaggi storici. </visualbasic>
Per prima cosa ricordiamo che storicamente il Basic, ed in
particolare tutte le precedenti implementazioni realizzate da
Microsoft, era un linguaggio interpretato; è facile pensare che per
realizzare Visual Basic, la Microsoft si sia basata sulla
precedente versione DOS. D'altronde, anche da Visual C++ traspare
un retaggio del compilatore C/C++ per DOS della Microsoft.
Inoltre, bisogna notare che Visual Basic è storicamente un prodotto
rivolto allo sviluppo nelle grandi aziende (corporate) e allo
sviluppo non professionale. Probabilmente nella strategia
dell'offerta dei prodotti Microsoft, Visual Basic si colloca ad un
profilo più basso, mentre per applicazioni di qualità commerciale,
viene proposto <visualplusplus>Visual C++</visualplusplus>. E'
quindi immediato capire come mai Visual Basic sia interpretato e
non compilato.</visualbasic>
```

Compilatori

Entrambi i linguaggi possono però essere compilati in codice nativo per aumentarne le prestazioni.

```
<java Compilato="Si">In Java sono presenti due possibilità: i JIT
(Just In Time compilers) ed i compilatori nativi. Nel primo caso la
compilazione avviene "al volo": il codice presente nella JVM viene
trasformato in codice nativo ed eseguito. In questo modo la
portabilità del codice è mantenuta: il codice dei file di classi è
sempre java bytecode e quindi ne mantiene tutte le caratteristiche
di portabilità, sicurezza e dinamicità. Nel secondo caso troviamo
che il codice risultato della compilazione nativa è assimilabile ad
un comune eseguibile nativo prodotto da altri compilatori, quali
quelli per C/C++. Nel secondo caso, però, la portabilità ne
risente: il codice nativo è specifico di una singola piattaforma.
</java>
```

Per quanto riguarda Visual Basic, dalla versione 5.0 è presente la possibilità di generare codice nativo. Il codice generato non è però paragonabile ad un ipotetico corrispettivo prodotto in C/C++: sussiste anche in questo caso un livello intermedio.

Hotspot ed IBM

Le tecnologie Java sono in continua evoluzione e miglioramento: recentemente due rilasci rivestono particolare importanza nell'ambito delle performance di Java:

il JDK 1.1.7 di IBM;

la nuova JVM Hotspot di Javasoft;

Il primo è una versione del JDK Javasoft riveduta dalla IBM: la JVM fornita con questo JDK ha prestazioni eccellenti e dimostra come anche un linguaggio interpretato possa essere veloce. Questa tecnologia è la dimostrazione pratica che una delle critiche che viene rivolta a Java, quella di essere lento, è in realtà senza fondamento.

Hotspot è invece la nuova JVM della Javasoft. In pratica questo è un nuovo compilatore JIT che si basa su tecnologie adatte in grado di aumentare sensibilmente le prestazioni di un'applicazione

Java. Questa tecnologia è presentata come una tecnologia server: non ottimizza infatti due parti molto importanti di Java sul lato client: la AWT e Swing.

Ulteriori informazioni possono essere reperite a:

www.ibm.com/java

www.javasoft.com/products/hotspot

Conclusione

<Prestazioni>Le prestazioni di Java sono paragonabili a quelle di Visual Basic. Entrambe le piattaforme sono però più lente di una similare applicazione scritta in C/C++.
Si noti, però, che la tecnologia Java è in costante miglioramento da parte di molti grandi produttori di software (Javasoft, IBM, Inprise, Symantec), mentre quella legata a Visual Basic è legata allo sviluppo di una singola compagnia (Microsoft) che notoriamente non ha apportato grosse migliorie al linguaggio nel corso del tempo.</Prestazioni>

Qualità delle applicazioni realizzate

Tramite Java è possibile realizzare applicazioni di qualità commerciale: per tutti i principali elementi presenti in un progetto commerciale, Java dispone di soluzioni professionali.

Ne riportiamo qui alcuni dei principali:

elaborazione distribuita - CORBA, RMI;

interfaccia utente - AWT, Swing;

modello ad oggetti;

gestione dati - JDBC, XML.

Le tecnologie Java sono particolarmente flessibili e la sua struttura ad oggetti ne facilita la personalizzazione.

Visual Basic non è dotato di tecnologie comparabili a quelle presenti sulla piattaforma Java: si appoggia esclusivamente ai servizi di Windows. Qui troviamo:

elaborazione distribuita - DCOM;

interfaccia utente - Windows;

modello ad oggetti - solo object-based;

gestione dati DAO, ODBC.

Visual Basic risulta quindi un'interfaccia al mondo Windows.

Per confrontare la qualità delle applicazioni realizzate sulle singole piattaforme, dobbiamo quindi confrontare tutti i singoli elementi che le compongono.

Java e Visual Basic

Elaborazione distribuita Corba

Corba è uno standard multipiattaforma e multilinguaggio per l'interoperabilità dei componenti software progettato da un consorzio di produttori.

Corba è molto stabile ed efficiente.

J)RMI

Remote Method Invocation è un insieme di API che consentono l'interoperabilità di oggetti Java. Questa soluzione è valida in ambienti all-Java, ma ad essa si preferisce solitamente Corba.

VB)DCOM

Distributed Com è un'estensione dell'architettura COM di Microsoft per consentire l'elaborazione distribuita. La soluzione tecnica COM è meno stabile di Corba.

J)<java Interfaccia_grafica="Swing">Interfaccia utente Swing
Swing è l'estensione standard della piattaforma Java per l'interfaccia utente. Tramite il pluggable look&feel, è possibile presentare l'interfaccia dell'applicazione con aspetti

diversi che simulino l'aspetto dativo di più sistemi operativi come Windows, Unix, Macintosh, ed altri. In aggiunta è presente un look&feel, nativo Java, detto Metal.</java>

VB)Windows

Windows è dotato di una interfaccia utente molto nota che presenta i suoi vantaggi e svantaggi ma ha il pregio, senz'altro, di essere molto conosciuta agli utenti.

J)Modello ad oggetti Linguaggio completamente orientato agli oggetti. Supporto agli oggetti object-based.

VB)Nessun supporto all'ereditarietà, al poliformismo, alle interfacce.

J)Gestione dati JDBC Consente di connettersi ad un numero elevatissimo di database, sia nativi Java che legacy. E' presente un ponte per connettersi ad ODBC.

VB) DAO Supporto per un largo numero di formati di database; ODBC Architettura aperta per l'accesso ad un vasto numero di database server.

Accesso alle informazioni

L'accesso alle informazioni relative ad una piattaforma è fondamentale per lo sviluppatore. Solitamente, la documentazione fornita con un sistema di sviluppo non è mai sufficiente: il programmatore è sempre in cerca di informazioni. Queste possono essere reperite su libri, riviste o sul web.

Quale è la disponibilità di informazioni in relazione a Java e Visual Basic ?

<visualbasic Documentazione="Buona">Visual Basic

Sul Web esistono decine di siti dedicati a Visual Basic, oltre ovviamente al sito ufficiale della Microsoft. Qui si possono trovare articoli, porzioni di codice, suggerimenti utili per migliorare la conoscenza di questa piattaforma. Si noti però che sul sito Microsoft le informazioni presenti si riferiscono solo alla versione attuale dell'ambiente di sviluppo: gli utenti che non si sono aggiornati non avranno più informazioni relative alla propria versione nel momento in cui una nuova release si presenta sul mercato.

In aggiunta è possibile sottoscrivere la Microsoft Developer Network. Questo abbonamento (disponibile su più livelli), prevede, dietro compenso, la fornitura di un set di CD contenenti le ultime release dei programmi e tutte le informazioni relative ai sistemi Microsoft. </visualbasic>

<java Documentazione="Buona">Java

Anche per quanto riguarda Java, è possibile trovare sul Web siti dedicati allo sviluppo. Il numero di pagine Web dedicate è in costante aumento, come il supporto di terze parti. Il sito ufficiale della Sun è inoltre un importante fonte di informazioni, dove è possibile reperire informazioni di prima mano sulla tecnologia.

Inoltre, essendo Java uno standard supportato da diversi produttori software, è possibile riferirsi anche al loro supporto. Alcuni link importanti sono:

<http://www.sun.com>

<http://www.ibm.com/java>

<http://www.symantec.com>

<http://www.inprise.com>

La Sun, inoltre, mette a disposizione un sito dedicato agli sviluppatori dove è possibile accedere a pre-release della tecnologia legata a Java.

Anche Sun fornisce un programma di abbonamento per la distribuzione di pre-release agli sviluppatori, chiamato Developer Essentials. Le informazioni sono però comunque reperibili attraverso il sito della società. L'abbonamento ha il solo scopo di fornire questi rilasci, che in alcuni casi possono raggiungere i diversi megabyte di dimensione, su un formato comodo come il CD.

Diversi produttori, tra cui IBM, stanno unificando i programmi di certificazione con quelli della Sun. Parafrasando lo slogan di Java WORA (Write Once, Run Anywhere), questi programmi porteranno al SOCA (Studied Once, Certified Anywhere). </java>

La piattaforma Java e Visual Basic hanno quindi entrambi un vasto supporto, anche se sussistono alcune differenze: soprattutto a livello internet, Java è più diffuso: esistono molti più siti dedicati a Java che a VB; le informazioni dei programmi di abbonamento Sun sono disponibili gratuitamente su internet - MSDN è accessibile solo a pagamento; <OpenSource>la piattaforma Java, i suoi standard, molte delle applicazioni sviluppate su di essa sono open source.

Cosa significa open source ? Un'applicazione, o una tecnologia, open source è provvista di una licenza che ne consente la consultazione dei sorgenti.

La piattaforma Java, le sue estensioni, le JFA (Java Application Framework, vedi qui), ed altre applicazioni reperibili su internet sono fornite in formato compilato e comprendono i sorgenti. Questi sono una fonte preziosa di informazione: se la documentazione non è sufficiente, è possibile consultare i sorgenti per ottenere le informazioni che mancano. Questa importante caratteristica è presente solo su piattaforme aperte come Java. Visual Basic è uno standard proprietario Microsoft e non viene fornito con nessun tipo di codice sorgente di riferimento.</OpenSource>

Allo stato attuale non ci sono indicazioni che Microsoft possa rilasciare i sorgenti di tutto o parti di Visual Basic.

Utilizzo da parte dell'industria

Qual'è la diffusione di questi linguaggi nel mondo informatico e quale è la tendenza per il futuro?

<visualbasic Diffusione="Alta">Visual Basic è uno strumento molto utilizzato per lo sviluppo corporate negli USA e per la produzione di applicazioni e package da parte di ISV italiani. Visual Basic è legato alla diffusione di Windows. Nato come strumento per la produzione semplice di applicazioni Windows, ha presto rivestito lo strumento di scelta degli sviluppatori Windows. </visualbasic>In contrapposizione troviamo Borland Delphi, strumento basato su Pascal per la produzione rapida di applicazioni (RAD).

<java Diffusione="Alta">Java, d'altra parte, è un linguaggio di concezione più recente. Ha compiuto da poco i quattro anni di età, ma è stato fin da subito abbracciato dall'ambiente informatico come l'ultima rivoluzione informatica. (Uno studio di GartnerGroup - settembre 1999- indica che più del 60% delle grandi aziende entro il 2004 utilizzerà Java per qualche progetto).

La sua sintassi che lo rende simile al C++, ha attirato molti sviluppatori, tra i quali si possono identificare anche programmatori Visual Basic stanchi di districarsi tra le limitazioni di VB, la complessità di Windows e l'impotenza di risolvere da soli problemi pratici, per via della chiusura della piattaforma.

L'adozione di Java da parte dell'industria è in costante aumento. Questo per via di numerosi fattori: l'avvento del web ha originato la necessità di applicazioni internet-ready. Java è il linguaggio del web e offre un supporto completo per la produzione applicazioni

di questo tipo; Visual Basic è meno adatto allo scopo, non essendo nato con questo obiettivo;

Java piace agli sviluppatori: programmatori C++ frustrati dalla difficoltà del linguaggio e sviluppatori Visual Basic stanchi di utilizzare il vecchio Basic sono migrati a Java; molti produttori di software stanno producendo per Java strumenti di sviluppo, componenti e programmi di utilità che rendono sempre più appetibile lo sviluppo tramite questa piattaforma.

Allo stato attuale possiamo identificare Visual Basic come linguaggio già affermato e Java come tendenza per il futuro.</java>

Valutazione economica

L'impatto economico che implica l'adozione di una particolare tecnologia è un elemento da tenere in particolare considerazione. In particolare si deve valutare:

- il costo dell'acquisto degli strumenti di sviluppo;
- il costo dello sviluppo;
- il costo della consegna;
- il costo del mantenimento.

<visualbasic Costo="Licenze Microsoft">Costo di acquisto Visual Basic è disponibile in tre versioni: learning, professional ed enterprise. Per acquistare queste tre versioni è necessario spendere:
Learning \$109, Professional \$549, Enterprise \$1299. </visualbasic>

<java Costo="Gratuito">Il compilatore Java è gratuito: il JDK, come la documentazione e tutto il materiale relativo alle API aggiuntive, è scaricabile gratuitamente dal sito della Javasoft. Altre implementazioni di Java, come quella della IBM, sono scaricabili gratuitamente dai siti delle rispettive società.</java>
<IDE>Il compilatore Java non offre però alcune caratteristiche visuali presenti in Visual Basic: per ottenere un'ambiente di sviluppo di questo tipo è necessario acquistare ambienti di sviluppo di terze parti. Per una trattazione completa degli IDE disponibili per Java, consultare il sito della Javalobby. Riportiamo qui i costi indicativi di due dei più diffusi IDE per Java, Visual Café di Symantec e JBuilder di Borland.</IDE>

Symantec Visual Café 3.0 Standard \$ 99.95
Symantec Visual Café 3.0 Professional \$ 299.95
Symantec Visual Café 3.0 Database \$ 799.95
Symantec Visual Café 3.0 Enterprise \$ 2795.00
Borland JBuilder 3 Standard \$ 99.95
Borland JBuilder 3 Professional \$ 799.99
Borland JBuilder 3 Enterprise \$ 2499.00

I costi dei due prodotti sono abbastanza allineati tra di loro, mentre rileviamo una differenza con la piattaforma VB. Il costo maggiore delle versioni Enterprise non deve però stupire: Visual Basic in versione Enterprise non offre alcune funzionalità aggiuntive presenti negli IDE per Java che devono essere acquistati a parte.

Costo di sviluppo

La valutazione economica soggetta a maggiore attenzione solitamente è quella legata ai costi effettivi di sviluppo. Questa valutazione richiama le considerazioni espresse nel paragrafo tempi di sviluppo.

<visualbasic Costo_sviluppo="Prototipazione in tempi piu' brevi">Riassumendo, Visual Basic consente una prototipazione in tempi più brevi</visualbasic>, <java Costo_sviluppo="Tempo totale ridotto">mentre Java consente la produzione di applicazioni

complete e professionali, in un tempo totale inferiore.</java>I costi di sviluppo sono quindi immediatamente calcolabili.

Costo di consegna

Una volta terminato lo sviluppo del codice, viene il momento di consegnarlo al cliente o all'utente. Quali azioni comporta la consegna di un programma Visual Basic e di uno Java ? <visualbasic Distribuzione="Setup.exe">Il programma Visual Basic richiede il runtime VB, diverso per ogni versione e che occupa diversi megabyte di spazio. Questo programma (in formato DLL), deve essere installato su ogni macchina che dovrà eseguire il programma. Inoltre, se il progetto Visual Basic fa uso di controlli aggiuntivi, sarà necessario includere nell'installazione tutti i file e le librerie condivise necessarie al corretto funzionamento di questi controlli. Tutte queste operazioni sono alquanto complicate, in quanto lo sviluppatore non è sempre a conoscenza di tutti i file richiesti dai controlli aggiuntivi che utilizza o dei file di runtime di Visual Basic. Fortunatamente, in Visual Basic è presente una procedura che consente, in maniera automatica, di determinare tutte le dipendenze di un progetto VB e di generare il programma di installazione relativo. Sebbene questa procedura non funzioni sempre a dovere, essa costituisce comunque un notevole aiuto allo sviluppatore. </visualbasic>

<java Distribuzione="Jar">I programmi Java vengono invece distribuiti come file JAR. I file JAR (Java Archive File) sono compressi con il popolare algoritmo ZIP e contengono tutti i file binari che compongono il programma (comprese immagini, suoni ed altro) oltre che ad un file "manifesto" che consente di specificare le proprietà dei singoli file o dell'intero archivio. Nel file manifesto è inoltre possibile impostare una firma digitale. Il programma di installazione dovrà rendere accessibile all'utente i file JAR che compongono l'applicazione, compresi eventuali JavaBean o altri package utilizzati dall'applicazione. La struttura fortemente network-oriented di Java consente di installare questi file in un punto qualsiasi della rete (internet o intranet) e di effettuare il download dinamico dei soli file necessari (poichè variati o mancanti). </java>

Visual Basic e Java sono fortemente differenti per quanto riguarda l'installazione: mentre il primo necessita l'installazione in modo "tradizionale", il secondo può essere installato tramite una rete, centralizzandone la gestione (magari tramite un application server) e di conseguenza minimizzandone i costi.

Costo del mantenimento

Il costo di mantenimento identifica il costo totale legato alle variazioni da effettuare sul sistema. Lo sviluppo è spesso al 90% modifica di quanto già realizzato: questo è un fatto positivo, in quanto generalmente si tende a non reinventare sempre la ruota. In realtà, per poter mantenere in modo efficiente un sistema software, è necessario che il programma sia strutturato in modo efficiente (vedi riutilizzabilità del codice).

<java Mantenimento="Alta">Java, essendo ad oggetti, è più manutentabile di Visual Basic: questo porta a costi di mantenimento inferiori per il primo sistema ed a maggiori per il secondo.</java>

Riassunto del confronto

In questa sezione riassumiamo in una tabella le principali caratteristiche dei software evidenziando un voto, da 1 a 5, per ogni caratteristica.

Java Visual Basic

Qualità del codice 5 2

Tempi di sviluppo: dipende dall'applicazione.
 Riutilizzabilità del codice 5 3
 Ambiente internet/intranet 5 4
 Multiplatforma 5 1
 Prestazioni 3 4
 Qualità applicazioni realizzate 4 4
 Accesso alle informazioni 5 5
 Utilizzo da parte dell'industria 4 4
 Valutazione economica 5 4
 Totale 41 31

Conclusioni

Dopo questa lunga trattazione è il momento di tirare le somme. Qual'è la scelta migliore tra Java e Visual Basic? La risposta è: dipende. Per applicazioni semplici, basati in modo limitato su internet e legati strettamente a Windows ed alle sue evoluzioni, la scelta migliore è Visual Basic. Per quanto riguarda applicazioni complesse, internet/intranet based ed indipendenti dalla piattaforma, Java è la scelta migliore. Osservando i dati relativi all'adozione della piattaforma Java da parte dell'industria si può notare che senz'altro questa piattaforma riveste l'investimento migliore per il futuro.

Citazioni

Bruce Eckel, Thinking in Java, Foreword

"As another example, Visual Basic (VB) was tied to BASIC, which wasn't really designed to be an extensible language, so all the extensions piled upon VB have produced some truly horrible and un-maintainable syntax."

Bruce McKinney, A hardcore declaration of independence, Autore di Hardcore Visual Basic ha abbandonato il linguaggio in quanto troppo limitato, per utilizzare Delphi. "Visual Basic designers have chosen to pile more and more doodads on a weak foundation, knowing that doodads, not foundations, sell boxes."

<Cplusplus>Perchè il C++ è più complesso di Java.

I creatori di Java si sono sicuramente ispirati al C++ per produrre la loro creatura. Java è però più semplice del C++, perchè? Per prima cosa la sintassi, seppur molto simile, è più esplicita (extends, implements); in Java, inoltre, non è presente l'ereditarietà multipla. Questa caratteristica non deve sembrare una limitazione: in realtà molti dei vantaggi portati dall'ereditarietà multipla possono essere ottenuti con l'uso delle interfacce e del pattern delegation. </Cplusplus>
 </documento>

Articolo.dtd

» Descrizione: file DTD generato da XMLStudio a partire dal file
Articolo.xml.

» Directory: ./XMLStudio/Esempi

```
<!ELEMENT Basic ( #PCDATA ) >
<!ELEMENT Cplusplus ( #PCDATA ) >
<!ELEMENT Delega ( #PCDATA | Ereditarieta )* >
<!ELEMENT Ereditarieta ( #PCDATA ) >
<!ELEMENT IDE ( #PCDATA ) >
<!ELEMENT Object_based ( #PCDATA ) >
<!ELEMENT Object_oriented ( #PCDATA ) >
<!ELEMENT OpenSource ( #PCDATA ) >
<!ELEMENT Poliformismo ( #PCDATA ) >
<!ELEMENT Prestazioni ( #PCDATA ) >
<!ELEMENT Qualita_codice ( #PCDATA ) >
<!ELEMENT Sottoclassamento ( #PCDATA ) >
<!ELEMENT Tempi ( #PCDATA ) >
<!ATTLIST Tempi java NMTOKEN #IMPLIED >
<!ATTLIST Tempi visualbasic NMTOKEN #IMPLIED >
<!ELEMENT activex ( #PCDATA | applet | visualbasic )* >
<!ELEMENT applet ( #PCDATA ) >
<!ELEMENT byte-code ( #PCDATA ) >
<!ELEMENT documentazione ( #PCDATA ) >
<!ELEMENT documento ( #PCDATA | Basic | Cplusplus | Delega | IDE |
Object_based | Object_oriented | OpenSource | Poliformismo |
Prestazioni | Qualita_codice | Sottoclassamento | Tempi | activex |
applet | byte-code | documentazione | java | p-code | plug-in |
prestazioni | riutilizzabilita | visualbasic )* >
<!ELEMENT java ( #PCDATA ) >
<!ATTLIST java Autore CDATA #IMPLIED >
<!ATTLIST java Compilato NMTOKEN #IMPLIED >
<!ATTLIST java Costo NMTOKEN #IMPLIED >
<!ATTLIST java Costo_sviluppo CDATA #IMPLIED >
<!ATTLIST java Diffusione NMTOKEN #IMPLIED >
<!ATTLIST java Distribuzione NMTOKEN #IMPLIED >
<!ATTLIST java Documentazione NMTOKEN #IMPLIED >
<!ATTLIST java Interfaccia_grafica NMTOKEN #IMPLIED >
```

```

<!ATTLIST java Internet NMTOKEN #IMPLIED >
<!ATTLIST java Mantenimento NMTOKEN #IMPLIED >
<!ATTLIST java Multipiattaforma NMTOKEN #IMPLIED >
<!ATTLIST java Object_oriented NMTOKEN #IMPLIED >
<!ATTLIST java Prestazioni NMTOKEN #IMPLIED >
<!ATTLIST java Prima_pubblicazione CDATA #IMPLIED >
<!ATTLIST java Sicurezza NMTOKEN #IMPLIED >

<!ELEMENT p-code ( #PCDATA ) >

<!ELEMENT plug-in ( #PCDATA ) >

<!ELEMENT prestazioni ( #PCDATA ) >

<!ELEMENT riutilizzabilita ( #PCDATA ) >

<!ELEMENT visualbasic ( #PCDATA | visualbasic | visualcplusplus )*
>
<!ATTLIST visualbasic Costo CDATA #IMPLIED >
<!ATTLIST visualbasic Costo_sviluppo CDATA #IMPLIED >
<!ATTLIST visualbasic Diffusione NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Distribuzione NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Documentazione NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Internet NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Interpretato NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Multipiattaforma NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Object_oriented NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Riutilizzabilita NMTOKEN #IMPLIED >
<!ATTLIST visualbasic Sicurezza NMTOKEN #IMPLIED >

<!ELEMENT visualcplusplus ( #PCDATA ) >

```

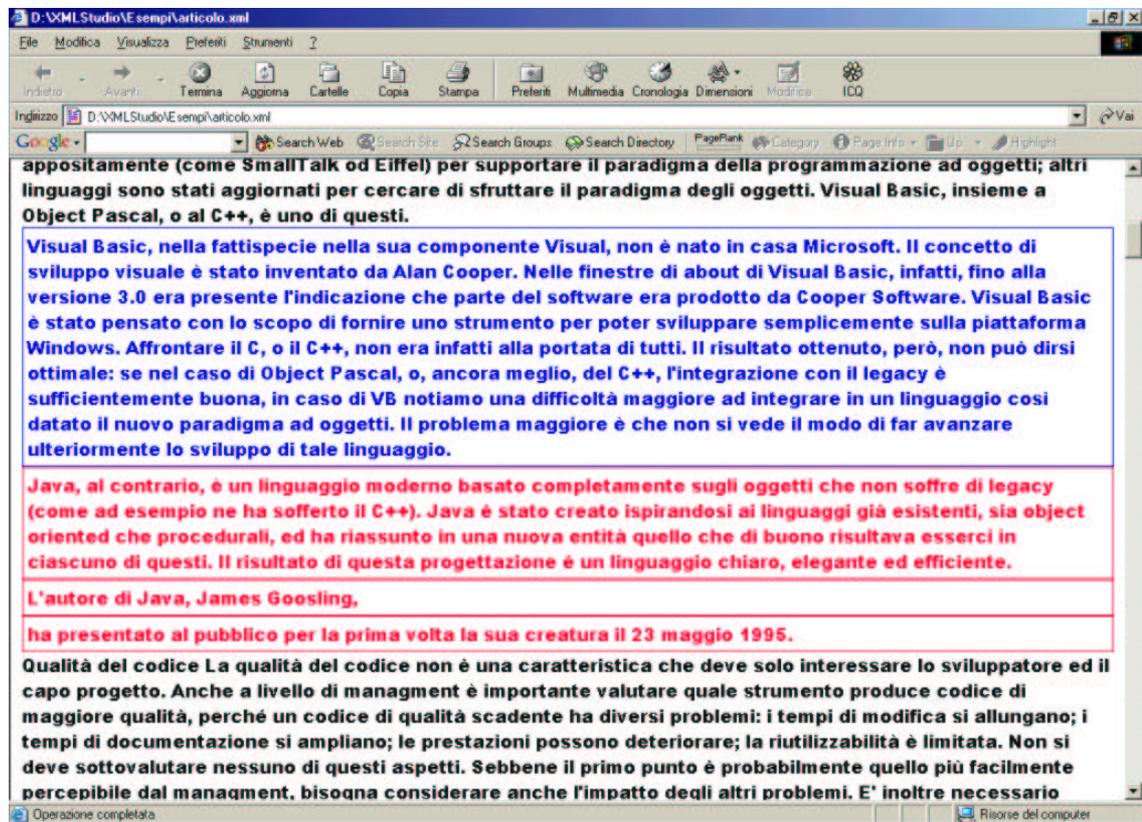
Articolo.css

» Descrizione: foglio di stile generato da XMLStudio per il file Articolo.xml.

» Directory: ./XMLStudio/Esempi

```
documento { color : #000000; font-family : Arial Black; }
linguaggi { color : #ff0066; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
visualbasic { color : #0000cc; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
java { color : #ff0033; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
```

Visualizzazione di Articolo.xml nel browser



XMLStudio.xml

» Descrizione: file dei marcatori XML utilizzato per etichettare secondo lo standard XML il file Articolo.xml.

» Directory: ./XMLStudio/Tag

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<informatica>
  <linguaggi>
    <java Produttore="Sun" Ultima_release="1.4.0"
Autore="James Gosling" Prima_pubblicazione="23 maggio 1995"
Object_oriented="Si" Internet="Si" Multipiattaforma="Si"
Sicurezza="Si" Prestazioni="Discrete" Compilato="Si"
Interfaccia_grafica="Swing" Documentazione="Buona"
Diffusione="Alta" Costo="Gratuito" Costo_sviluppo="Tempo totale
ridotto" Mantenimento="Alta" Distribuzione="Jar">
      <applet>
      </applet>
      <plug-in>
      </plug-in>
      <byte-code>
      </byte-code>
      <jvm Interpretato="Si">
      </jvm>
      <IDE>
        <JBuilder Produttore="Borland">
        </JBuilder>
        <Forte Produttore="Sun">
        </Forte>
        <VisualCafe Produttore="Symantec">
        </VisualCafe>
      </IDE>
    </java>
    <visualbasic Produttore="Microsoft"
Ultima_release="6.0" Object_oriented="No" Delega="Si"
Riutilizzabilita="Parziale" Internet="Parziale" Multipiattaforma="No"
Sicurezza="Scarsa" Interpretato="Si" Documentazione="Buona"
Diffusione="Alta" Costo="Licenze Microsoft"
Costo_sviluppo="Prototipazione in tempi piu' brevi"
Distribuzione="Setup.exe">
      <activex>
      </activex>
      <p-code>
      </p-code>
    </visualbasic>
    <Pascal Produttore="Borland" Ultima_release="5.5">
    </Pascal>
    <Cplusplus>
    </Cplusplus>
    <Perl>
    </Perl>
    <Proprieta>
      <Object_based>
        <Delega>
        </Delega>
      </Object_based>
      <Object_oriented>
        <riutilizzabilita>
```

```

        </riutilizzabilita>
        <Poliformismo>
        </Poliformismo>
        <Ereditarieta>
        </Ereditarieta>
        <Sottoclassamento>
        </Sottoclassamento>
        <Delega>
        </Delega>
    </Object_oriented>
    <Qualita_codice>
    </Qualita_codice>
    <Tempi visualbasic="6.0" java="1.4.0">
    </Tempi>
    <Prestazioni>
    </Prestazioni>
    <OpenSource>
    </OpenSource>
</Proprieta>
<Basic>
</Basic>
<SmallTalk>
</SmallTalk>
<Eiffel>
</Eiffel>
<visualcplusplus>
</visualcplusplus>
</linguaggi>
<sicurezza>
    <crittografia>
        <DSAKey>
        </DSAKey>
        <RSAKey>
        </RSAKey>
        <BinaryEncoding>
        </BinaryEncoding>
    </crittografia>
    <antivirus>
    </antivirus>
    <firewall>
    </firewall>
</sicurezza>
<software>
    <documentazione>
    </documentazione>
    <prestazioni>
    </prestazioni>
    <internet>
    </internet>
</software>
</informatica>

```

Esempio di articolo medico

Medicina.xml

» Descrizione: articolo medico trasformato in file XML al quale sono state aggiunte etichette XML riportate nel file Medicinatag.xml.

» Directory: ./XMLStudio/Esempi

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE documento SYSTEM "./medicina.dtd">
<?xml-stylesheet type="text/css" href="medicina.css"?>
<documento>

<titolo>L'artrosi senza cure</titolo>

<artrosi>Li chiamano i deparecidos dell'artrosi e sarebbero almeno
tre milioni e mezzo. Tanti sono gli italiani affetti da patologie
artrosiche di varia natura che non hanno mai consultato uno
specialista. E' l'allarme lanciato dai reumatologi italiani alla
vigilia del VI Congresso nazionale Limar-Anmar "Lotta alle malattie
reumatiche, la realizzazione di un'utopia", tenutosi di recente a
Siena. "Mettendo insieme tutti i pazienti afferenti ai vari centri
di reumatologia presenti nel nostro paese - afferma Roberto
Marcolongo, direttore dell'Istituto di Reumatologia e presidente
della Limar (Lega italiana per la lotta alle malattie reumatiche)-
si arriva più o meno ad un milione di pazienti. Degli altri 4
milioni circa di artrosici non si ha traccia". Molti di loro non si
curano.</artrosi>

L'odissea dei pazienti

<reumatismi reumatologi="scarsi">Ma il problema è che le malattie
reumatiche sono patologie tutt'altro che silenti. Ed è in genere il
sintomo dolore che spinge il paziente a cercare aiuto. La richiesta
di intervento infatti riguarda spesso il superamento della
sintomatologia dolorosa.

E in questo si viene in qualche modo accontentati, in genere
purtroppo solo in modo superficiale e sintomatico, a volte da
personaggi che poco hanno a che spartire con la medicina. "Il
risultato - prosegue Marcolongo - è un pellegrinaggio che dura
anni, portando ad un ritardo nella cura, che alla fine pesa
drammaticamente sulla progressione della malattia: una cura
tempestiva al contrario permetterebbe di bloccare la progressione
della malattia".</reumatismi>

<reumatismi disinformazione="si">La causa di tutto ciò va ricercata
nell'ignoranza più o meno diffusa tra la popolazione circa la
natura delle patologie reumatiche e sull'esistenza dello
specialista ad hoc, ma anche nella scarsa e disomogenea diffusione
dei centri reumatologici in Italia. Appena cinquanta sono infatti i
centri ospedalieri o universitari, carenti soprattutto al centro-
sud, quando non assenti del tutto, come avviene in Calabria.
</reumatismi>
```

Il ruolo del medico di famiglia

```
<medico>"La scarsità e la disomogenea distribuzione dei reumatologi nel nostro paese - sostiene Augusto Zaninelli, della Società italiana di medicina generale (Simg) - fa sì che gli specialisti di riferimento diventino l'ortopedico e il fisiatra, quando non l'anestesista per la terapia del dolore. </medico>
```

```
<strutture collaborazioni="si">In realtà il medico di famiglia è già in grado, grazie ai farmaci disponibili, di tenere sotto controllo almeno la metà delle patologie dolorose legate all'artrosi e alle malattie reumatiche. Ed è sempre compito della medicina generale, demandare agli specialisti solo quei pazienti che ne hanno veramente bisogno". É altamente improbabile che da un mese all'altro vengano improvvisamente attivati decine di centri reumatologici ex novo. Non resta dunque che puntare sulla creazione di un network assistenziale di specialisti (reumatologi, fisiatrici, ortopedici, terapisti del dolore). "Sarebbe auspicabile - sostiene Marcolongo - che le quattro società scientifiche (reumatologia, medicina generale, fisioterapia, ortopedia) arrivassero ad istituzionalizzare un rapporto di collaborazione, in virtù del quale su tutto il territorio nazionale sia possibile reperire delle Equipe composte da varie figure specialistiche che interagiscono tra loro".</strutture>
```

Interventi inutili e dolorosi

```
<medico reumatologo="si">
```

```
L'archittrave di tutto questo sistema dovrebbe essere rappresentata dal medico di medicina generale e dal reumatologo. "Troppo spesso - prosegue lo specialista - nei nostri ambulatori vediamo pazienti sottoposti inutilmente, e a costo di gravi disagi e sofferenze, ad interventi di riprotesizzazione d'anca o di ginocchio, quando si sarebbe potuto fare ancora molto, prima di ricorrere a queste soluzioni. É giusto che il paziente sappia, coadiuvato in questo dal medico di famiglia, che esiste lo specialista per i problemi reumatici e delle articolazioni". L'Anmar (Associazione nazionale malati reumatici) per il momento ha attivato due iniziative di tipo osservazionale per definire le dimensioni del problema malattie reumatiche.</medico>
```

La situazione in ambulatorio

```
<strutture ambulatorio="si">Lo studio mira a conoscere la tipologia dei pazienti che si presentano in una settimana presso l'ambulatorio del medico di famiglia (gli artrosici sarebbero secondi solo ai cardiopatici) e anche se, quelli affetti da patologie reumatiche, vengono indirizzati allo specialista di riferimento. Sono oltre cento le patologie reumatiche note; per ognuna di queste c'è ormai un ampio background di conoscenze e di possibilità di cure. </strutture>
</documento>
```

Medicina.dtd

» Descrizione: file DTD generato da XMLStudio a partire dal file

Medicina.xml.

» Directory: ./XMLStudio/Esempi

```
<!ELEMENT artrosi ( #PCDATA ) >

<!ELEMENT documento ( #PCDATA | artrosi | medico | reumatismi |
strutture | titolo )* >

<!ELEMENT medico ( #PCDATA ) >
<!ATTLIST medico reumatologo NMTOKEN #IMPLIED >

<!ELEMENT reumatismi ( #PCDATA ) >
<!ATTLIST reumatismi disinformazione NMTOKEN #IMPLIED >
<!ATTLIST reumatismi reumatologi NMTOKEN #IMPLIED >

<!ELEMENT strutture ( #PCDATA ) >
<!ATTLIST strutture ambulatorio NMTOKEN #IMPLIED >
<!ATTLIST strutture collaborazioni NMTOKEN #IMPLIED >

<!ELEMENT titolo ( #PCDATA ) >
```

Medicina.css

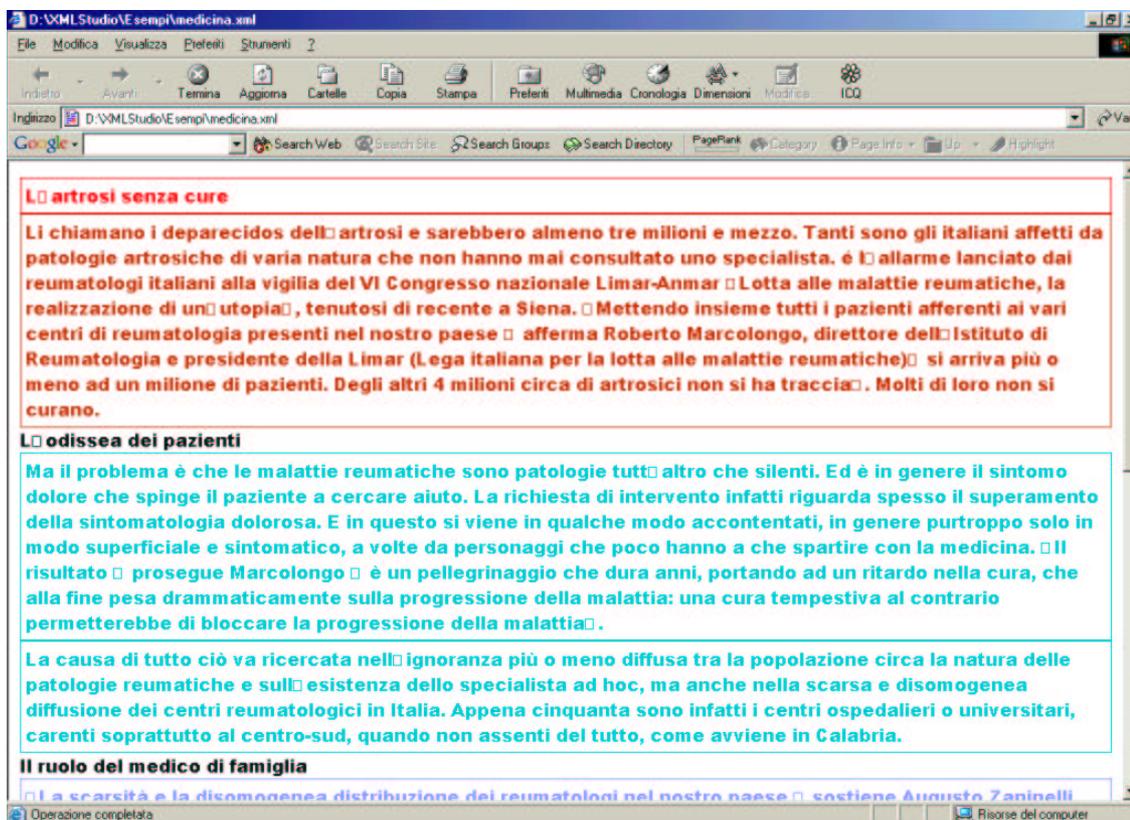
» Descrizione: foglio di stile generato da XMLStudio per il file

Medicina.xml.

» Directory: ./XMLStudio/Esempi

```
documento { color : #000000; font-family : Arial Black; }
titolo { color : #ff0000; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
autore { color : #33ff33; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
medicina { color : #6600cc; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
artrosi { color : #cc3300; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
reumatismi { color : #00cccc; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
medico { color : #9999ff; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
strutture { color : #009933; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
```

Visualizzazione di Medicina.xml nel browser



Medicinatag.xml

» Descrizione: file dei marcatori XML utilizzato per etichettare secondo lo standard XML il file Medicina.xml.

» Directory: ./XMLStudio/Tag

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<articolo>
  <titolo>
  </titolo>
  <autore>
  </autore>
  <argomenti>
    <medicina>
      <artrosi>
      </artrosi>
      <reumatismi disinformazione="si"
reumatologi="scarsi" terapia="pesante">
      </reumatismi>
      <medico>
      </medico>
      <strutture collaborazioni="si" ambulatorio="si"
prontosoccorso="si">
      </strutture>
    </medicina>
  </argomenti>
</articolo>
```

Esempio di articolo storico

Storia.xml

» Descrizione: articolo storico trasformato in file XML al quale sono state aggiunte etichette XML riportate nel file Storiatag.xml.

» Directory: ./XMLStudio/Esempi

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!DOCTYPE documento SYSTEM "./storia.dtd">
<?xml-stylesheet type="text/css" href="storia.css"?>
<documento>

<autore>Roberta Fidanzia*</autore>

<titolo>La Religione Popolare nel Medioevo</titolo>

<medioevo>Per poter comprendere il fenomeno della religione popolare nel Medioevo bisogna tener ben presente un presupposto importante: la religione del Medioevo è il Cristianesimo, come evidenzia Morghen nel suo libro Medioevo Cristiano.</medioevo>
<cristianesimo>Il Cristianesimo è una religione rivelata, completamente differente da quella che era la religione del passato classico. Nel mondo classico pagano, infatti, gli dei nascono dalla fantasia degli uomini che riflettono su di loro i propri vizi e le proprie virtù. Quindi sono "creature" dell'uomo. Nel Cristianesimo, invece, è esattamente il contrario: Dio crea l'uomo, la natura dell'uomo in quanto creatura di Dio è perfetta, ma l'uomo la corrompe con il peccato originale. Dio rivela all'uomo la Verità attraverso il Logos. Quindi il Cristianesimo è una religione rivelata.</cristianesimo>

<paganesimo>Si trova però a doversi inserire nel mondo pagano, mondo che è permeato di culti politeisti, di sacrifici e di riti magici legati alla natura. Era un mondo che cercava di tenere sotto controllo tutti gli eventi naturali, quali pioggia, grandine, terremoti, eruzioni vulcaniche, ecc.

Il mondo pagano, come sostiene Manselli nel libro Magia e stregoneria nel Medioevo, era caratterizzato da un "sincretismo religioso" "permeato dal magico" nel quale il cristianesimo avanza.

Nel vecchio mondo classico l'esigenza dell'uomo era quella di tenere sotto controllo le forze della natura e di poterle volgere a proprio vantaggio. Per questo si rivolgeva a personaggi, maghi, tempestarii, stregoni, che si dichiaravano in grado di intervenire sulle forze della natura attraverso riti e formule magiche.</paganesimo>

<cristianesimo>Il cristianesimo ha operato nella società pagana preesistente "con una forza dirompente - poiché - è monoteista ed ha un atteggiamento fortemente ostile contro la magia". Il cristianesimo introduce una rivoluzione: Dio è buono, Dio è Amore, l'azione di Dio è comunque Amore. Per questo l'uomo deve accettare la volontà di Dio, che è l'unico detentore del potere sulle forze naturali. L'uomo non può in nessun modo influire sugli eventi
```

naturali, perché l'unico che ha il potere su di essi è Dio. Per questo l'uomo deve accettare la volontà di Dio.</cristianesimo>

<criso>Con Gesù, Figlio di Dio, si assiste ad un cambiamento importantissimo di interpretazione della vita umana. Nel mondo pagano si chiedeva aiuto ai maghi per combattere le situazioni difficili, per migliorare la propria vita. Nel mondo cristiano si assiste ad una vera rivoluzione: Cristo si è sacrificato per salvare il mondo dal peccato originale, ha compiuto un enorme gesto d'amore. Ha accettato tutte le sofferenze con l'aiuto dell'amore. Cristo è l'esempio che bisogna seguire, quindi il dovere di un cristiano è di accettare le sofferenze, e non deve cercare di modificare quello che la vita gli ha presentato.</criso>
<amore>L'amore può risolvere tutto. "Ogni avversione viene annullata, resa inefficace, davanti al senso traboccante dell'amore. Questo è l'elemento a-magico del cristianesimo".</amore>

<chiesa>La Chiesa combatterà contro questi riti cercando in tutti i modi di distruggerli associandoli alle forze negative dei demoni.</chiesa>

<magia>La magia diventa quindi una specie di inganno diabolico che per lungo tempo ha allontanato gli uomini da Dio. Tutti gli dei pagani, per i cristiani, non sono divinità, ma sono diavoli.</magia>

<paganesimo>Il pagano convertito al cristianesimo, però, non dimentica il suo passato di pagano. Si rammenta del fatto che certe pratiche, che la Chiesa definisce superstizioni, certi riti, certe formule possono aiutarlo anche in casi disperati. Prima di abbandonarsi alla volontà del Dio cristiano, è facile, quindi, che nei casi di estremo bisogno, senta la necessità anche di valersi di questi riti della sua precedente esistenza pagana".</paganesimo>

<magia>Si assiste per tutto il Medioevo ad un ritorno del magico. Con la magia si riescono ad ottenere i risultati e si avverano i desideri che spesso Dio non esaudisce.

La magia, però, per avere il suo effetto ha bisogno di alcuni elementi: la presenza dell'esperto, ovvero il mago; lo svolgimento di riti precisi; ed, infine, l'uso di determinate formule magiche.

Il mago deve, quindi, conoscere il rito magico, saperlo eseguire e saper riconoscere le forze positive e le forze negative. Le persone che si rivolgono al mago ripongono in lui una fede: la fede nei suoi poteri.</magia> <chiesa>La Chiesa non poteva accettarlo.

Da qui nasce la polemica contro la magia da parte degli scrittori cristiani. Gli dei pagani decadono al rango di demoni. La Chiesa lotta per questo così accanitamente contro la magia.</chiesa>

<magia>Quindi esistono due posizioni nei confronti della magia: da una parte c'è la visione completamente cristiana che vuole abbandonarsi a Dio, alla Sua volontà; dall'altra c'è invece la posizione semi cristiana o quasi pagana che non si arrende a questa volontà, e vuole tentare di combatterla ricorrendo alle pratiche magiche.</magia>

Per questo già sant'Agostino condannava la magia, perché ricorreva alle forze diaboliche. Egli non negava l'esistenza della magia, ma

ne evidenziava il carattere malefico, negativo. "I demoni - per sant'Agostino - possono, infatti, compiere degli eventi sovrannaturali, ma non dominano tutto il sovrannaturale".

<demoni>I demoni non possono creare cose nuove, ma possono modificare l'apparenza delle cose create da Dio ingannando gli uomini. Ma "Dio è comunque più potente di loro e fa prodigi maggiori".

Si registra quindi un allontanamento da quelli che erano i riti e le cerimonie stabiliti dalla Chiesa. Il popolo ha bisogno di un rapporto più diretto con la divinità. Vengono ripresi vecchi riti pagani, modificati e adattati al Cristianesimo, per ingraziarsi la volontà del Signore. Si ritorna, dunque, all'antico, adattandolo alle nuove esigenze.</demoni>

<chiesa>La Chiesa, dunque, con il passare dei secoli, paradossalmente si impossessa di certi riti pagani cristianizzandoli. Per esempio il caso delle ordalie. Con esse "fu riconosciuto il fatto che Dio, giudice giusto ed onnipotente, dava ragione soprannaturalmente a colui che vinceva la prova". La Chiesa, dunque, si avvaleva di una procedura pagana per attestare la verità, ma la cristianizzava inserendola in una dimensione religiosa, "che serve ad annullare la necessità di ricorrere al paganesimo, alla magia". Queste esigenze nascono dal fatto che la mentalità dell'epoca era ancora pervasa di elementi magici e si può dire che ne avesse bisogno, per cui la Chiesa rende questi elementi liturgici. "La Chiesa elabora tutta una serie di preghiere che non solo mira a sacralizzare la prova, ma anche ad impedire che la vittoria avven[ga] con l'aiuto di mezzi magici".</chiesa>

<cristianesimo popolare="si">È qui ed in questo modo che si può cominciare a parlare di forme di religiosità popolare. Manselli mette in evidenza, nel suo libro La religiosità popolare nel Medioevo, come questo momento di passaggio da rito pagano a rito cristiano segni la nascita di quelle forme di religiosità popolare che caratterizzeranno tutto il Medioevo. La Chiesa stessa, in qualche modo, approva queste forme, questi riti. Dio è costretto a rivelarsi con queste preghiere. Questo è il nuovo elemento magico della preghiera.

Tutto ciò consentirà lo sviluppo di tutte quelle forme di religiosità popolare che si staccheranno dalla religiosità colta, quella dell'élite clericale. Questi due tipi di religiosità avranno due diverse evoluzioni, parallele e mai del tutto divise.
</cristianesimo>

Si può dire che si assiste in qualche modo al ricrearsi di quella dicotomia che si ritrova tra le pagine di Esiodo e di Omero, siamo nel VII - VIII secolo a. C., <esiodo>dove in Esiodo era evidenziato il duro lavoro quotidiano del popolo</esiodo> ed <omero>in Omero le grandi gesta degli eroi.</omero> Da ciò deriva che <esiodo>in Esiodo c'è la necessità di una religione vicina alle esigenze della pura quotidianità del popolo e quindi ecco la religione dei Misteri;</esiodo> mentre <omero>in Omero, che canta gli eroi, c'è la religione Olimpica.</omero>

<cristianesimo popolare="si" colto="si">Più di un millennio dopo, come è ben messo in evidenza dal Manselli nel suo libro La religione popolare nel Medioevo, si assiste ad una nuova dicotomia tra religione del popolo e religione dell'élite, religione popolare e religione colta.

Ma Manselli evidenzia molto bene anche un altro punto. La religiosità popolare e la religiosità colta sono due espressioni della stessa religione: il Cristianesimo.

Per quanto riguarda questo concetto si può fare un raffronto con quanto espresso da Croce nel suo libro *Poesia popolare e poesia d'arte*, edito da Laterza, II edizione del 1946, nel quale egli mette bene in evidenza come non esista per la poesia popolare e la poesia d'arte una distinzione netta. Croce combatte, infatti, quella corrente di studio che sosteneva che tra poesia d'arte e poesia popolare esiste una differenza qualitativa e oggettiva. La poesia d'arte sarebbe tecnica, storica, firmata e ragionata, mentre quella popolare sarebbe atecnica, astorica, anonima, spontanea. Croce evidenzia e dimostra come questa separazione sia in realtà falsa, in quanto entrambe le poesie sono poesie. Né una è migliore dell'altra. La differenza è psicologica, *simple things in simple words*, l'una si esprime con forme semplici, l'altra con forme più complesse. Per sottolineare questo concetto Croce riprende la distinzione tra buon senso e pensiero critico e sistematico: non c'è buon senso senza ragionamento e critica, e non esiste pensiero critico e sistematico privo di buon senso. Lo stesso vale per la poesia. Non esiste differenza qualitativa, oggettiva, ma psicologica. Questo concetto si può dimostrare prendendo come spunto due poesie diverse ma simili per contenuto. La poesia del Tommaseo su Matilde di Canossa che piange la perdita dell'amato compagno e la poesia popolare della tortora che piange la morte della compagna. Nell'una si assiste ad una tensione spirituale e sensuale, nell'altra è evidenziata la pena, la disperazione e la difficoltà nell'affrontare la vita senza la compagna.

Due forme diverse di manifestare lo stesso sentimento.

Alla stessa conclusione è giunto il Manselli nella distinzione tra religione popolare e religione colta: sono due forme diverse di manifestare la medesima religione, il Cristianesimo.

Per Manselli, dunque, la religiosità colta è quella che appartiene al clero, ai monaci e a pochi laici che hanno studiato i testi sacri. La religiosità popolare, invece, appartiene al popolo, a coloro che non hanno studiato i testi sacri, ma che si trovavano a vivere una realtà quotidiana diversa e con necessità diverse da quelle dell'altro gruppo.

<cristianesimo popolare="si">La religione popolare non è, però, una degenerazione della religione ufficiale, non è solo superstizione. È una forma diversa di religione, pur sempre collegata alle manifestazioni ufficiali.</cristianesimo>

Religiosità popolare e religiosità colta, come Manselli evidenzia nel suo libro *Il secolo XII: religione popolare ed eresia*, sono una realtà unitaria. Non vi è differenza di religione, la differenza sta "nella diversità delle sue forme." La religione colta "tende a sistemarsi in un'organizzazione concettuale dei dati offerti dalla 'parola' della rivelazione cristiana", mentre la religione popolare "recepisce la stessa 'parola' della rivelazione non come un complesso di realtà concettuali, ma come delle verità, garantite da un'autorità suprema".</cristianesimo>

<cristianesimo popolare="si">Ne deriva che, come Manselli sottolinea nel libro *Il soprannaturale e la religione popolare nel Medioevo*, nella religione popolare cristiana sopravvivono forme

precedenti di sentimento religioso, perché rimane costante la necessità di appagare un "bisogno di protezione, di aiuto, di conforto, che è forte se non imperioso nella difficile vita delle masse popolari del Medioevo".</cristianesimo>

<cristianesimo popolare="si" colto="si">La religiosità popolare, però, non si chiude su se stessa. Anzi si incontra spesso con quella colta. A far da tramite fra queste due forme si trova la classe sacerdotale, che, per quanto fosse ignorante, doveva di tanto in tanto incontrarsi con "esponenti di religiosità colta e, bene o male, ne dovevano risentire una qualche influenza". A loro volta questi esponenti della classe clericale colta venivano a contatto con la religiosità popolare "ne erano colpiti e reagivano, cercando di indirizzarla, modificarla o correggerla. Abbiamo, quindi, lungo tutto il Medioevo un nesso fra queste due forme di religiosità che la trasformazione dell'una finisce per comportare anche quella dell'altra".</cristianesimo>

<cristianesimo popolare="si">La religione popolare ha, comunque, una sua vitalità, "non è mai percepita come il risultato di un'acculturazione passiva ed inerte, ma come forza viva, capace alla lunga di modificare in modo decisivo le strutture della religione ufficiale".</cristianesimo>

* Dott.ssa in Scienze Politiche, Segretario del Medioevo Italiano Project

<bibliografia>Bibliografia:

CROCE, Benedetto, Poesia popolare e poesia d'arte, Bari, Laterza, 1946.

MANSELLI, Raoul, La religione popolare nel medioevo. (Sec. VI - XII), Giappichelli, 1974.

MANSELLI, Raoul, Magia e stregoneria nel Medioevo, Giappichelli, 1976.

MANSELLI, Raoul, Il secolo XII: religione popolare ed eresia, Jouvence, 1983.

MANSELLI, Raoul, Il soprannaturale e la religione popolare nel Medioevo, Edizioni Studium, 1985.

MANSELLI, Raoul, a cura di, La religiosità popolare nel Medioevo, Il Mulino, 1983.

MANSELLI, Raoul, San Francesco, Bulzoni Editore, Roma, 1980.

MANSELLI, Raoul, Studi sulle eresie del secolo XII, Giappichelli, 1953.

MORGHEN, Raffaello, Medioevo Cristiano, Laterza, 1994.

MORGHEN, Raffaello, Movimenti religiosi popolari ed eresie nel Medioevo, Sansoni, Firenze, 1955.

NARDI, Bruno, Studi sulla filosofia medievale, Roma, 1960.

NICOLOSI, Salvatore, Medioevo francescano, Borla, Roma, 1983
</bibliografia>

```
<note>(c) Roberta Fidanzia e StoriaOnline - in caso di utilizzo per  
studi, saggi, ricerche, tesi di laurea, occorre la citazione  
completa: Roberta Fidanzia, La religione popolare nel medioevo,  
http://www.storiaonline.org/a.religione.htm - Pubblicato il  
04.02.2001 </note>  
</documento>
```

Storia.dtd

» Descrizione: file DTD generato da XMLStudio a partire dal file Storia.xml.

» Directory: ./XMLStudio/Esempi

```

<!ELEMENT amore ( #PCDATA ) >
<!ELEMENT autore ( #PCDATA ) >
<!ELEMENT bibliografia ( #PCDATA ) >
<!ELEMENT chiesa ( #PCDATA ) >
<!ELEMENT cristianesimo ( #PCDATA | cristianesimo )* >
<!ATTLIST cristianesimo colto NMTOKEN #IMPLIED >
<!ATTLIST cristianesimo popolare NMTOKEN #IMPLIED >
<!ELEMENT cristo ( #PCDATA ) >
<!ELEMENT demoni ( #PCDATA ) >
<!ELEMENT documento ( #PCDATA | amore | autore | bibliografia |
chiesa | cristianesimo | cristo | demoni | esiodo | magia |
medioevo | note | omero | paganesimo | titolo )* >
<!ELEMENT esiodo ( #PCDATA ) >
<!ELEMENT magia ( #PCDATA ) >
<!ELEMENT medioevo ( #PCDATA ) >
<!ELEMENT note ( #PCDATA ) >
<!ELEMENT omero ( #PCDATA ) >
<!ELEMENT paganesimo ( #PCDATA ) >
<!ELEMENT titolo ( #PCDATA ) >

```

Storia.css

» Descrizione: foglio di stile generato da XMLStudio per il file Storia.xml.

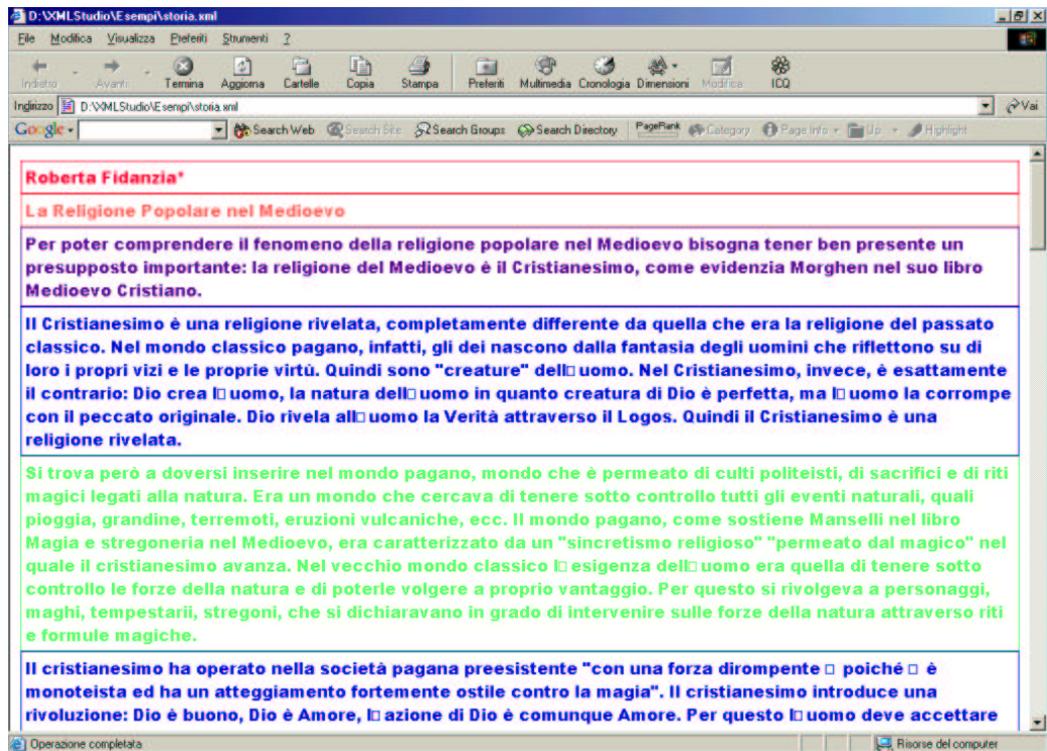
» Directory: ./XMLStudio/Esempi

```

documento { color : #000000; font-family : Arial Black; }
articolo { color : #990000; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
autore { color : #ff0033; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
titolo { color : #ff6666; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
bibliografia { color : #ff6633; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
note { color : #ff3300; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
religione { color : #6666ff; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
cristianesimo { color : #0000cc; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
amore { color : #9999ff; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
cristo { color : #6633ff; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
chiesa { color : #0033cc; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
demoni { color : #000099; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
medioevo { color : #660099; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
paganesimo { color : #66ff66; border : solid 1px; font-family :
Arial Black; padding: 4px; width: 100%; }
magia { color : #00ff33; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
esiodo { color : #00cccc; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }
omero { color : #99ffcc; border : solid 1px; font-family : Arial
Black; padding: 4px; width: 100%; }

```

Visualizzazione di Storia.xml nel browser



Storiatag.xml

» Descrizione: file dei marcatori XML utilizzato per etichettare secondo lo standard XML il file Storia.xml.

» Directory: ./XMLStudio/Tag

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<storia>
  <articolo www="si" rivista="no">
    <autore>
    </autore>
    <titolo>
    </titolo>
    <bibliografia>
    </bibliografia>
    <note>
    </note>
  </articolo>
  <argomenti>
    <religione>
      <cristianesimo popolare="si" colto="si">
        <amore>
        </amore>
        <criso>
        </criso>
        <chiesa>
        </chiesa>
        <demoni>
        </demoni>
      </cristianesimo>
      <paganesimo>
        <magia>
        </magia>
      </paganesimo>
    </religione>
    <medioevo>
    </medioevo>
    <classici>
      <autore>
        <esiodo>
        </esiodo>
        <omero>
        </omero>
      </autore>
    </classici>
  </argomenti>
</storia>
```

Bibliografia

- SERGE ABITEBOUL, PETER BUNEMAN, DAN SUCIU – *DATA ON THE WEB, FROM RELATIONS TO SEMISTRUCTURED DATA AND XML.*
- FABIO CIOTTI – *LA FORMA DEL TESTO ELETTRONICO, INTRODUZIONE A SGML E XML.*
- PAOLO DE LAZZARO – *XML, L'IPERTESTO SBARCA SUL WEB.*
- MARCO GIANNI – *CORSO SU XML.*
- DIEGO PERONI – *LINGUAGGI PER IL WEB.*
- MASSIMO CARLI – *JAVA E XML (DEV, NUMERO 84 – APRILE 2001).*
- ELLIOTTE RUSTY HAROLD – *PROCESSING XML WITH JAVA.*
- MAURO MOLINO – *XML & JAVA (MOKABYTE, NUMERO 29 – APRILE 1999).*
- GIUSEPPE CAPODIECI – *PARSING DI DOCUMENTI XML.*
- TONY MOBILI – *EDITORIALE (LOGIN, NUMERO 17 – LUGLIO/AGOSTO 1999).*
- PONTUS NORMAN – *A STUDY OF EXTENSIBLE MARKUP LANGUAGE (XML).*
- MARK JOHNSON – *PROGRAMMING XML IN JAVA (JAVA WORLD – LUGLIO 2000).*
- MARK JOHNSON – *XML FOR THE ABSOLUTE BEGINNER (JAVA WORLD – APRILE 1999).*
- MATTHEW ROBINSON, PAVEL VOROBIEV – *SWING.*
- ANDREA GIOVANNINI – *JAVA E XML (CAPITOLO 13 – MOKABOOK).*
- MARANGONI ROSANNA – *JAVAHHELP (MOKABYTE, NUMERO 28 – MARZO 1999).*
- SERGIO BENEDEUCE – *EDITOR XML A CONFRONTO (INTERNET NEWS, NUMERO 10 – OTTOBRE 1999).*
- MINERVINI CORRADO DANILO & BERGAMASCHINI EMILIANO – *XML ASCII DEL FUTURO (VERSIONE 1.1 – 1° GIUGNO 2001).*

Bibliografia Web

- EXTENSIBLE MARKUP LANGUAGE (XML): <http://www.w3.org/XML>
- THE EXTENSIBLE STYLESHEET LANGUAGE (XSL):
<http://www.w3.org/Style/XSL>
- DOCUMENT OBJECT MODEL: <http://www.w3.org/DOM>
- O'REILLY XML.COM: <http://www.xml.com>
- DEVX XML ZONE: <http://www.devx.com/xml/>
- OASIS (ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS): <http://www.oasis-open.org>
- PAGINA XML DI MICROSOFT:
<http://msdn.microsoft.com/xml/default.asp>
- JAVA LECTURE NOTES: <http://www.cafeaulait.org/course>
- JAVA PROGRAMMER'S FAQ: <http://pvdl.best.vwh.net/intro.html>
- TECNOLOGIA JAVA E XML: <http://java.sun.com/xml>
- PROGETTO XDIDATTICA: <http://www.ebookpertutti.com>
- XML PER TUTTI: <http://www.xmlpertutti.com>
- "SWING" BY MATTHEW ROBINSON AND PAVEL VOROBIEV:
<http://manning.spindoczone.com/sbe>
- PROCESSING XML WITH JAVA BY ELLIOTTE RUSTY HAROLD:
<http://www.cafeconleche.org/books/xmljava/chapters>

Home Page di XMLStudio

La presente tesi, la documentazione, i file sorgenti e l'applicazione XMLStudio sono scaricabili via Internet dalla home page del programma alla pagina <http://web.tiscali.it/xmlstudio>

Ringraziamenti

Ringrazio sentitamente la *Prof.ssa Paola Giannini* per la proposta di tesi, la fiducia e la preziosa assistenza durante tutto il suo sviluppo.

Un ringraziamento particolare al *Dott.Fabrizio Tambussa* per il beta-testing dell'editor XMLStudio e al *Dott.Giovanni Porcelli* per i suggerimenti legati all'applicazione e la realizzazione dell'esempio legato ai documenti RFC.

Ringrazio gli iscritti ai newsgroup *it.comp.java*, *comp.lang.java.help* e *comp.lang.java.programmer* per gli utili suggerimenti legati all'uso del linguaggio Java. Un grazie sentito anche a *Giovanni Negri*, *Simone Giulioni*, *Karim Farahat* e *Marco Zaino* per le utili indicazioni in laboratorio e a *Franco Dellavalle* per il cd-r realizzato.

Ringrazio infine ma soprattutto *mia madre*, *la mia famiglia* e poi *i miei amici* e tutte le persone vicine che mi hanno incoraggiato in questi lunghi anni di studio.